Este exemplar corresponde à redação final da Tese/Dissertação devidamente corrigida e defendida por: Marcelo Abdalla dos Reis

e aprovada pela Banca Examinadora.

Campinas, 16 de abril de 2003

COORDENADOR DE PÓS-GRADUAÇÃO
CPG-IC

Forense computacional e sua aplicação em segurança imunológica

Marcelo Abdalla dos Reis

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Forense computacional e sua aplicação em segurança imunológica

Marcelo Abdalla dos Reis

Janeiro de 2003

Banca Examinadora:

- Prof. Dr. Paulo Lício de Geus
  Instituto de Computação, UNICAMP (orientador)
- Prof. Dr. Carlos Maziero
  Centro de Ciências Exatas e de Tecnologia, PUC-PR
- Prof. Dr. Ricardo Dahab
  Instituto de Computação, UNICAMP
- Prof. Dr. Edmundo Madeira (suplente)
  Instituto de Computação, UNICAMP

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA CENTRAL DA UNICAMP

BIB ID 290979

R277f

Reis, Marcelo Abdalla dos
Forense computacional e sua aplicação em segurança imunológica / Marcelo Abdalla dos Reis --
Campinas, SP : [s.n.], 2003.

Orientador: Paulo Lício de Geus.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Redes de computação - Medidas de segurança. 2. Ciências forenses. 3. Auditoria - Processamento de dados. I. Geus, Paulo Lício de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 26 de fevereiro de 2003, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Carlos Alberto Maziero
PUC - PR

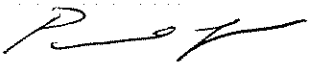
Prof. Dr. Ricardo Dahab
IC - UNICAMP

Prof. Dr. Paulo Lício de Geus
IC – UNICAMP

Forense computacional e sua aplicação em segurança imunológica

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Marcelo Abdalla dos Reis e aprovada pela Banca Examinadora.

Campinas, 26 de Fevereiro de 2003.



Prof. Dr. Paulo Lício de Geus
Instituto de Computação, UNICAMP
(orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Para meus pais, Raquel e meus amigos.

"O horizonte da perícia é a busca da verdade, tendo como meios para alcançá-la: o perito em seu reto procedimento, a tecnologia e o conhecimento científico."

Alberi Espindula - Perito Criminalístico

Agradecimentos

A Deus por mais uma conquista em minha vida.

A meus pais pelo apoio incondicional, educação e formação de meu caráter.

A Raquel pelo amor, carinho e, sobretudo, paciência.

A meus tios e tias que tanto me ajudaram nessa caminhada. Quando Campinas parecia longe e o Hawaii um grande sonho, vocês me receberam e apoiaram, ajudando a transformar Campinas em um lar e o Hawaii em realidade.

Ao professor Paulo Lício de Geus pela orientação e oportunidades ao longo deste trabalho.

Aos grandes amigos Diego e Fabrício pelo convívio, discussões sobre a pesquisa, "galhos quebrados"e, sobretudo, pela perpétua amizade.

Aos demais amigos do LAS — Curitiba, Cleymone, Edmar, Flávio, Jansen e João — pela amizade e discussões enriquecedoras.

Aos eternos amigos da graduação pelo incentivo e ajuda quando foi preciso.

Ao CNPq e à FAPESP pelo apoio financeiro essencial à execução deste trabalho.

Ao Deputado Federal Waldemir Moka pela ajuda financeira em algumas de minhas participações em congressos científicos, demonstrando sua consciência política relacionada à educação e à pesquisa científica.

Resumo

Soluções eficazes de detecção de intrusão continuam a ser perseguidas à medida que os ambientes computacionais tornam-se mais complexos e os atacantes continuamente adaptam suas técnicas para sobrepujar as inovações em segurança de computadores. É nesse sentido que a adoção de melhores modelos de segurança, que representem de maneira mais próxima as condições em que a maioria das redes de computadores se encontra (um ambiente hostil e sujeito a falhas), pode representar um passo na direção dessa busca por soluções eficazes de detecção de intrusão. A analogia entre segurança de computadores e o sistema imunológico humano constitui uma rica fonte de inspiração para o desenvolvimento de novos mecanismos de defesa, sejam algoritmos e técnicas de detecção de intrusão, políticas de segurança mais atentas à existência de falhas ou sistemas completos de segurança.

Em linhas gerais, quando um micróbio desconhecido é identificado pelo sistema imunológico humano, um mecanismo de aprendizado é aplicado com o intuito de adquirir conhecimento sobre o invasor e gerar um conjunto de células de defesa especializadas em sua detecção. Desse modo, a memória imunológica é atualizada autonomamente, permitindo a identificação futura mais eficiente do mesmo micróbio. Com o objetivo de mapear essa característica de aprendizado para um sistema de segurança de computadores, baseado no modelo imunológico humano, este trabalho apresenta um estudo no sentido de se entender como utilizar a forense computacional, de maneira automatizada, na identificação e caracterização de um ataque. Como resultados desta pesquisa são apresentados a modelagem de um sistema de segurança imunológico, uma arquitetura extensível para o desenvolvimento de um sistema automatizado de análise forense e um protótipo inicial que implementa parte dessa arquitetura.

Abstract

The challenge faced by intrusion detection is the design of more effective solutions as long as computer systems become more complex and intruders continually adapt their techniques to overcome the inovations on computer security. In this sense the adoption of better security models, that closely resembles the conditions in which most of computer networks are (a hostile and flawy environment), may represent a step towards the design of better solutions to intrusion detection. The analogy between computer security and the human immune system provides a rich source of inspiration to the development of new defense strategies, might it be intrusion detection algorithms and techniques, security policies more conscious about the existance of flaws or integrated security systems.

When an unknown microbe is identified by the human immune system, a learning mechanism is applied in order to aquire knowledge about the intruder and to generate a set of defense cells specialized in its detection. In this way, the immune memory is autonomously updated, allowing a more efficient detection of the same microbe in the future. In order to map this learning feature to a computer security system, based on the human immune model, this dissertation presents a research towards the understanding of how to use computer forensics, in an automatic fashion, to identify and characterize a computer attack. As results to this research are presented the desing model to a computer immune security system, an extensible architecture to the development of an automated forensic analyser and a prototype that implements part of this architecture.

Sumário

Agradecimentos xv

Resumo xvii

Abstract xix

1 Introdução 1
- 1.1 Escopo e objetivo do trabalho . . . 4
- 1.2 Nota sobre a utilização dos exemplos . . . 5
- 1.3 Organização do trabalho . . . 6

2 Detecção de intrusão 9
- 2.1 Conceitos da detecção de intrusão . . . 9
  - 2.1.1 Arquitetura . . . 10
  - 2.1.2 Estratégia de monitoramento . . . 10
  - 2.1.3 Tipo de análise . . . 11
  - 2.1.4 Momento da análise . . . 12
  - 2.1.5 Objetivo da detecção . . . 13
  - 2.1.6 Estratégia de controle . . . 13
- 2.2 Técnicas de análise . . . 13
  - 2.2.1 Técnicas de detecção por mau uso . . . 13
    - 2.2.1.1 Sistemas de produção/especialistas . . . 14
    - 2.2.1.2 Abordagens baseadas em transição de estados . . . 14
    - 2.2.1.3 Recuperação de informações para análise em modo *batch* . 18
    - 2.2.1.4 Detecção por modelagem (*model-based*) . . . 19
  - 2.2.2 Técnicas de detecção por anomalia . . . 19
    - 2.2.2.1 Modelo original de Dorothy Denning . . . 19
    - 2.2.2.2 Análise quantitativa . . . 20
    - 2.2.2.3 Medidas estatísticas . . . 22
    - 2.2.2.4 Medidas estatísticas não-paramétricas . . . 23

2.2.2.5 Abordagens baseadas em regras 24
2.2.2.6 Redes neurais 25
2.2.3 Esquemas alternativos de detecção 25
2.2.3.1 Abordagens baseadas em sistemas imunológicos 26
2.2.3.2 Algoritmos genéticos 26
2.2.3.3 Detecção baseada em agentes 27
2.2.3.4 Mineração de dados (*data mining*) 28
2.3 Conclusão 29

3 Imunologia computacional **31**
3.1 O sistema imunológico humano 31
3.1.1 Organização estrutural 31
3.1.2 Resposta imunológica 32
3.2 Analogias entre o sistema imunológico e a segurança de computadores . . . 36
3.3 Trabalhos correlatos 38
3.3.1 Universidade do Novo México 38
3.3.1.1 Princípios de um sistema imunológico computacional . . . 39
3.3.1.2 Um método de detecção de intrusão baseado em uma máquina 41
3.3.1.3 Um sistema de detecção de intrusão baseado em rede . . . 42
3.3.1.4 Um algoritmo distribuído de detecção de mudanças 44
3.3.1.5 Um método para introdução intencional de diversidade em sistemas computacionais para reduzir vulnerabilidades . . 45
3.3.2 Sistema imunológico computacional para detecção e análise de vírus de computador 45
3.3.3 Modelo imunológico artificial para detecção de intrusão baseada em rede 47
3.3.4 Sistema de detecção de intrusão baseado em agentes 50
3.4 Conclusão 51

4 Arquitetura de um sistema de segurança imunológico **53**
4.1 Um modelo de IDS híbrido baseado no sistema imunológico humano 54
4.2 Analogias entre o sistema de defesa humano e o modelo de IDS imunológico 57
4.3 ADenoIdS: Uma implementação do modelo de IDS imunológico 59
4.4 O papel da forense computacional no modelo de IDS imunológico 62
4.5 Conclusão 63

5 Forense computacional **65**
5.1 O que é forense computacional? 65
5.2 *Modus operandi* do invasor 67

5.3 Evidências digitais 70
5.3.1 Dispositivos de armazenagem da CPU 71
5.3.2 Memória de periféricos 72
5.3.3 Memória principal do sistema 72
5.3.4 Tráfego de rede 73
5.3.5 Estado do sistema operacional 74
5.3.5.1 Processos em execução 74
5.3.5.2 Conexões de rede 76
5.3.5.3 Estado das interfaces de rede 76
5.3.5.4 Usuários logados 76
5.3.5.5 Tabelas e *caches* 77
5.3.5.6 Módulos de *kernel* carregados 77
5.3.6 Dispositivos de armazenagem secundária 78
5.3.6.1 Áreas não acessíveis através de um sistema de arquivos . . 78
5.3.6.2 Sistema de arquivos 79
5.3.7 Arquivos de configuração 82
5.3.7.1 *Scripts* de inicialização 82
5.3.7.2 Relações de confiança e controle de acesso 83
5.3.7.3 Usuários e grupos 84
5.3.7.4 Serviços de rede 85
5.3.7.5 Tarefas agendadas 86
5.3.7.6 Sistemas de segurança 86
5.3.8 Arquivos de *log* 86
5.4 Correlação de evidências 91
5.5 Estrutura geral do processo de investigação forense 92
5.5.1 Considerações e inteligência preliminares 95
5.5.2 Planejamento 96
5.5.3 Estabilização do sistema e decisões iniciais 96
5.5.4 Coleta, autenticação, documentação e preservação de material para análise 99
5.5.5 Análise 102
5.6 Conclusão 103

6 Automatização do processo de análise forense **105**
6.1 Introdução 105
6.2 Uma arquitetura extensível de análise forense 108
6.2.1 Coleta de informações 109
6.2.2 Busca e extração de evidências 110

6.2.3 Análise das evidências encontradas 112
6.3 Um protótipo inicial 116
6.4 Aplicação da arquitetura de análise forense no modelo de IDS imunológico 117
6.4.1 Aplicação específica no sistema ADenoIdS 118
6.5 Conclusão 118

7 Conclusão **121**
7.1 Conclusões gerais 121
7.2 Considerações finais e experiência adquirida 123
7.2.1 Forense computacional 123
7.2.2 Imunologia computacional 124
7.3 Contribuições efetivas do trabalho 125
7.4 Trabalhos futuros 126

Referências bibliográficas **129**

A Detalhes particulares sobre a extração e análise das informações de um sistema computacional **141**
A.1 Extração e reprodução dos dados da memória de vídeo 141
A.2 Acesso às informações da memória principal do sistema 142
A.2.1 Processo de *dump* da memória e análise dos dados extraídos 142
A.2.2 Geração e análise de *core files* 143
A.3 Captura e análise do tráfego de rede 144
A.4 Acesso às informações dos processos em execução 146
A.4.1 Atividade do sistema 146
A.4.2 Listagem e detalhes dos processos 147
A.4.3 Arquivos acessados 147
A.4.4 Interface `/proc` 148
A.5 Acesso às informações sobre o estado das conexões de rede 151
A.6 Acesso às informações sobre o estado das interfaces de rede 151
A.7 Acesso às informações sobre os usuários logados 152
A.8 Acesso às tabelas e *caches* mantidas pelo sistema operacional 152
A.8.1 Tabela e *cache* de roteamento 152
A.8.2 *Cache* de resolução de endereços MAC 153
A.8.3 *Cache* de resolução de nomes 153
A.9 Investigação dos módulos de *kernel* carregados 154
A.10 Acesso e recuperação de informações escondidas ou "deletadas" 156
A.10.1 Estrutura e organização do disco 156
A.10.2 Possíveis locais para se esconder informações no disco 159

A.10.3 Extração e recuperação de informações escondidas ou "deletadas" . 160
A.11 Preparação do sistema de arquivos para o processo de análise forense . . . 163
A.12 Extração de informações das estruturas internas do sistema de arquivos . . 165
A.12.1 Acesso às informações do *inode* 167
A.12.2 Acesso às entradas de diretório 168
A.12.3 *Mactimes* 169
A.12.4 Recuperação de dados "deletados" do sistema de arquivos 171
A.13 Análise das configurações do sistema 173
A.14 Busca e análise dos executáveis e bibliotecas suspeitas 174
A.14.1 Localização dos executáveis 174
A.14.2 Identificação dos executáveis e bibliotecas suspeitas 174
A.14.3 Análise dos executáveis e bibliotecas 175
A.15 Transmissão das informações coletadas através da rede 177
A.16 Procedimentos de imagem dos discos suspeitos 178
A.16.1 Imagem em arquivo de todo o disco 179
A.16.2 Imagem em arquivo das partições separadas 180
A.16.3 Espelhamento 181

B Conjunto de ferramentas para análise forense em sistemas computacionais 183
B.1 Sistema de análise 183
B.2 Ferramentas auxiliares 185
B.3 Ferramentas para utilização no sistema suspeito 185
B.4 Utilitários de forense 186
B.4.1 *The Coroner's Toolkit* (TCT) 186
B.4.2 TCTUTILs 191
B.4.3 *The @stake Sleuth Kit* (TASK) 192
B.4.4 *Autopsy Forensic Browser* (AFB) 193
B.4.5 ForensiX 195
B.4.6 *New Technologies Incorporated* (NTI) 196
B.4.7 EnCase 198

C Standardization of computer forensic protocols and procedures 201
C.1 Abstract 201
C.2 Introduction 201
C.3 Computer Forensic Science 203
C.4 Standardization Model 204
C.4.1 Legal Standards Class 206
C.4.2 Technical Standards Class 208

C.4.2.1 Technical Principles 208
C.4.2.2 Policies of Analysis 210
C.4.2.3 Techniques and Solutions 211
C.5 Further Debate 212
C.5.1 Scientific Community Acceptance 212
C.5.2 General Purpose Discipline 212
C.5.3 *Proven Correct* Tools 212
C.5.4 Legal and Technical Standards Relationship 213
C.5.5 Standardization Level 213
C.6 Conclusions 213
C.7 References 215

D Aspectos de utilização e configuração do AFA **217**
D.1 Manual de utilização do `afa_server` 217
D.2 Manual de utilização do `gather_target` 219
D.3 Arquivos de configuração do AFA 221
D.3.1 Arquivo `afa.cf` 221
D.3.2 Arquivo `paths.cf` 223
D.3.3 Arquivo `gather.rules` 224
D.3.4 Arquivo `plugins.cf` 224
D.4 *Plugins* implementados 225
D.4.1 Manual de utilização do `ps_analyser` 225
D.4.2 Arquivos de configuração do `ps_analyser` 226
D.4.3 Manual de utilização do `packet_analyser` 230

Glossário **235**

Lista de Tabelas

4.1 Analogias entre os componentes do modelo de IDS imunológico e o sistema de defesa humano. 58

5.1 Relação entre a habilidade do invasor e a quantidade de evidências deixadas. 69
5.2 Resumo dos principais arquivos e diretórios que comumente contêm ou representam evidências de uma intrusão (continua na próxima página). . . 80
5.3 Principais arquivos de *log* encontrados durante a análise forense. 88
5.4 Relação entre o método de coleta e análise e o nível de esforço empregado para proteger as evidências e evitar a execução de código hostil. 99

A.1 Conteúdo de um *inode*. 166
A.2 Conteúdo de uma entrada de diretório. 169

Lista de Figuras

2.1 Diagrama de um IDS híbrido genérico. 12
2.2 Diagrama de transição de estados. 15
2.3 Arquitetura do AAFID. 27

3.1 A detecção é consequência da reação entre estruturas químicas complementares. Em (a) o receptor de célula T reage com uma molécula MHC contendo um peptídeo de antígeno. Em (b) o receptor de célula B reage diretamente com um antígeno. 34
3.2 Ciclo de vida de um detector do sistema LISYS. 43
3.3 Sistema imunológico computacional de Kephart. 46
3.4 Arquitetura do modelo imunológico artificial de Kim e Bentley para detecção de intrusão baseada em rede. 48
3.5 Funcionamento do modelo imunológico artificial de Kim e Bentley. 49

4.1 Modelo de IDS híbrido baseado no sistema imunológico humano. 55
4.2 Analogia entre os sistemas inato e adaptativo e o modelo de IDS imunológico. 57

6.1 Arquitetura extensível para um sistema automatizado de análise forense. . 108
6.2 Exemplos gerais de possíveis evidências a serem procuradas. 111

A.1 Partições estendidas e lógicas. 157
A.2 Representação das estruturas internas de um sistema de arquivos, referentes ao arquivo `/file.txt` (o *inode* 2 é alocado ao diretório `/`, cujas entradas de diretório estão armazenadas no bloco de dados 511, e o *inode* 10 é alocado ao arquivo `file.txt`, cujas informações estão no bloco de dados 1000). . . 171
A.3 Fluxo das informações voláteis desde a coleta no sistema suspeito até seu armazenamento no sistema de análise. 178

B.1 Unidade portátil de investigação forense da Forensic-Computers.com. . . . 184
B.2 Mapa dos blocos reconhecidos pelo programa `lazarus`. 190
B.3 Navegação pelos blocos interpretados pelo programa `lazarus`. 191

B.4 Exemplo da interface provida pelo *Autopsy Forensic Browser*. 194
B.5 Exemplo da interface provida pelo ForensiX. 195
B.6 Exemplo da interface provida pelo EnCase. 198

C.1 Standardization model. 206

Capítulo 1

Introdução

Não há dúvidas de que o crescente volume e diversidade de informações disponíveis através de sistemas computacionais, isolados ou interconectados através de redes locais ou públicas como a Internet, teve um profundo impacto positivo na eficiência e modo de atuação das instituições em todo o mundo. Entretanto, esse aumento na disponibilidade de informações trouxe consigo uma crescente dependência dos recursos computacionais e de comunicações, ao ponto que qualquer interrupção de funcionamento ou acesso não-autorizado pode resultar em efeitos negativos dramáticos nas operações dessas instituições. Dada essa situação, a busca por um sistema computacional seguro tem sido constantemente objeto de diversos estudos.

De maneira prática e abrangente, Garfinkel e Spafford [34] definem um sistema computacional seguro como sendo aquele que se comporta de maneira esperada. Nesse caso, é importante que o comportamento esperado do sistema seja bem definido e formalizado dentro da política de segurança da organização, além da regulamentação sobre como a organização gerencia, protege e disponibiliza seus recursos.

Uma definição mais formal de segurança de computadores baseia-se na confidencialidade, integridade e disponibilidade dos recursos do sistema [86]. A confidencialidade determina que as informações sejam acessíveis somente aos usuários autorizados para tal, a integridade requer que as informações permaneçam inalteradas por acidentes ou tentativas maliciosas de manipulação, e a disponibilidade significa que o sistema computacional deve funcionar adequadamente, sem degradação de acesso, provendo recursos aos usuários autorizados quando eles necessitarem.

De fato, a definição de segurança em sistemas computacionais é bastante flexível e dependente de características próprias do sistema e da organização. Em resumo, um sistema seguro protege seus dados e recursos contra acesso não-autorizado, interferência e bloqueio de uso [55]. Desse modo, com base nas definições apresentadas para segurança de computadores, uma intrusão ou incidente de segurança pode ser caracterizada como

qualquer conjunto de ações que tentem comprometer a integridade, confidencialidade ou disponibilidade de um recurso computacional [39]. Ou ainda, de forma mais abrangente, uma intrusão é uma violação de normas ou procedimentos estabelecidos pela política de segurança adotada.

Na defesa de um sistema computacional, inúmeras tecnologias podem ser aplicadas. *Firewalls*, tecnologias de cifragem e autenticação, ferramentas de análise de vulnerabilidades, entre outras soluções, podem oferecer maior segurança. Do ponto de vista teórico, é possível aumentar a segurança de um sistema observando alguns critérios: é necessário especificar e implantar corretamente uma política de segurança, implementar corretamente os programas e configurar adequadamente o sistema. Porém, na prática, observa-se que as políticas de segurança, as implementações dos programas e as configurações dos sistemas podem conter falhas, tornando a segurança imperfeita [98, 107].

Mesmo quando um sistema computacional está equipado com rigorosos procedimentos de autenticação e controle de acesso, ele ainda está suscetível a ação de usuários maliciosos que exploram as falhas do sistema e utilizam truques de engenharia social (obtenção de senhas de usuários, fazendo-se passar pelo administrador do sistema, por exemplo). Os sistemas computacionais que não possuem conexões com redes públicas também estão vulneráveis, pois podem sofrer ações de usuários internos que abusam de seus privilégios. Dada a ameaça constante de um incidente de segurança, o estabelecimento de uma segunda linha de defesa na forma de um sistema de detecção de intrusão (IDS — *Intrusion Detection System*) pode ser uma atitude sensata [37].

A detecção de intrusão pode ser definida como o processo de monitoramento dos eventos ocorridos em um sistema computacional ou rede, em busca de sinais que indiquem problemas de segurança [3]. Um sistema de detecção de intrusão é, portanto, um sistema que automatiza esse processo de monitoramento. De acordo com o método de análise, a detecção de intrusão pode ser classificada em duas categorias: detecção de intrusão por anomalia e detecção por mau uso. Na deteção por anomalia, uma intrusão representa um desvio no comportamento normalmente observado no sistema, de modo que um IDS por anomalia conhece o perfil de comportamento usual do sistema e busca por anomalias em relação a esse perfil [3]. Por outro lado, a detecção por mau uso representa o conhecimento sobre o comportamento impróprio ou inaceitável e procura localizá-lo diretamente [3]. Uma terceira abordagem é o emprego de ambos os métodos de análise em um sistema híbrido de detecção.

A pesquisa na área da detecção de intrusão tem produzido propostas de soluções baseadas nas mais variadas estratégias, muito embora ainda exista uma distância a ser percorrida entre os aspectos teóricos e práticos. Soluções eficazes de detecção de intrusão continuam a ser perseguidas à medida que os ambientes computacionais tornam-se mais complexos e os atacantes continuamente adaptam suas técnicas para sobrepujar as ino-

vações em segurança de computadores [37]. É nesse sentido que a adoção de melhores modelos de segurança, que representem de maneira mais próxima as condições em que a maioria das redes de computadores se encontra — um ambiente hostil e sujeito a falhas, pode representar um passo na direção dessa busca por soluções eficazes de detecção de intrusão.

É possível encontrar na natureza um modelo de defesa que apresenta uma série de características desejáveis a um sistema de segurança: o sistema imunológico humano. O sistema de defesa do corpo humano, por ser capaz de garantir a sobrevivência de um indivíduo durante cerca de 70 anos, mesmo que ele se depare, a cada dia, com bactérias e vírus potencialmente mortais, apresenta um paralelo bastante forte com a segurança de sistemas computacionais.

Os trabalhos iniciais em torno desse paralelo concentravam-se em mecanismos isolados do sistema imunológico e como eles poderiam ser aplicados para melhorar a segurança de um sistema. Mais recentemente, as pesquisas passaram a considerar a estrutura de funcionamento do sistema imunológico como modelo de desenvolvimento de um sistema de segurança, baseando-se em uma série de princípios característicos do sistema de defesa do corpo humano. Entretanto, a maioria dos esforços concentra-se no desenvolvimento de sistemas de detecção de intrusão por anomalia, o que explora apenas uma parte do modelo oferecido pelo sistema imunológico humano.

Em linhas gerais, o sistema imunológico monitora o corpo humano contra a presença de proteínas estranhas àquelas normalmente encontradas. Essa característica é bastante semelhante à detecção de intrusão por anomalia. Entretanto, as características do sistema imunológico relacionadas à memorização e detecção específica e eficiente de agentes patogênicos conhecidos representam claramente propriedades da detecção de intrusão por mau uso. Através dessas características, o sistema imunológico pode armazenar informações sobre patógenos com os quais já teve contato, permitindo uma reação mais eficiente a novos ataques de um invasor conhecido. Além disso, o sistema imunológico adiciona uma melhoria a suas propriedades relacionadas à detecção de intrusão por mau uso — ele pode autonomamente alterar sua base de assinaturas de patógenos conhecidos (memória imunológica).

Essa característica adaptativa, relacionada à detecção por mau uso, pode ser introduzida em um sistema de detecção de intrusão através da conversão das evidências deixadas por um ataque em uma assinatura que especificamente o identifique. A geração de assinaturas de ataques desconhecidos pode ser realizada através da análise minuciosa de suas evidências, aplicando-se técnicas e conceitos de forense computacional. No entanto, o processo de análise forense executado para a geração de assinaturas deve ser automatizado, de modo a permitir que a base de conhecimento de um IDS por mau uso possa ser alterada autonomamente, de maneira análoga ao sistema imunológico humano.

A partir desta motivação, o escopo e objetivo deste trabalho são apresentados na seção seguinte. Algumas considerações quanto aos exemplos de técnicas e ferramentas empregadas nesta pesquisa são apresentadas na Seção 1.2 e a Seção 1.3 descreve a organização deste trabalho.

1.1 Escopo e objetivo do trabalho

Esta dissertação de mestrado é parte de um projeto de pesquisa desenvolvido pelo grupo de imunologia computacional do Laboratório de Administração e Segurança de Sistemas do Instituto de Computação da Universidade Estadual de Campinas (LAS-IC-UNICAMP). O grupo de imunologia computacional do LAS-IC-UNICAMP, sob orientação do Prof. Dr. Paulo Lício de Geus, é composto pelo autor desta dissertação, pelo então aluno de mestrado Diego de Assis Monteiro Fernandes e pelo então aluno de doutorado Fabrício Sérgio de Paula.

O projeto de pesquisa desenvolvido pelo grupo de imunologia computacional do LAS-IC-UNICAMP visa o desenvolvimento de um sistema de segurança imunológico que seja capaz de detectar e caracterizar um ataque, elaborar um plano de resposta especializado e efetuar o contra-ataque. E talvez o mais importante, um sistema que possua a mesma capacidade de aprendizado e adaptação do sistema imunológico humano, podendo reagir a ataques desconhecidos e melhorar sua reação a exposições subsequentes de um mesmo ataque.

Para viabilizar a capacidade de aprendizado e adaptação, proposta para o sistema de segurança imunológico, é necessário um mecanismo que permita analisar um ataque desconhecido e gerar uma caracterização específica que possa ser usada para a detecção eficiente no caso de futuras exposições ao mesmo ataque. Através da análise das evidências deixadas pelo invasor como, por exemplo, registros em arquivos de *log*, processos em execução e conexões de rede, utilizando-se técnicas de forense computacional, é possível reconstruir as principais características do ataque e gerar uma assinatura que o identifique. É justamente na análise forense para a caracterização de um ataque em que se limita o escopo desta dissertação, de modo que seu objetivo pode ser resumido como segue:

> **Fornecer subsídios para o desenvolvimento de um sistema de segurança imunológico, no sentido de se entender como utilizar a forense computacional, de maneira automatizada, na identificação e caracterização de um ataque.**

Nesse sentido, foi desenvolvido um modelo de IDS híbrido baseado no sistema imunológico humano (referenciado, ao longo desta dissertação, por modelo de IDS imunológico),

que agrega as características desejadas para o referido sistema de segurança imunológico, servindo de base para seu desenvolvimento. Além disso, foram investigados a fundo os diversos conceitos e técnicas empregados na área da forense computacional, com o objetivo de compreender como eles podem ser aplicados no modelo de IDS desenvolvido. Com base nesse estudo sobre forense computacional, foi desenvolvida uma arquitetura extensível para a automatização do processo de análise forense, com o intuito de permitir que a base de conhecimento de mau uso do modelo de IDS imunológico possa ser alterada autonomamente. Com o objetivo de testar a viabilidade dessa arquitetura de automatização, foi implementado um protótipo inicial, denominado AFA (*Automated Forensic Analyser*), que codifica parte das funcionalidades propostas para a arquitetura.

1.2 Nota sobre a utilização dos exemplos

Neste trabalho são usados exemplos de técnicas, ferramentas e descrições de ambientes computacionais derivados do sistema operacional Linux (um sistema da família UNIX). Isso é feito considerando-se que os princípios e técnicas de detecção de intrusão e análise forense aplicados ao UNIX (em especial ao sistema Linux) podem ser aplicáveis a outros sistemas operacionais (incluindo-se os sistemas Windows e outros ambientes da família UNIX), ainda que alguns detalhes e características particulares possam variar.

A escolha da plataforma Linux, para ilustrar como as técnicas de detecção de intrusão e análise forense podem ser utilizadas na identificação e investigação de um incidente de segurança, deve-se a uma série de fatores, incluindo-se, por exemplo:

- Familiaridade de uso;
- Os sistemas UNIX, em geral, têm suas vulnerabilidades discutidas de forma mais pública, abrangente e aberta;
- Grande parte das discussões na área da forense computacional iniciaram-se e tem-se concentrado em torno dos sistemas UNIX. Algumas dessas discussões são disponíveis em listas eletrônicas abertas como *tct-users*[1] e *forensics*[2];
- Disponibilidade de ferramentas gratuitas e de código aberto, permitindo investigar a fundo seu funcionamento e até mesmo promover modificações e reutilização de código;

[1]Maiores informações sobre a lista de discussões eletrônica *tct-users* podem ser encontradas na URL http://www.porcupine.org/forensics/tct.html (disponível em janeiro de 2003).

[2]A lista de discussões eletrônica *forensics* pode ser acessada através da URL http://www.securityfocus.com (disponível em janeiro de 2003).

- O sistema Linux possibilita total controle sobre seu funcionamento, em especial no acesso às informações de um sistema de arquivos. Esse controle é essencial durante a análise forense de um sistema computacional;
- O sistema Linux, em particular, possui a capacidade de acessar diversos tipos de sistemas de arquivos, incluindo-se os da família DOS/Windows e os dos ambientes UNIX. A possibilidade de se ter um único conjunto de ferramentas para a aplicação de técnicas de análise consistentes em uma grande variedade de sistemas alvo torna a escolha do ambiente Linux virtualmente irresistível [10];

Os exemplos de uso de ferramentas e técnicas aparecem ao longo do texto em fonte `verbatim`. Nesses exemplos, o caracter "\" contido no final de uma linha é utilizado para indicar a sua continuidade na linha seguinte. Além disso, as ferramentas são executadas pelo usuário *root*, a partir do interpretador de comandos `bash`.

1.3 Organização do trabalho

A organização desta dissertação traduz de maneira fiel as etapas de desenvolvimento da pesquisa aqui apresentada. Desse modo, no Capítulo 2, é apresentada uma revisão dos conceitos e técnicas de análise empregados na detecção de intrusão, com o objetivo de fornecer um embasamento teórico para a compreensão do restante do trabalho. Em seguida, no Capítulo 3, o sistema imunológico humano é introduzido como modelo de desenvolvimento de um sistema de segurança, onde são descritas suas estruturas e seu funcionamento, identificadas algumas analogias entre o sistema de defesa humano e a segurança de computadores e apresentados alguns dos principais trabalhos que exploram essa relação.

Com base nas técnicas de detecção de intrusão e no funcionamento do sistema imunológico humano, descritos nos capítulos anteriores, o Capítulo 4 detalha o modelo de IDS híbrido baseado no sistema de defesa do corpo humano, desenvolvido pelo grupo de imunologia computacional do LAS-IC-UNICAMP. Para a compreensão de como a forense computacional pode ser aplicada no referido modelo de IDS imunológico, o Capítulo 5 apresenta uma discussão sobre os conceitos e técnicas aplicados na investigação forense de sistemas computacionais invadidos. Em seguida, no Capítulo 6, é discutida a automatização do processo de análise forense, apresentando-se uma arquitetura extensível para viabilizar essa automatização e o protótipo AFA, que implementa parte dessa arquitetura. Por fim, no Capítulo 7, são feitas algumas considerações finais sobre o trabalho desenvolvido e indicadas possíveis extensões para a continuação desta pesquisa.

O Apêndice A apresenta uma série de detalhes práticos relacionados à extração e análise dos dados contidos nas diversas fontes de informação de um sistema computacio-

nal. O Apêndice B aborda a questão do conjunto de ferramentas para a análise forense em sistemas computacionais, detalhando sua composição e descrevendo uma série de aplicativos usados na área. O Apêndice C apresenta uma discussão sobre o desenvolvimento e padronização de procedimentos e protocolos de análise forense e, no Apêndice D, são detalhadas algumas questões sobre a utilização e configuração dos componentes do AFA. Ao final desta dissertação há um glossário, onde é definida a terminologia principal utilizada neste trabalho.

Capítulo 2

Detecção de intrusão

A detecção de intrusão é uma evolução da tradicional prática de auditoria de sistemas, onde os registros de auditoria, gerados pelo sistema operacional e outros mecanismos de *log*, eram revisados manualmente de tempos em tempos. À medida que os computadores ficaram mais rápidos, complexos e numerosos, os registros de auditoria também aumentaram seu tamanho e complexidade, exigindo um processo automatizado de revisão.

A detecção de intrusão é uma tecnologia relativamente recente, tendo iniciado seu desenvolvimento a partir de 1980. Entretanto, alguns conceitos e técnicas fundamentais surgiram ao longo dos últimos 20 anos de evolução na área. O objetivo deste capítulo é introduzir esses conceitos, fornecendo o embasamento teórico necessário tanto para a análise das técnicas atuais quanto para o desenvolvimento de novas abordagens. Nesse sentido, são discutidos, na Seção 2.1, alguns conceitos aplicados na detecção de intrusão e apresentadas, na Seção 2.2, algumas das principais técnicas de análise.

De fato, este capítulo é um resumo de pontos relevantes do livro *Intrusion Detection* [3] de Rebecca Bace, cujos conceitos são essenciais para a compreensão deste trabalho. É importante mencionar, ainda, que este capítulo não compreende um levantamento das diversas ferramentas de detecção de intrusão disponíveis atualmente, e sim uma introdução aos conceitos e técnicas de análise empregados na área.

2.1 Conceitos da detecção de intrusão

Em termos gerais, os sistemas de detecção de intrusão consistem de três componentes fundamentais:

- uma fonte de informações, que provê um fluxo de registros de auditoria, também chamados de registros de eventos, utilizados para determinar a ocorrência de uma intrusão;

- um mecanismo de análise, que busca por sinais de intrusões no fluxo de eventos derivado da fonte de informações;
- um componente de resposta, que gera reações baseadas na saída do mecanismo de análise;

Existem várias abordagens de projeto utilizadas na detecção de intrusão. Tais abordagens determinam as características providas por um IDS específico, incluindo suas capacidades de detecção. Ao longo desta seção, as principais abordagens de projeto de um IDS são apresentadas.

2.1.1 Arquitetura

A arquitetura refere-se à localização do IDS em relação ao sistema monitorado. Existem duas abordagens:

- **mesma localização**: em alguns casos não é possível separar o IDS do sistema monitorado, devido, por exemplo, à indisponibilidade de um sistema separado ou a restrições impostas pela estratégia de monitoramento (conceito apresentado na sequência). Essa abordagem apresenta um problema de segurança, de modo que um atacante pode simplesmente desabilitar o IDS como parte de seu ataque;
- **localização separada**: com o advento das estações de trabalho e dos computadores pessoais, a maioria dos IDSs passaram a utilizar uma arquitetura onde os registros de auditoria são armazenados e processados em um ambiente separado do sistema que se deseja proteger. Esse tipo de arquitetura busca evitar que um intruso desabilite o IDS ou modifique seus resultados, além de diminuir a carga adicional de processamento causada pelo IDS;

2.1.2 Estratégia de monitoramento

De acordo com a fonte de informações, a estratégia de monitoramento de um IDS pode ser dividida em quatro categorias, como segue:

- **baseada em uma máquina**: monitora fontes de informações internas a uma máquina, usualmente no nível do sistema operacional. Essas informações incluem os registros de auditoria gerados pelo sistema operacional e os arquivos de *log* do sistema;
- **baseada em rede**: monitora os pacotes que trafegam na rede, usualmente através de dispositivos de rede em modo promíscuo;

- **baseada em aplicação**: monitora os dados das aplicações em execução, incluindo *logs* de eventos e estruturas de dados internas à aplicação. É um sub-conjunto da estratégia de monitoramento baseada em uma máquina, entretanto o processo é especializado para uma determinada aplicação executada no sistema;

- **baseada em alvo**: essa abordagem tem um funcionamento diferente das demais, no sentido de que ela produz suas próprias informações. O monitoramento é feito através de *hashes* criptográficos[1], buscando detectar alterações em objetos do sistema (alvos). Também é um sub-conjunto da estratégia de monitoramento baseada em uma máquina;

2.1.3 Tipo de análise

A maioria das técnicas de análise empregadas na detecção de intrusão podem ser classificadas segundo uma abordagem de detecção por mau uso, detecção por anomalia ou alguma mistura das duas:

- **detecção por mau uso**: o mecanismo de análise monitora as atividades do sistema em busca de eventos ou conjuntos de eventos que representam padrões característicos de ataques conhecidos. Esses padrões característicos são comumente chamados de assinatura do ataque. Em geral, essa abordagem é bastante eficiente e gera um número menor de alarmes falsos. Além disso, por ser baseada em assinaturas, é possível diagnosticar rápida e confiavelmente o uso de ferramentas e técnicas específicas de ataque. Entretanto, esse tipo de detecção é capaz apenas de detectar ataques previamente conhecidos e codificados e, dependendo do nível de detalhes das assinaturas, pode também apresentar dificuldade na detecção de variantes de ataques conhecidos;

- **detecção por anomalia**: o mecanismo de análise busca por comportamentos anormais. Os IDSs por anomalia funcionam com a suposição de que os ataques causam diferenças perceptíveis no comportamento normal do sistema, permitindo a identificação do ataque. A detecção por anomalia utiliza perfis representando o comportamento normal dos usuários, do sistema e da rede. Então, uma variedade de medidas são aplicadas sobre o fluxo de eventos monitorados, buscando identificar desvios em relação ao perfil de comportamento normal. Essa abordagem, por ser baseada no comportamento normal do sistema e não em características específicas dos ataques, como na deteção por mau uso, é capaz de detectar ataques desconhecidos. Além disso, em alguns casos, as informações produzidas pela detecção de um ataque podem

[1]Maiores informações sobre *hashes* criptográficos e outras tecnologias de criptografia podem ser encontradas em [88].

ser utilizadas na definição de assinaturas para a detecção por mau uso. Entretanto, as maiores desvantagens dessa abordagem são a geração de um grande número de alarmes falsos, devido ao comportamento imprevisível de usuários e redes, e à necessidade de conjuntos extensivos de eventos de treinamento para caracterizar um perfil de comportamento normal;

- **detecção híbrida**: o mecanismo de análise combina as duas abordagens anteriores, buscando detectar ataques conhecidos e comportamentos anormais. A Figura 2.1 ilustra um IDS genérico que utiliza uma abordagem híbrida de detecção;

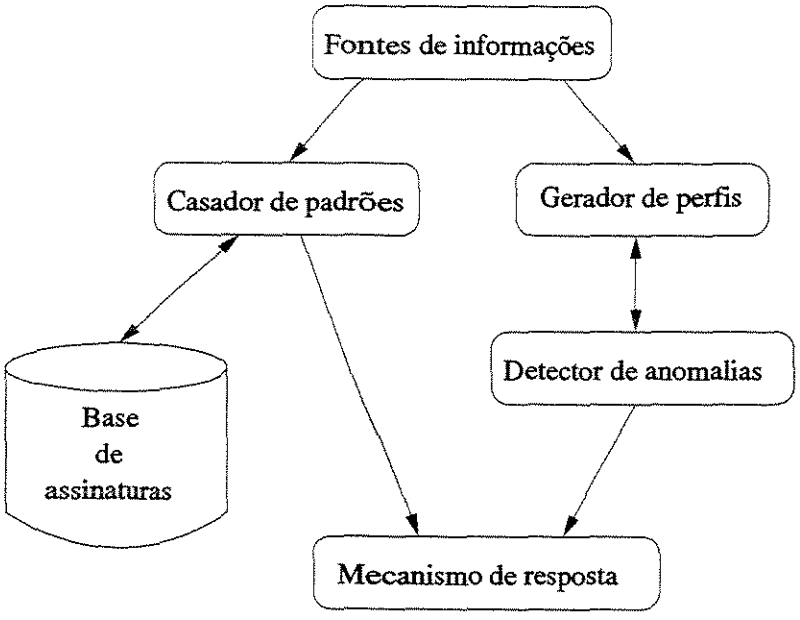


Figura 2.1: Diagrama de um IDS híbrido genérico.

2.1.4 Momento da análise

Outro modo de diferenciar os IDSs é quanto ao momento em que análise é feita. Esse momento refere-se ao tempo passado entre o acontecimento dos eventos monitorados e a sua análise. Basicamente, a detecção pode ser conduzida em tempo real ou em modo *batch* (também conhecida como detecção baseada em intervalo):

- **detecção em tempo real**: as informações são processadas imediatamente após sua geração. Permite detectar uma intrusão em andamento;

- **detecção em modo *batch***: os eventos gerados pelo sistema são inicialmente armazenados, sendo processados em um momento posterior;

2.1.5 Objetivo da detecção

Os dois objetivos tradicionais por trás da detecção de intrusão são a atribuição de responsabilidades e a geração de respostas:

- **atribuição de responsabilidades**: é a capacidade de mapear uma determinada atividade ou evento à fonte responsável;
- **geração de respostas**: o IDS produz algum tipo de ação, incluindo a geração de alarmes e relatórios, reconfiguração de roteadores e *firewalls*, finalização de conexões e geração de contra-ataques direcionados ao invasor;

2.1.6 Estratégia de controle

A estratégia de controle descreve a maneira como os elementos de um IDS são controlados. Existem duas principais abordagens quando o IDS monitora múltiplos sistemas:

- **centralizado**: um ponto central controla todos os elementos do IDS;
- **distribuído**: o monitoramento e detecção são conduzidos segundo uma abordagem baseada em agentes, onde as decisões de resposta são tomadas no ponto onde é feita a análise;

2.2 Técnicas de análise

Inúmeras técnicas podem ser utilizadas pelo mecanismo de análise para a detecção de uma intrusão. Em geral, essas técnicas podem ser classificadas em alguma das duas abordagens principais de análise: a detecção por mau uso e a detecção por anomalia. Esta seção descreve as principais técnicas de análise empregadas em cada abordagem.

2.2.1 Técnicas de detecção por mau uso

Como visto anteriormente, a detecção por mau uso envolve a codificação de informações acerca de atividades conhecidamente intrusivas e, posteriormente, a filtragem dos eventos monitorados em busca dessas atividades codificadas. Essa abordagem apresenta uma maior confiabilidade na detecção e é adotada pela maioria dos IDSs comerciais, no entanto, ela restringe-se somente a detecção de ataques conhecidos. A seguir são apresentadas as principais técnicas de análise utilizadas na detecção por mau uso.

2.2.1.1 Sistemas de produção/especialistas

Os sistemas de produção/especialistas foram um dos primeiros esquemas usados na detecção por mau uso. Entre os sistemas de produção empregados estão o P-BEST [89], desenvolvido por Alan Whitehurst, e o sistema CLIPS [36], um sistema de domínio público desenvolvido pela NASA (*National Aeronautics and Space Administration*).

A base de conhecimento de ataques é expressa na forma de regras do tipo "se-então" (*if-then*). As condições indicativas de uma intrusão são especificadas na parte "se" da regra. Quando essas condições são satisfeitas, as ações especificadas na parte "então" são executadas.

A principal vantagem associada ao uso de sistemas de produção/especialistas é a possibilidade do usuário especificar os conhecimentos sobre os ataques sem afetar, ou se quer entender, o funcionamento interno do sistema de detecção. No entanto, alguns problemas práticos são relacionados à utilização de sistemas de produção/especialistas na detecção de intrusão:

- eles são deficientes na manipulação de grandes volumes de dados, pois geralmente são implementados como sistemas interpretados;
- não provêem uma forma natural para a manipulação de dados sequencialmente ordenados;
- a habilidade refletida pelo sistema de produção/especialista é tão boa quanto a da pessoa que o modelou;
- eles não podem lidar com incertezas;

2.2.1.2 Abordagens baseadas em transição de estados

As abordagens baseadas em transição de estados estruturam o problema da detecção por mau uso, de modo a permitir a utilização de técnicas otimizadas de casamento de padrões (*pattern-matching*). Sua velocidade e flexibilidade as tornam uma das mais poderosas técnicas de detecção de intrusão atuais.

Essas abordagens utilizam expressões de estados e transições para descrever e detectar ataques conhecidos. Entre as principais abordagens estão a análise de transição de estados, as redes de Petri coloridas e as linguagens de caracterização de transição de estados.

Análise de transição de estados

A análise de transição de estados é uma abordagem de detecção por mau uso que utiliza diagramas de transição de estados de alto nível para representar e detectar cenários

conhecidos de intrusões. Essa abordagem foi inicialmente explorada nos sistemas STAT [76] e USTAT [48].

Um diagrama de transição de estados é uma representação gráfica de um cenário de intrusão, onde os nós representam os estados e os arcos representam as transições, como ilustrado na Figura 2.2.

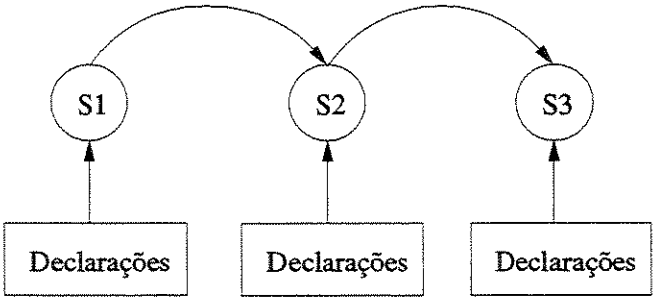


Figura 2.2: Diagrama de transição de estados.

Os sistemas de análise de transição de estados utilizam autômatos finitos para modelar as intrusões. Uma intrusão é composta de uma sequência de ações que levam de algum estado inicial até um estado intrusivo. O estado inicial representa o estado do sistema antes da execução da intrusão e o estado intrusivo representa o estado do sistema após a consumação da intrusão. O estado do sistema é descrito em termos de atributos do sistema e/ou privilégios de usuários. A transição de estados é direcionada pelas ações dos usuários.

O mecanismo de transição de estados mantém um conjunto de diagramas de transição de estados, onde cada um representa um cenário de intrusão. Em um determinado momento, cada diagrama encontra-se em um estado particular. Quando um novo evento ocorre, o mecanismo de transição checa esse evento em cada um dos diagramas de transição para determinar se ele causa uma mudança de estado. Se o evento anula as declarações do estado corrente, o mecanismo de inferência move a transição de estados para o estado mais próximo onde as declarações ainda são válidas. Caso o evento leve o cenário para o estado final, indicando uma intrusão, a informação sobre a transição anterior é enviada para o mecanismo de decisão, que gera um alerta sobre a ocorrência da intrusão.

Entre as vantagens apresentadas pelo sistema STAT estão:

- os diagramas de transição de estados provêem uma representação intuitiva, de alto nível e independente do registro de auditoria para os cenários de intrusão;

- as transições permitem representar uma ordem parcial nas ações que constituem os cenários de ataque;
- um diagrama de transição de estados usa o menor conjunto possível de ações que devem ocorrer para a intrusão ser caracterizada. Assim, o detector pode generalizar sobre variantes de um mesmo cenário de intrusão;
- é possível detectar ataques coordenados e lentos;

No entanto, o sistema STAT apresenta algumas deficiências, incluindo:

- a lista de declarações dos estados e ações de uma assinatura são codificadas manualmente;
- as declarações dos estados e assinaturas podem não ser poderosas o suficiente para expressar cenários intrusivos mais elaborados;
- a avaliação de algumas declarações dos estados pode causar uma degradação de performance;

Redes de Petri coloridas

Outra abordagem baseada em transição de estados é o uso de redes de Petri coloridas (CP-*Nets* — *Colored Petri Nets*) [51], implementada no sistema IDIOT [55].

O sistema IDIOT utiliza uma variação de CP-*Nets* para representar e detectar padrões de intrusões. Segundo esse modelo, uma intrusão é representada por uma CP-*Net* onde a cor de *tokens* em cada estado serve para modelar o contexto de um evento. O casamento da assinatura é dirigido pelo fluxo de registros de auditoria e é conduzido através da movimentação progressiva dos *tokens*, partindo de um estado inicial até um estado final (representando a intrusão). Expressões são posicionadas nas transições, permitindo a definição dos contextos em que os padrões são considerados detectados, e ações podem ser definidas para serem executadas ao final de uma detecção.

A princípio, essa abordagem pode parecer idêntica à abordagem de análise de transição de estados implementada no sistema STAT. No entanto, existe uma diferença significativa entre elas: no sistema STAT, uma intrusão é detectada pelos efeitos que ela causa no estado do sistema monitorado, enquanto que no sistema IDIOT, uma intrusão é detectada pelo casamento de padrões que constituem a invasão em si.

No sistema IDIOT, cada assinatura de intrusão é expressa como um conjunto de padrões que representa o relacionamento entre os eventos e seus contextos. Esse conjunto de padrões precisamente representa uma intrusão bem sucedida ou sua tentativa. O modelo de casamento de padrões consiste do seguinte:

- uma representação de contexto que permite o correlacionamento de vários eventos que constituem a assinatura da intrusão;
- uma semântica que acomoda a possibilidade de vários padrões (possivelmente pertencentes a múltiplas fontes de informações) serem misturados no mesmo fluxo de eventos;
- uma especificação de ações que provê a execução de certas atividades à medida que os padrões são verificados;

Entre as principais vantagens associadas a essa abordagem podem ser citadas as seguintes:

- é extremamente rápida;
- o mecanismo de casamento de padrões é independente do formato dos registros de auditoria;
- os padrões podem ser especificados de acordo com o que precisa ser detectado, e não como eles devem ser analisados;
- o sequenciamento e outras restrições de ordenação dos eventos podem ser representados diretamente;
- o sistema permite a especificação de ações executadas após o casamento de uma assinatura, permitindo a implementação de respostas automatizadas;

Abordagens baseadas em linguagens ou interfaces de programação de aplicações (API — *Application Programming Interface*)

Uma estratégia comum para a otimização de ferramentas comerciais de detecção por mau uso é o desenvolvimento de meios para a descrição de intrusões, de forma que um mecanismo de detecção possa utilizar. Embora algumas linguagens usadas em sistemas de produção/especialistas estejam disponíveis (como as anteriormente mencionadas P-BEST e CLIPS), elas não foram desenvolvidas para os propósitos da detecção de intrusão. Existem algumas abordagens para a expressão de intrusões, objetivando os propósitos da detecção por mau uso, podendo ser citadas as seguintes:

- **RUSSEL** [69]: uma linguagem baseada em regras, utilizada no sistema ASAX [38], desenvolvida para otimizar o processamento de fluxos de dados não-estruturados (especialmente fluxo de auditoria do sistema operacional). Seu objetivo é possibilitar

o correlacionamento de eventos entre múltiplas máquinas e suportar múltiplos níveis de abstração para os eventos. Os padrões de eventos são expressos na forma de ações condicionadas (condição → ação). Essa linguagem é estruturada no sentido *bottom-up*: ela inicia com a declaração de registros de auditoria como fatos básicos e então, baseada nesses fatos, ela tenta encontrar padrões no registro de auditoria que podem ser vistos como fatos derivados;

- **STALKER** [95]: essa abordagem patenteada utiliza um formato comum de dados de auditoria (o padrão SVR 4++ [94]) e uma estrutura de dados baseada em estados para as assinaturas de ataques. O detector é implementado como uma máquina de estados finita, otimizada para o casamento de padrões. A expressão de uma assinatura consiste de uma estrutura de dados que contém um estado inicial, um estado final e um ou mais conjuntos de funções de transição para cada mau uso;

- **N-*Code***: essa linguagem é utilizada no *Network Flight Recorder*[2] (um sistema de monitoramento de rede) para a definição de filtros de pacotes de rede. Esses filtros de pacotes podem ser programados para reconhecer ataques de rede, além de outras atividades quaisquer. A linguagem N-*Code* é uma linguagem interpretada que opera em instruções *byte-code* e implementa uma máquina de pilha simples;

2.2.1.3 Recuperação de informações para análise em modo *batch*

Embora a maioria dos IDSs seja baseada na coleta e análise das eventos em tempo real, algumas abordagens envolvem a análise de dados de auditoria arquivados, buscando por padrões de atividades específicas ou provendo a habilidade de isolar atividades relacionadas a um determinado objeto (um usuário particular, por exemplo). Essas abordagens são especialmente interessantes para investigadores e equipes de resposta a incidentes.

Uma implementação nesse sentido envolve a utilização de técnicas de recuperação de informações (*information retrieval*)[2], amplamente aplicadas nos mecanismos de busca da Internet (como o AltaVista[3], por exemplo). Os sistemas de recuperação de informações utilizam arquivos invertidos como índices, permitindo a busca eficiente de palavras-chave e combinações delas. Esses sistemas também utilizam algoritmos que aprendem as preferências de um usuário e usam inferência Bayesiana para ajudar no refinamento das buscas.

[2]Maiores detalhes sobre o *Network Flight Recorder* podem ser encontrados na URL http://www.nfr.net (disponível em janeiro de 2003).

[3]O mecanismo de busca AltaVista pode ser acessado através da URL http://www.altavista.com (disponível em janeiro de 2003).

2.2.1.4 Detecção por modelagem (*model-based*)

Esta abordagem tenta modelar as intrusões em um nível maior de abstração que o nível dos registros de auditoria. O objetivo é a construção de cenários modelo que representam o comportamento característico das intrusões. Isso permite que sejam geradas representações abstratas dos ataques, passando para o sistema de detecção a responsabilidade de determinar quais registros de auditoria são parte de uma sequência suspeita.

A técnica de detecção por modelagem proposta por Garvey e Lunt [35] suporta a abstração de cenários intrusivos através de uma ferramenta de raciocínio por evidência (*evidential reasoning*), chamada Gister[4][63, 61]. Essa ferramenta avalia pedaços de evidência contra uma hipótese, no esforço de construir uma medida de confiança quanto à veracidade da hipótese testada. Conforme são identificadas evidências nos registros de auditoria que indicam que a atividade representada em um cenário modelo está sendo executada, o cenário é adicionado a um conjunto de cenários ativos. Esses cenários ativos são utilizados pela ferramenta Gister para aumentar ou diminuir a medida de confiança acerca da ocorrência de uma intrusão.

2.2.2 Técnicas de detecção por anomalia

Como visto na seção 2.1.3, a detecção por anomalia envolve a construção de perfis representando o comportamento normal de usuários, sistemas, processos e conexões de rede. Tais perfis são construídos a partir de dados históricos, coletados em um período de operação normal, e são utilizados para definir um conjunto de métricas. Essas métricas são medidas de aspectos particulares do comportamento monitorado. Após a construção desses perfis, as atividades monitoradas são comparadas a eles e uma variedade de medidas são aplicadas para determinar se as atividades desviam do comportamento normal. A detecção por anomalia funciona na suposição de que os ataques são diferentes da atividade normal e podem, portanto, ser detectados por sistemas que identificam essas diferenças. Algumas das principais técnicas de análise empregadas nessa abordagem são apresentadas na sequência.

2.2.2.1 Modelo original de Dorothy Denning

Dorothy Denning define em [21] quatro modelos estatísticos que podem ser utilizados em um detector por anomalia. Cada modelo, descrito na sequência, é considerado apropriado para um tipo particular de métrica.

[4]A ferramenta Gister é uma marca registrada de SRI International. Maiores informações podem ser encontradas na URL http://www.aic.sri.com (disponível em janeiro de 2003).

Modelo operacional

O modelo operacional aplica-se a métricas como, por exemplo, contadores de eventos para o número de falhas de *login* em um determinado intervalo de tempo. O modelo compara a métrica a um limiar definido, identificando uma anomalia quando a métrica excede o valor limite. Esse modelo, aplicado tanto na detecção por anomalia quanto na detecção por mau uso, corresponde à detecção por limiar, apresentada adiante.

Modelo de média e desvio padrão

O segundo modelo de Denning propõe uma caracterização clássica de média e desvio padrão para os dados. Uma nova observação de comportamento é identificada como anormal se ela encontra-se fora de um intervalo de confiança. Esse intervalo de confiança é definido como sendo d desvios padrões da média, para algum parâmetro d. Denning levanta a hipótese de que essa caracterização é aplicável a métricas do tipo contadores de eventos, intervalos de tempo e medidas de recursos.

Modelo multivalorado

O modelo multivalorado é uma extensão ao modelo de média e desvio padrão. Ele é baseado na correlação entre duas ou mais métricas. Desse modo, ao invés de basear a detecção de uma anomalia estritamente em uma métrica, essa detecção é baseada na correlação dessa métrica com alguma outra medida.

Modelo de processo de Markov

Esta parte do modelo de Dorothy Denning é mais complexa e limitada a contadores de eventos. Segundo o modelo de processo de Markov, o detector considera cada tipo diferente de evento de auditoria como uma variável de estado e usa uma matriz de transição de estados para caracterizar as frequências com que ocorrem as transições entre os estados. Uma nova observação de comportamento é definida como anormal se sua probabilidade, determinada pelo estado anterior e pelo valor na matriz de transição de estados, for muito baixa. Esse modelo permite que o detector identifique sequências não usuais de comandos e eventos, introduzindo a noção de análise de fluxos de eventos com memória de estado (*stateful analysis*).

2.2.2.2 Análise quantitativa

A abordagem de detecção por anomalia mais comumente utilizada é a análise quantitativa, onde as regras e atributos de detecção são expressos no formato numérico. Essa abordagem envolve algum tipo de cálculo, podendo variar de uma simples adição a cálculos criptográficos mais complexos. Os resultados das técnicas de análise quantitativa podem

servir de base para as assinaturas de detecção por mau uso e também para os modelos estatísticos de detecção por anomalia. Algumas dessas técnicas são apresentadas na sequência.

Detecção por limiar (*threshold detection*)

A detecção por limiar é provavelmente a forma mais comum de análise quantitativa. Nesse tipo de detecção, certos atributos do comportamento monitorado são caracterizados em termos de contadores, com algum limiar estabelecido como permitido. Se um contador excede o limiar permitido, o comportamento relacionado é identificado como anômalo. Um exemplo clássico de limiar é o número máximo permitido de falhas de *login* em um sistema. Uma suposição inerente à detecção por limiar é que a medida é feita ao longo de um determinado intervalo de tempo. Esse intervalo pode ser fixado no tempo (por exemplo, a medida é zerada em uma determinada hora do dia) ou pode funcionar sobre um janela deslizante (por exemplo, a medida é feita ao longo das últimas oito horas).

Detecção por limiar heurístico

A detecção por limiar heurístico leva a um passo adiante a detecção por limiar simples, através da adaptação do limiar aos níveis observados. Esse processo aumenta a precisão da detecção, especialmente quando ela é dirigida a uma grande variedade de usuários e sistemas alvo. Desse modo, por exemplo, ao invés de ter uma regra de detecção por limiar que dispara um alerta quando o número de falhas de *login* excede três em um período de oito horas, pode-se ter uma regra que alerta quando um número anormal de falhas de *login* ocorre. Esse número anormal pode ser definido através de várias fórmulas como, por exemplo, a média do número de falhas de *login* (essa média é calculada com base nos níveis observados e o número de falhas subsequentes é então comparado com a média calculada, adicionada a algum desvio padrão).

Checagem de integridade baseada no alvo

A checagem de integridade baseada no alvo é uma checagem de mudanças em um determinado objeto do sistema que não deveria estar sujeito a mudanças imprevistas. O exemplo mais comum é a utilização de *hashes* criptográficos. Depois que o *hash* é calculado e armazenado em um local seguro, o sistema periodicamente recalcula o *hash* do objeto, comparando-o com o valor de referência armazenado. Se uma diferença é encontrada, um alarme é disparado. O programa Tripwire[5] implementa essa técnica.

[5]Informações sobre o programa Tripwire podem ser encontradas na URL http://www.tripwire.com (disponível em janeiro de 2003).

Análise quantitativa e redução de dados

Um dos usos mais interessantes da análise quantitativa é a redução de dados. A redução de dados é o processo de eliminação de informações supérfluas ou redundantes contidas nos volumosos registros de eventos, reduzindo a carga de armazenamento e otimizando o processo de detecção. Um exemplo da utilização de análise quantitativa para a redução de dados é o sistema NADIR [41], que transforma as atividades dos usuários, contidas nos registros de auditoria, em vetores de medidas quantitativas. Os perfis gerados são agregados por tempo (com resumos semanais) ou por sistema (com visões das atividades dos usuários em cada sistema), sendo submetidos a análises estatísticas e a sistemas especialistas.

2.2.2.3 Medidas estatísticas

Outra abordagem para detecção por anomalia é a utilização de medidas estatísticas. Entre os sistemas que utilizam essa técnica podem ser citados o *Intrusion Detection Expert System* (IDES) e o *Haystack*, discutidos na sequência.

IDES

As técnicas de análise estatística aplicadas no sistema IDES [50] suportam históricos de perfis estatísticos, estabelecidos e mantidos para cada usuário e sistema monitorados. Esses perfis são atualizados periodicamente, com os dados mais velhos evoluídos para que o perfil se adapte a mudanças no comportamento dos usuários ao longo do tempo.

O sistema IDES mantém uma base de conhecimento estatístico contendo os perfis. Cada perfil expressa os comportamentos normais de um usuário particular, em termos de um conjunto de métricas. Uma vez ao dia, novos dados de auditoria são incorporados à base de conhecimento (depois que os perfis antigos são evoluídos através de um fator de declínio exponencial), segundo a atividade do usuário durante o dia.

Cada vez que um registro de auditoria é gerado, uma estatística, chamada de IDES *score* (IS), é gerada pela seguinte fórmula:

$$IS = (S_1, S_2, S_3 \ldots S_n)C^{-1}(S_1, S_2, S_3 \ldots S_n)t,$$

onde $(S \ldots)C^{-1}$ é o inverso de um matriz ou vetor de correlação e $(S \ldots)t$ é a transposta do vetor. Cada S_i mede algum aspecto do comportamento como, por exemplo, número de acessos a arquivos, número de terminais usados e tempo de CPU. Valores diferentes de S_i podem também representar visões diferentes para o mesmo aspecto do comportamento.

Haystack

O sistema *Haystack* [93] emprega uma abordagem estatística de duas partes para a

detecção por anomalia. A primeira medida determina em que grau uma sessão de usuário assemelha-se a um tipo de intrusão estabelecido. Essa medida é calculada da seguinte forma:

1. o sistema mantém um vetor de medidas de comportamento do usuário;
2. para cada tipo de intrusão, o sistema associa um peso a cada medida de comportamento, refletindo a relevância da medida para um determinado tipo de intrusão;
3. para cada sessão, o vetor de medidas de comportamento do usuário é calculado e comparado com o vetor limiar mantido;
4. aquelas medidas de comportamento, cujo limiar definido é excedido, são anotadas;
5. os pesos associados às medidas que excederam os limiares são somados;
6. a soma é usada para atribuir um quociente de suspeita à sessão, baseado na distribuição dos valores ponderados das intrusões para todas as sessões anteriores;

O segundo método estatístico complementar detecta desvios, do perfil de uma sessão normal, nas atividades de uma sessão de usuário. Esse método busca por estatísticas da sessão que significantemente desviam do perfil estatístico histórico normal para o usuário.

2.2.2.4 Medidas estatísticas não-paramétricas

As primeiras técnicas estatísticas eram similares na utilização de abordagens paramétricas para caracterizar os padrões de comportamento dos usuários e outras entidades do sistema monitorado. As abordagens paramétricas referem-se a abordagens analíticas onde suposições são feitas acerca da distribuição dos dados analisados. Por exemplo, nas primeiras versões do sistema IDES as distribuições dos padrões de uso dos usuários eram assumidas como sendo Gaussianas ou normais.

O problema com essas suposições é que as taxas de erros são altas quando as suposições são incorretas. Linda Lankewicz e Mark Benard [56] propuseram que uma forma de resolver esses problemas seria a utilização de técnicas não-paramétricas para a detecção por anomalia. Essa abordagem permite acomodar usuários com padrões de uso menos previsíveis, além de possibilitar a utilização de medidas que não são facilmente aplicáveis em esquemas paramétricos.

A abordagem de Lankewicz e Benard envolve técnicas não-paramétricas de classificação de dados, especialmente a análise por agrupamento (*clustering analysis*). Na análise por agrupamento, grandes quantidades de dados históricos são coletados e organizados em grupos (*clusters*), de acordo com algum critério de avaliação. É executado um

pré-processamento onde as características associadas a um determinado fluxo de eventos (geralmente mapeado para um usuário específico) são convertidas em uma representação de vetor. Um algoritmo de agrupamento é usado para organizar os vetores em classes de comportamentos, de modo que os membros de cada classe sejam o mais semelhante possível, enquanto que as diferentes classes sejam o mais distante possível uma da outra.

Na detecção por anomalia, usando medidas estatísticas não-paramétricas, a premissa é que os dados de atividades de um usuário recaem em dois grupos distintos: um indicando atividades anômalas e outro indicando atividades normais.

2.2.2.5 Abordagens baseadas em regras

Outra variação para a detecção por anomalia é representada pelas abordagens baseadas em regras. As suposições que embasam as abordagens baseadas em regras são as mesmas daquelas associadas aos métodos estatísticos de detecção por anomalia. A diferença principal é que os IDSs baseados em regras utilizam conjuntos de regras para representar e armazenar os padrões de comportamento. Como exemplos desses sistemas podem ser citados o *Wisdom and Sense* e o *Time-Based Inductive Machine* (TIM), descritos como segue.

Wisdom and Sense

O primeiro sistema de detecção por anomalia baseado em regras foi o *Wisdom and Sense* (W&S) [58]. O sistema W&S provê dois esquemas para o povoamento das bases de regras: adição manual de regras (para refletir uma restrição da política de segurança, por exemplo) e geração de regras a partir de dados históricos de auditoria. As regras são derivadas dos dados históricos de auditoria através de um exame categórico, expressando os padrões encontrados na forma de regras. As regras refletem o comportamento passado das entidades do sistema monitorado e são armazenadas em uma estrutura de árvore. Valores de dados específicos contidos nos registros de auditoria são agrupados em classes de execução, com as quais são associadas conjuntos de operações ou regras.

Como exemplo de uma classe de execução pode ser citado: "todos os registros contendo os mesmos valores para o campo usuário". As regras são aplicadas aos dados de uma classe de execução toda vez que uma atividade associada a essa classe ocorrer. As anomalias são detectadas da seguinte forma: quando as transações são processadas, elas são comparadas aos eventos da classe de execução correspondente, para determinar se os eventos processados casam com os padrões históricos de atividade ou representam uma anomalia.

TIM

O sistema TIM [104] utiliza uma abordagem indutiva para gerar dinamicamente regras definindo intrusões. A diferença entre o sistema TIM e outros IDSs por anomalia é que ele busca por padrões em sequências de eventos, não em eventos individuais. Ele efetivamente implementa um modelo probabilístico de transição de Markov, como proposto por Denning em seu modelo original (descrito anteriormente).

O sistema TIM observa sequências de registros de eventos históricos, caracterizando a probabilidade de determinadas sequências de eventos ocorrerem. Outros IDSs por anomalia medem se a ocorrência de um único evento representa um desvio do padrão normal de atividade. Já o sistema TIM focaliza em sequências de eventos, checando se uma cadeia de eventos corresponde ao que seria esperado, com base em sua observação das sequências de eventos históricos.

O sistema TIM automaticamente gera regras acerca das sequências de eventos conforme ele analisa os dados históricos, armazenando-as em uma base de regras. Se uma sequência de eventos casa com o início de uma regra, então o próximo evento é considerado anômalo se ele não está no conjunto de eventos previstos no corpo da regra.

2.2.2.6 Redes neurais

Abordagens baseadas em redes neurais [6, 19] usam técnicas de aprendizado adaptativas para caracterizar comportamentos anômalos. Essa técnica de análise não-paramétrica opera em conjuntos de dados históricos de treinamento, que presumidamente são livres de qualquer dado indicativo de intrusão ou outros comportamentos indesejáveis.

As redes neurais consistem de inúmeros elementos de processamento, chamados de unidades, que se interagem através de conexões ponderadas. O conhecimento de uma rede neural é codificado na estrutura da rede em termos das conexões entre as unidades e dos pesos dessas conexões. O processo de aprendizado ocorre através da alteração dos pesos e adição ou remoção de conexões.

O processamento de uma rede neural envolve dois estágios. No primeiro, a rede neural é povoada através do conjunto histórico de dados de treinamento, que representa o comportamento normal dos usuários. No segundo estágio, a rede recebe os dados dos eventos e compara-os com as referências históricas de comportamento, determinando as semelhanças e diferenças.

2.2.3 Esquemas alternativos de detecção

Alguns esquemas recentes de detecção de intrusão representam atividades precursoras que podem direcionar ou refinar os métodos de detecção conhecidos. Algumas dessas

abordagens alternativas são descritas na sequência.

2.2.3.1 Abordagens baseadas em sistemas imunológicos

Em um projeto de pesquisa desenvolvido na Universidade do Novo México, Forrest, Hofmeyr e Somayaji, entre outros, identificaram inúmeras similaridades entre sistemas imunológicos e mecanismos de proteção de sistemas computacionais.

O ponto chave para o funcionamento de ambos os sistemas de defesa é a capacidade de diferenciar o que é próprio do sistema (*self*) e o que é estranho ou indesejado (*nonself*) [29]. Como o sistema imunológico humano faz essa diferenciação com base em pequenas cadeias de proteínas (peptídeos), os pesquisadores decidiram focar em um atributo análogo — pequenas cadeias de chamadas de sistema [31, 45].

O sistema inicialmente proposto realiza detecção por anomalia, composta de duas fases. Na primeira, o base de conhecimento é povoada com perfis de comportamento normal. Nesse caso, os perfis caracterizam o comportamento normal de processos do sistema com base nas chamadas de sistema que esses processos executam. Na segunda fase, os perfis são usados para monitorar o comportamento subsequente dos processos do sistema.

As abordagens de detecção de intrusão baseadas em sistemas imunológicos são detalhadas no Capítulo 3.

2.2.3.2 Algoritmos genéticos

Outra abordagem mais sofisticada para a detecção por anomalia envolve a utilização de algoritmos genéticos para a análise dos dados de eventos.

Um algoritmo genético é uma instância de uma classe de algoritmos chamada de algoritmos evolucionários. Os algoritmos evolucionários incorporam conceitos de seleção natural de Darwin para otimizar soluções de problemas. Os algoritmos genéticos utilizam formas codificadas, chamadas de cromossomos, com métodos que permitem a combinação ou mutação desses cromossomos para gerar novas formas.

O processo de detecção de intrusão, através de algoritmos genéticos [66], envolve a definição de vetores de hipótese para os dados de eventos, onde um vetor indica ou não uma intrusão. A hipótese é então testada para determinar sua validade. Uma hipótese melhorada é derivada da anterior e testada. Esse processo repete-se até que uma solução seja encontrada. O papel do algoritmo genético nesse processo é a derivação da hipótese melhorada.

2.2.3.3 Detecção baseada em agentes

Abordagens de detecção de intrusão baseadas em agentes são fundamentadas em entidades de *software* que realizam certas funções de monitoramento de segurança em uma máquina. Os agentes funcionam autonomamente, sendo controlados apenas pelo sistema operacional, podendo se comunicar e cooperar com outros agentes de construção similar.

Um agente pode oferecer diversas capacidades, incluindo a alternação entre métodos de detecção por mau uso e anomalia, e a implementação de mecanismos de resposta (por exemplo, a alteração do nível de prioridade de um processo qualquer).

Duas arquiteturas que utilizam agentes para a detecção de intrusão são descritas na sequência.

Agentes autônomos para detecção de intrusão (AAFID — *Autonomous Agents for Intrusion Detection*)

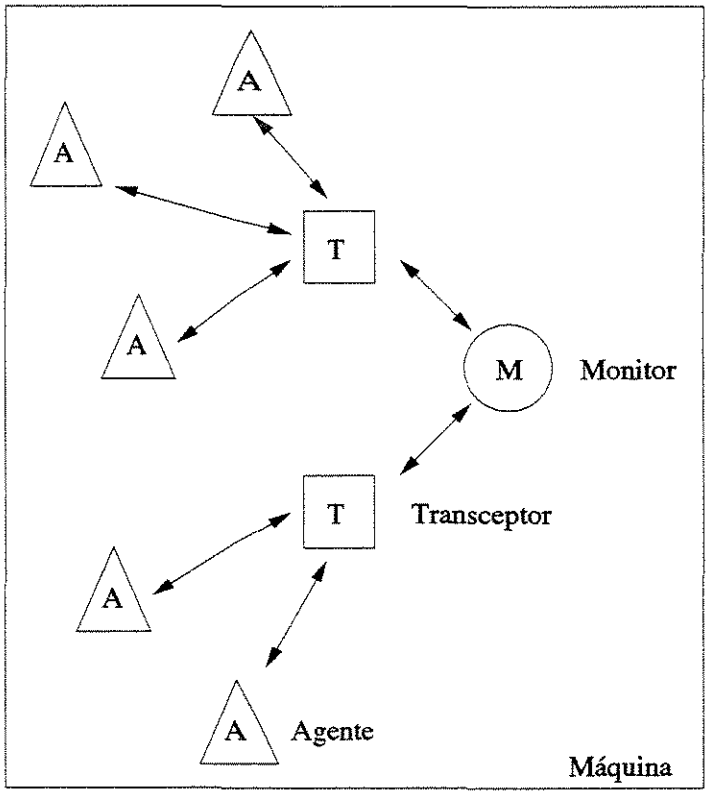


Figura 2.3: Arquitetura do AAFID.

A arquitetura provida pelo AAFID [4] requer uma estrutura de controle e comunicação

hierarquicamente ordenada para os agentes, como ilustrada na Figura 2.3. Qualquer número de agentes pode residir em uma máquina. Todos os agentes em uma determinada máquina reportam seus achados a um único transceptor. Os transceptores monitoram a operação de todos os agentes executando em uma máquina, com a capacidade de enviar comandos de inicialização, término e reconfiguração aos agentes. Os transceptores também realizam redução de dados sobre as informações reportadas pelos agentes, enviando os resultados a um ou mais monitores (no próximo nível da hierarquia).

Os monitores controlam e consolidam as informações provenientes de diversos transceptores. A arquitetura do AAFID permite a redundância nos relatos dos transceptores aos monitores, de modo que a falha em um monitor não arrisca o funcionamento do sistema de detecção. Os monitores têm a capacidade de acessar dados de toda a rede e, desse modo, realizar agregação em alto nível dos resultados provenientes dos transceptores, permitindo a detecção de ataques envolvendo múltiplas máquinas.

EMERALD

Uma segunda arquitetura que utiliza uma abordagem baseada em agentes distribuídos para a detecção de intrusões é o sistema EMERALD [77, 70]. Esse sistema inclui numerosos monitores locais em uma arquitetura que suporta a distribuição de resultados locais a um conjunto global de detectores que, por sua vez, consolidam alarmes e alertas.

O sistema EMERALD é um IDS híbrido, utilizando análise de assinaturas e perfis estatísticos para detectar problemas de segurança. Seu componente central é o monitor de serviço, semelhante no formato e função ao agente autônomo do AAFID. Os monitores de serviço podem ser dispostos em camadas para suportar a redução hierárquica dos dados, de modo que os monitores de um determinado nível realizam análises locais e reportam os resultados e informações adicionais aos monitores do próximo nível. O sistema também suporta uma grande variedade de respostas automatizadas.

2.2.3.4 Mineração de dados (*data mining*)

Uma abordagem similar a alguns dos esforços em detecção por anomalia baseada em regras envolve a utilização de técnicas de mineração de dados para a construção de modelos de detecção de intrusão [57]. O objetivo dessa abordagem é a descoberta de padrões consistentes de características do sistema que possam ser usados para descrever comportamentos de programas e usuários. Esses conjuntos de características do sistema, por sua vez, podem ser processados por métodos indutivos para formar classificadores (mecanismos de detecção) que podem reconhecer cenários de mau uso e anomalias.

Mineração de dados refere-se ao processo de extração de modelos a partir de grandes quantidades de dados. Esses modelos geralmente descobrem fatos contidos nos dados que

não são aparentes através de outros métodos de inspeção. Alguns algoritmos de mineração de dados úteis para o processamento de dados de auditoria são descritos como segue:

- **Classificação**: atribui um item de dado a uma das várias categorias predefinidas. Algoritmos de classificação produzem classificadores como, por exemplo, árvores de decisão e regras. Na detecção de intrusão, um classificador ótimo pode confiavelmente identificar dados de auditoria como pertencentes a uma categoria normal ou anormal;
- **Análise relacional (*link analysis*)**: identifica relacionamentos e correlações entre os campos dos registros de auditoria. Um algoritmo relacional ótimo, na detecção de intrusão, identifica o conjunto de características do sistema mais capaz de revelar confiavelmente as intrusões;
- **Análise de sequência**: modela padrões sequenciais. Esses algoritmos podem revelar quais eventos de auditoria tipicamente ocorrem juntos;

2.3 Conclusão

Este capítulo apresentou os principais conceitos e técnicas aplicados na área da detecção de intrusão, buscando fornecer o embasamento teórico necessário tanto para a análise das técnicas atuais quanto para o desenvolvimento de novas abordagens. Através dessa revisão, notou-se a busca constante por soluções eficazes de detecção de intrusão, devido à inexistência de um método único que possa ser aplicado com total eficácia em todos os casos e à contínua adaptação dos atacantes aos avanços em segurança de computadores. Nesse sentido, a adoção de melhores modelos de segurança pode representar um passo na direção de soluções mais eficazes de detecção de intrusão.

Capítulo 3

Imunologia computacional

Uma vez compreendidos os conceitos e técnicas de detecção de intrusão, é possível investigar e avaliar a aplicação de outras áreas de pesquisa no desenvolvimento de novas abordagens de análise e paradigmas de detecção.

Nesse sentido, este capítulo aborda a relação existente entre o sistema imunológico humano e a segurança de sistemas computacionais, com o objetivo de introduzir o sistema de defesa humano como modelo para o desenvolvimento de um sistema de segurança de computadores.

Inicialmente, na Seção 3.1, é apresentada uma introdução às estruturas e funcionamento do sistema imunológico humano e, na Seção 3.2, alguns pontos relevantes da analogia entre o sistema de defesa humano e a segurança de computadores são considerados. Por fim, alguns dos principais trabalhos em torno dessa analogia são apresentados na Seção 3.3.

3.1 O sistema imunológico humano

É impossível entender como o sistema imunológico pode ser usado como um modelo para o desenvolvimento de um sistema de defesa de computadores sem antes compreender seu funcionamento. Esta seção apresenta as estruturas básicas do sistema imunológico e detalha seu funcionamento. O material para esta revisão é amplamente baseado em [85, 72, 71, 25, 98].

3.1.1 Organização estrutural

O sistema imunológico defende o corpo humano contra ataques de invasores reconhecidos como substâncias estranhas. Ele constitui um sistema extraordinariamente complexo que

depende de uma rede de comunicação elaborada e dinâmica existente entre os diferentes tipos de células que patrulham o corpo humano.

O sistema imunológico é composto pelos sistemas inato e adaptativo. O sistema inato distingue-se por sua natureza congênita e por sua capacidade limitada em diferenciar um agente infeccioso de outro, reagindo de maneira semelhante a todas as substâncias estranhas. O sistema inato representa a primeira linha de defesa contra a ação de micróbios, e sua resposta, por não ser específica para um determinado micróbio, é, na maioria das vezes, insuficiente. Entre seus principais componentes estão as barreiras físicas e químicas, como a pele e os ácidos gástricos, e as células conhecidas como fagócitos, que vasculham o corpo humano em busca de substâncias estranhas.

Em contraste com o sistema inato, o sistema adaptativo é capaz de identificar especializadamente um determinado agente patogênico, permitindo uma resposta mais eficiente. Além disso, ele é capaz de "memorizar" um agente infeccioso e responder mais vigorosamente a novas exposições a esse micróbio. Esse sistema é chamado de adaptativo por ser responsável pela imunidade adquirida ao longo da vida do indivíduo. O sistema adaptativo é composto pelas células conhecidas como linfócitos (células T e células B) e seus produtos, como os anticorpos.

Os mecanismos de ambos os sistemas, inato e adaptativo, constituem um sistema integrado de defesa em que um grande número de células e moléculas agem cooperativamente. O sistema inato não só provê a primeira linha de defesa, mas também atua em diversas etapas da resposta do sistema adaptativo.

3.1.2 Resposta imunológica

No coração do sistema imunológico está a habilidade de reconhecer e responder a substâncias chamadas de antígenos (formadas por proteínas), presentes na superfície dos agentes infecciosos e também nas substâncias do corpo humano (antígenos próprios). Para fazer isso, o sistema imunológico executa tarefas de reconhecimento de padrões para diferenciar as células e moléculas do corpo humano (chamadas de *self*) das substâncias estranhas (chamadas *nonself*). Esse reconhecimento de padrões é realizado através da reação entre os antígenos e as cadeias de proteínas contidas na superfície das células do sistema imunológico, chamadas de receptores. Desse modo, os antígenos são os padrões a serem reconhecidos e os receptores funcionam como um tipo de complemento dos antígenos. Quando um antígeno reage com um receptor, um reconhecimento ocorre e a resposta imunológica é iniciada.

Existem dois tipos de receptores. Os linfócitos conhecidos como células B possuem receptores de imunoglobulina, também chamados de receptores de células B, que são produzidos na forma solúvel como anticorpos. Os outros linfócitos, chamados de células T,

também reconhecem os antígenos através de moléculas de proteína em sua superfície, conhecidas como receptores de células T. Ao contrário dos receptores de células B, os receptores de células T não são secretados como anticorpos e eles não reagem com antígenos livres e intactos. Ao invés disso, eles reagem com moléculas MHC (*Major Histocompatibility Complex*) que expressam fragmentos dos antígenos (chamados de peptídeos) na superfície das células contaminadas pelo agente infeccioso.

Um determinado receptor não é capaz de reagir com todos os antígenos. Essa reação é determinada pelas propriedades físicas e químicas dos antígenos e dos receptores, de modo que quanto mais complementares são suas estruturas, maior é a probabilidades de haver uma reação. Um linfócito possui aproximadamente cem mil receptores em sua superfície, entretanto, como todos esses receptores possuem a mesma estrutura, um determinado linfócito consegue reconhecer um conjunto limitado de antígenos estruturalmente relacionados. Esses antígenos estruturalmente relacionados definem o subconjunto de similaridade que o linfócito detecta. O número de receptores que reagem com um micróbio determina a afinidade que o linfócito possui ao agente infeccioso. Se uma detecção é bastante provável de ocorrer, então muitos receptores irão reagir com o antígeno, resultando em uma alta afinidade ao micróbio. Um linfócito somente é ativado se sua afinidade ao agente infeccioso exceder um determinado limiar. Conforme o limiar de afinidade de um linfócito aumenta, o número de antígenos diferentes que podem ativá-lo diminui, ou seja, o subconjunto de similaridade do linfócito fica menor. Desse modo, a detecção efetuada por uma determinada célula do sistema imunológico é aproximada dentro de um subconjunto de similaridade e vai se especializando à medida que esse subconjunto diminui (devido, por exemplo, a exposições sucessivas a um determinado agente patogênico). A Figura 3.1 ilustra a reação de um antígeno com os receptores dos linfócitos, bem como a diferença entre os receptores das células B e T.

O problema enfrentado pelo sistema imunológico é o de distinguir o *self* do *nonself*. Essa distinção é uma tarefa bastante difícil por algumas razões. Primeiro, os componentes do corpo são contruídos a partir da mesma matéria prima dos *nonself*, basicamente proteínas. Além disso, a quantidade de antígenos diferentes que o sistema imunológico deve reconhecer é muito maior que a capacidade do corpo humano de gerar receptores para esses padrões.

A habilidade de reconhecer a maioria dos antígenos requer uma grande diversidade de receptores. Essa diversidade é parcialmente atingida pela geração de receptores através de um processo genético que introduz um fator aleatório. Porém, essa geração aleatória pode resultar em receptores que reagem com proteínas *self* ao invés de *nonself*, causando problemas de auto-imunidade onde o sistema imunológico ataca o próprio corpo.

O sistema imunológico previne problemas de auto-imunidade através de um processo chamado seleção negativa. Durante esse processo, as células do sistema imunológico recém

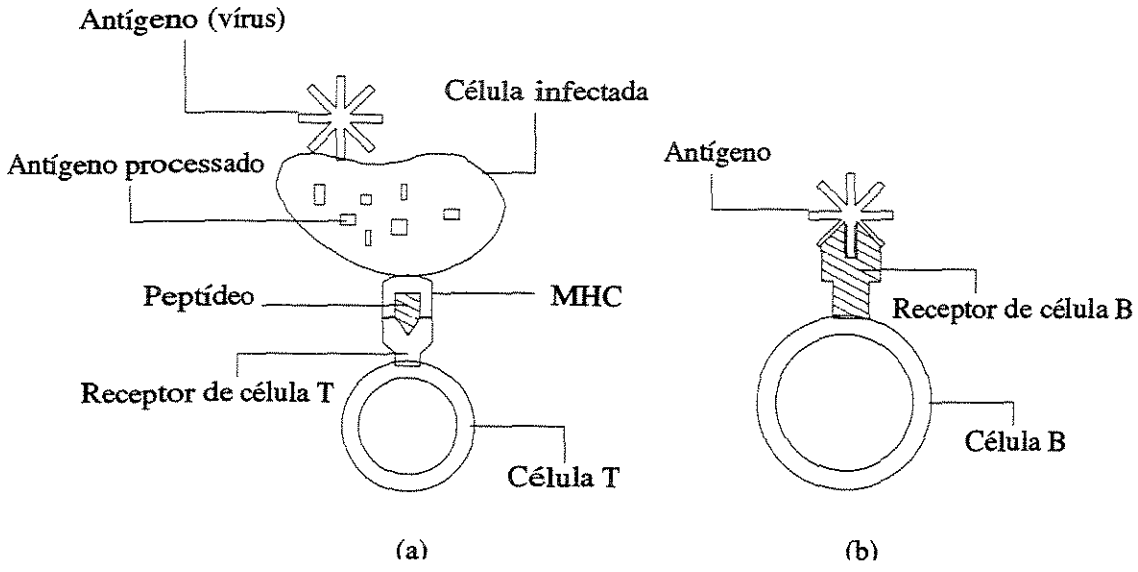


Figura 3.1: A detecção é consequência da reação entre estruturas químicas complementares. Em (a) o receptor de célula T reage com uma molécula MHC contendo um peptídeo de antígeno. Em (b) o receptor de célula B reage diretamente com um antígeno.

criadas passam por um estágio de maturação em um órgão chamado timo, onde a maioria das proteínas do corpo circulam. Qualquer célula, cujos receptores reagem com alguma dessas proteínas *self* durante a maturação, é eliminada. O processo de seleção negativa cria o conhecimento acerca do que é normal (*self*) para o sistema imunológico.

Mesmo que os receptores sejam gerados aleatoriamente, não existem células suficientes no sistema imunológico para prover uma cobertura completa contra todos os padrões de antígenos possíveis. O sistema imunológico aborda esse problema tornando sua resposta mais dinâmica e mais específica. Sua proteção é dinamizada através da circulação contínua de células do sistema imunológico pelo corpo humano e da constante renovação da população de células de defesa (as células do sistema imunológico são continuamente substituídas por novas células, com novos receptores gerados aleatoriamente). A proteção dinâmica aumenta a cobertura provida pelo sistema imunológico ao longo do tempo.

A proteção oferecida pelo sistema imunológico é especializada através de aprendizado e memória. Se o sistema imunológico detecta um agente patogênico com o qual ele nunca teve contato anterior, ele inicia uma resposta primária, onde a estrutura do agente patogênico específico é "estudada", evoluindo um conjunto de células de defesa com alta afinidade ao micróbio através de um processo chamado maturação de afinidade. A maturação de afinidade produz um grande número de células com alta afinidade a um micróbio particular, acelerando sua detecção e eliminação, uma vez que as informações codificadas

nas células adaptadas são retidas na memória imunológica. Em subsequentes exposições ao mesmo padrão de antígeno, o sistema imunológico executa uma resposta secundária mais precisa e eficiente.

Todas as células e produtos (anticorpos, por exemplo) do sistema imunológico circulam pela corrente sanguínea, tecidos e vasos linfáticos, atuando como sentinelas na busca de antígenos estranhos. Quando os receptores reagem com os antígenos, em uma concentração suficiente, um reconhecimento ocorre e um conjunto complexo de eventos é disparado, chamado de resposta imunológica, resultando na destruição dos agentes infecciosos. A resposta imunológica pode ser dividida em três fases como segue.

Fase de detecção

Como as células e produtos do sistema imunológico circulam pelo corpo humano, qualquer componente pode detectar um antígeno estranho. Quando os fagócitos encontram antígenos estranhos eles envolvem e destroem os antígenos através de um processo chamado fagocitose. Depois disso, os fagócitos expressam em sua superfície, através de moléculas MHC, fragmentos dos antígenos destruídos.

Se o antígeno estranho é "conhecido" do sistema imunológico, anticorpos específicos podem reagir diretamente com ele. Quando os anticorpos combinam-se com os antígenos, eles tornam os micróbios mais atrativos aos fagócitos e ativam uma cascata de proteínas complementares, que ajudam na destruição dos invasores. Além disso, os anticorpos impedem que os agentes infecciosos contaminem as células.

As células B podem também reagir com os antígenos estranhos, uma vez que os receptores de células B podem reagir com antígenos livres e intactos. Embora as células B envolvam e destruam os antígenos encontrados, elas não produzem anticorpos até que as células T sejam ativadas.

Fase de apresentação dos antígenos e ativação do sistema adaptativo

Fagócitos e células B, expressando em sua superfície moléculas MHC com peptídeos, atraem as células T que estavam circulando inativas (esse processo é conhecido por apresentação de antígenos). Se uma célula T reconhece um determinado complexo MHC-peptídeo, ela torna-se ativa e passa a estimular a transformação das células B em células plasmáticas, responsáveis pela produção de anticorpos. As células T ativas começam a se reproduzir, diferenciando-se em três tipos: células T ajudantes (estimulam a ação e multiplicação de fagócitos e células B), células T citotóxicas (identificam, através das moléculas MHC, e destroem as células contaminadas) e células T de memória (armazenam informações sobre o antígeno para futuras reações). As células B, estimuladas pelas células T ajudantes, multiplicam-se e diferenciam-se em dois tipos: células plasmáticas (produtoras de anticorpos) e células B de memória. Como consequência, um exército específico para

o antígeno em questão é reunido.

Fase de eliminação dos antígenos

Anticorpos específicos reagem com os antígenos, marcando-os para serem destruídos pelos fagócitos e proteínas complementares. As células T citotóxicas eliminam as células infectadas, impedindo a reprodução dos agentes patogênicos. Conforme a concentração de antígenos estranhos diminui, todos os estímulos químicos são gradualmente contidos, levando ao fim da resposta imunológica.

3.2 Analogias entre o sistema imunológico e a segurança de computadores

Com base na descrição das estruturas e funcionamento do sistema imunológico humano é possível identificar algumas analogias entre o sistema de defesa biológico e a segurança de sistemas computacionais. Essas analogias permitem enriquecer a compreensão do sistema imunológico humano como modelo de segurança.

O sistema imunológico humano protege um indivíduo de agentes patogênicos potencialmente mortais, incluindo bactérias, vírus, parasitas e toxinas. Seu papel no corpo humano é análogo ao de sistemas de segurança em computadores — proteger uma determinada entidade, localizada em um ambiente hostil e sujeito a falhas, contra a ação de atacantes.

Os imunologistas tradicionalmente descrevem o problema resolvido pelo sistema imunológico como o problema de distinguir o normal (*self*) do estranho (*nonself*) e eliminar o que for estranho. O *self* é tido como as células e moléculas do corpo, e o *nonself* é qualquer material estranho, particularmente bactérias, parasitas e vírus. O problema de proteger sistemas computacionais de intrusões pode ser visto, assim como no caso do sistema imunológico, como um problema de distinguir o *self* do *nonself*. A definição de *self* em um sistema computacional pode ser feita, por exemplo, em termos de padrões de acesso à memória em uma máquina, pacotes de rede entrando e saindo de um sistema, comportamento coletivo de uma rede, tráfego através de um roteador, sequências de instruções de um programa ou padrões de comportamento de usuários. Enfim, qualquer mecanismo que permita a identificação de algo estranho no sistema monitorado como, por exemplo, usuários não-autorizados, código externo na forma de vírus ou *worms*, ações ilegítimas de usuários autorizados ou dados corrompidos.

Uma definição estável daquilo que é normal em um sistema computacional pode resultar, assim como no sistema imunológico humano, em uma noção mais sofisticada de identidade e proteção, o que um grande número de administradores de sistemas não possui

(nem todos conhecem o comportamento usual de seu sistema).

O sistema imunológico é um sistema de defesa complexo e facinante, que possui uma série de propriedades que podem ser de particular interesse para sistemas de segurança de computadores. Entre essas propriedades podem ser citadas as seguintes:

- **diversidade**: o sistema imunológico de cada indivíduo é único, de modo que as vulnerabilidades diferem de um sistema para outro;
- **detecção distribuída**: os detectores usados pelo sistema imunológico são pequenos e eficientes, altamente distribuídos e funcionalmente independentes, não estando sujeitos a um controle centralizado (representando um ponto central de falha);
- **detecção aproximada**: uma detecção mais flexível permite que o sistema imunológico aborde o problema de falta de recursos para detectar todos os padrões *nonself* possíveis;
- **detecção de agentes desconhecidos**: o sistema imunológico é capaz de detectar e reagir a agentes patogênicos aos quais nunca foi exposto;
- **adaptabilidade**: o sistema imunológico pode aprender as estruturas dos agentes patogênicos, armazenando-as em sua memória imunológica, de modo que futuras respostas a um mesmo micróbio são mais eficientes;

É possível ainda identificar semelhanças entre características do sistema imunológico humano e técnicas de detecção de intrusão. Por exemplo, a geração aleatória de receptores unida ao processo de seleção negativa cria detectores capazes de reagir a tudo aquilo que é estranho ao conjunto de proteínas *self* do corpo humano. Esse conhecimento sobre o que é normal e a detecção daquilo que é diferente do normal constituem características de detecção de intrusão por anomalia. Nesse mesmo sentido, o processo de seleção negativa corresponde ao procedimento de treinamento de um IDS por anomalia, onde os perfis históricos de comportamento normal são usados para gerar um conceito de anomalia.

Outro exemplo de relação com técnicas de detecção de intrusão é o processo de maturação de afinidade e a memória imunológica. A maturação de afinidade especializa a detecção para um determinado agente patogênico conhecido e a memória imunológica abriga os detectores altamente especializados. Desse modo, o sistema imunológico reage de maneira mais rápida e eficiente a micróbios conhecidos. Essa característica de aprendizado do sistema imunológico está relacionada à compreensão daquilo que é conhecidamente indesejável e a memória imunológica é um tipo de base de dados para assinaturas de agentes patogênicos. Essas características do sistema imunológico representam fatores relacionados com a detecção de intrusão por mau uso.

Além disso, o sistema imunológico e os sistemas de detecção de intrusão enfrentam problemas semelhantes, incluindo as doenças de auto-imunidade (problema análogo à detecção de falso-positivos), a necessidade eventual de ajuda externa por vacinas ou soros (semelhante à necessidade de manutenção e refinamento de um IDS), imperfeições no processo de seleção negativa (problema análogo à dificuldade de geração de perfis de treinamento ótimos para IDSs por anomalia) e ataques ao próprio sistema de defesa.

Embora existam muitas diferenças entre organismos vivos e sistemas computacionais, as semelhanças apresentadas nesta seção são bastante evidentes. A analogia com o sistema imunológico pode contribuir com um importante ponto de vista para a obtenção de sistemas computacionais mais seguros, levando ao desenvolvimento de sistemas de defesa com conjuntos de suposições, embasamentos e princípios organizacionais potencialmente diferentes dos sistemas atuais.

3.3 Trabalhos correlatos

Segundo Dasgupta [15, 13], sistemas biológicos tais como os seres humanos podem ser considerados como sofisticados sistemas de processamento de informações, provendo inspiração para os diversos campos da ciência e engenharia. Dasgupta classifica os sistemas de processamento de informações baseados em processos biológicos em três áreas: sistemas nervosos (redes neurais), sistemas genéticos (algoritmos evolucionários) e sistemas imunológicos (sistemas imunológicos artificiais) [15].

Um número crescente de pesquisas tem se dedicado à área de sistemas imunológicos artificiais[1] (também chamada de imunologia computacional), muitas delas propondo modelos imunológicos computacionais para a resolução de problemas que incluem diagnóstico de falhas, reconhecimento de imagens e padrões, detecção de vírus e segurança de dados e sistemas computacionais [15, 13, 18].

Entre as várias áreas de aplicação de sistemas imunológicos artificiais, a detecção de intrusão é um campo de pesquisa onde a analogia é bastante imediata. A conexão entre imunologia e segurança de computadores teve início em 1994 com as publicações [29, 52], desencadeando uma série de outros trabalhos. Alguns desses trabalhos são apresentados na sequência.

3.3.1 Universidade do Novo México

Grande parte das pesquisas envolvendo a analogia entre segurança de computadores e sistemas imunológicos é desenvolvida na Universidade do Novo México [105], onde as idéias

[1] Uma coletânea de referências na área de sistemas imunológicos artificiais pode ser encontrada em [17].

extraídas da imunologia são aplicadas em quatro grandes áreas de pesquisa[2]: um método de detecção de intrusão baseado em uma máquina, um sistema de detecção de intrusão baseado em rede, um algoritmo distribuído de detecção de mudanças e um método para introdução intencional de diversidade em sistemas computacionais para reduzir vulnerabilidades.

3.3.1.1 Princípios de um sistema imunológico computacional

Segundo Somayaji, Hofmeyr e Forrest [98], o estudo do sistema imunológico humano revela um conjunto de princípios organizacionais que podem orientar o desenvolvimento de sistemas de segurança de computadores. Esse conjunto de princípios é apresentado como segue:

- **distributabilidade**: os detectores do sistema imunológico são capazes de determinar localmente a presença de um infecção. Não existe uma coordenação central e, portanto, um ponto único de falha;
- **multi-camadas**: nenhum mecanismo do sistema imunológico garante isoladamente a segurança do corpo. Ao invés disso, esses mecanismos constituem um sistema integrado de defesa em que um grande número de células e moléculas agem cooperativamente, cada qual exercendo uma função especializada;
- **diversidade**: através da diversidade de características dos sistemas, as vulnerabilidades de um sistema são pouco prováveis de serem encontradas em todos os outros. O sistema imunológico de cada indivíduo é único, de modo que um determinado agente patogênico pode ser fatal para uma pessoa e inofensivo para outra. Essa diversidade pode ser atingida de duas formas: os sistemas de proteção podem ser únicos (como no caso do sistema imunológico humano) ou os sistemas protegidos podem ser diversificados (maiores detalhes são apresentados na Seção 3.3.1.5);
- **disponibilidade**: nenhuma célula isolada do sistema imunológico é essencial, podendo ser substituída. A morte de células do sistema imunológico é balanceada pela produção de novos componentes;
- **autonomia**: o sistema imunológico, como um todo, não requer um gerenciamento ou manutenção externos. Ele pode autonomamente detectar, classificar e eliminar

[2] Além das aplicações na área da segurança de computadores, os projetos desenvolvidos na Universidade do Novo México também envolvem a modelagem de fenômenos imunológicos reais, como o processo de maturação de afinidade (envolvendo algoritmos genéticos) e a memória imunológica. Maiores detalhes podem ser encontrados em [30] ou na URL http://www.cs.unm.edu/~forrest/papers.html (disponível em janeiro de 2003).

os agentes patogênicos, bem como efetuar reparos através da substituição de suas células danificadas;

- **adaptabilidade**: o sistema imunológico possui a capacidade de aprendizado na detecção de novos agentes patogênicos, armazenando a habilidade adquirida de reconhecimento na memória imunológica. A capacidade de reconhecimento especializa-se a cada agente patogênico similar reconhecido, tornando respostas futuras mais eficientes se comparadas às anteriores;

- **nenhuma camada de segurança**: qualquer célula do corpo humano pode ser atacada por um agente patogênico, incluindo as células que compõem o sistema imunológico. Entretanto, os linfócitos podem proteger o corpo contra linfócitos comprometidos por uma infecção;

- **mudança dinâmica de cobertura**: o sistema imunológico realiza uma troca, no tempo e no espaço, de seu conjunto de detectores. Como ele não pode manter um conjunto de detectores grande o suficiente para cobrir o espaço de todos os agentes patogênicos, o sistema imunológico mantém, em um determinado momento, uma amostra aleatória de seu repertório de detectores circulando pelo corpo. Esse repertório é constantemente modificado através da morte e reprodução de células;

- **identificação por comportamento**: em uma infecção, uma célula contaminada é identificada através das moléculas MHC, em sua superfície, contendo fragmentos de antígenos (peptídeos). Desse modo, os peptídeos podem ser vistos como indicadores de comportamento;

- **detecção por anomalia**: o sistema imunológico possui a habilidade de detectar agentes patogênicos com os quais nunca teve contato, em outras palavras, ele realiza detecção por anomalia;

- **detecção imperfeita**: ao aceitar uma detecção imperfeita (aproximada), o sistema imunológico aumenta a flexibilidade com a qual ele pode alocar os recursos. Por exemplo, menos detectores específicos podem responder a uma variedade maior de padrões, entretanto, eles são menos eficientes na detecção de um agente patogênico particular;

- **o jogo de números**: o sistema imunológico regula o número de células de maneira a desenvolver um contra-ataque suficiente para a quantidade de agentes patogênicos.

Segundo Somayaji, Hofmeyr e Forrest, essas propriedades podem ser vistas como princípios para o desenvolvimento de um sistema imunológico computacional, podendo ajudar na concepção de sistemas computacionais mais seguros.

3.3.1.2 Um método de detecção de intrusão baseado em uma máquina

Forrest, Hofmeyr e Somayaji introduziram em [31] um método de detecção de intrusão por anomalia, baseado em uma máquina, que explora a analogia com o sistema imunológico humano. Posteriormente, essa linha de pesquisa originou uma série de outros trabalhos, podendo ser citadas as publicações [32, 45, 108, 97].

Segundo os proponentes do método de detecção de intrusão, a discriminação entre comportamento normal e anormal deve ser baseada em alguma estrutura característica que é ao mesmo tempo compacta e universal no sistema protegido, além de ser sensível à maioria dos comportamentos indesejáveis. No método proposto em [31], a definição de *self* (comportamento normal) é feita em termos de sequências curtas de chamadas ao sistema executadas por processos privilegiados.

O método é baseado na construção de uma base de dados do comportamento normal de cada programa de interesse. Cada base de dados é específica para uma determinada arquitetura, versão e configuração de *software*, política de administração e padrões de uso. Depois de construída, a base de dados é utilizada para monitorar o comportamento do programa relacionado. As sequências de chamadas ao sistema da base de dados formam o conjunto de padrões normais do processo, e as sequências não encontradas na base de dados indicam anomalias no comportamento do processo.

O método proposto em [31] apresenta dois estágios. No primeiro estágio, são extraídos traços de comportamento normal dos processos e construídas as bases de dados. O algoritmo utilizado para tal é bastante simples: os traços de chamadas de sistema gerados por um determinado processo são observados e todas as sequências únicas de tamanho k são armazenadas na base de dados do processo. Por exemplo, supondo o seguinte traço de chamadas de sistema:

```
open, read, mmap, mmap, open, read, mmap
```

Uma janela de tamanho k é então deslizada através do traço, registrando cada sequência única de tamanho k encontrada. Supondo $k = 3$, obtém-se as seguintes sequências:

```
open, read, mmap
read, mmap, mmap
mmap, mmap, open
mmap, open, read
```

Os parâmetros das chamadas de sistema são ignorados e os processos filhos de um determinado programa são rastreados individualmente. Por questões de eficiência, as sequências de chamadas de sistema são armazenadas como árvores, com cada árvore iniciando em uma chamada de sistema particular.

No segundo estágio do método de detecção, as sequências de chamadas ao sistema executadas pelos processos são comparadas com os padrões armazenados em suas bases de dados. Se uma sequência não é encontrada na base de dados do processo, é registrado um desacordo. O número de desacordos encontrados em um traço é usado para determinar o grau de anomalia do comportamento do processo monitorado.

Segundo os proponentes, embora esse método de detecção não proveja um discriminante criptograficamente forte ou completamente confiável entre comportamento normal e anormal, ele é muito mais simples que outros métodos propostos e pode ser aplicado em uma ferramenta leve para checagem em tempo real de código em execução. Vários estudos práticos foram realizados e os resultados podem ser encontrados nas publicações mencionadas anteriormente.

Nesse sentido, Somayaji e Forrest apresentam em [97] um sistema chamado pH (advindo do termo *process homeostasis*) que, segundo os autores, pode detectar e interromper intrusões antes que o sistema alvo seja comprometido. O sistema pH monitora todos os processos em execução no nível de chamada de sistemas (utilizando o método descrito anteriormente) e responde a anomalias inserindo atrasos ou abortando as chamadas de sistema. Um conjunto de resultados experimentais iniciais é apresentado em [97].

3.3.1.3 Um sistema de detecção de intrusão baseado em rede

Hofmeyr e Forrest introduziram em [43, 42] a arquitetura de um sistema imunológico artificial. Essa arquitetura, chamada de ARTIS (*Artificial Immune System*), é aplicada à segurança de computadores na forma de um sistema de detecção de intrusão baseado em rede, denominado LISYS (*Lightweight Intrusion Detection System*) [44, 5].

O sistema LISYS é um IDS distribuído que monitora o tráfego em uma rede TCP/IP local. Nesse domínio, o *self* é definido como sendo o conjunto de conexões normalmente observadas na rede. Cada conexão é caracterizada por uma tripla (endereço IP de origem, endereço IP de destino e porta de serviço). Na representação do sistema LISYS, essa informação é compactada em uma cadeia de 49 bits que unicamente define a conexão. Nesse sentido, o *nonself* também é um conjunto de conexões (usando a mesma representação de 49 bits), consistindo daquelas conexões, potencialmente um número enorme, que não são normalmente observadas na rede.

O sistema LISYS apresenta um único tipo de detector, que combina propriedades de várias células do sistema imunológico, podendo assumir diferentes estados. Cada detector é representado por uma cadeia de 49 bits e um conjunto de estados. O IDS depende de várias características imunológicas, ressaltando-se o processo de seleção negativa e a detecção aproximada. Os detectores são gerados aleatoriamente e treinados durante o processo de seleção negativa. A detecção é implementada através do casamento parcial de *strings* (casamento de r-bits contíguos), reduzindo o número necessário de detectores.

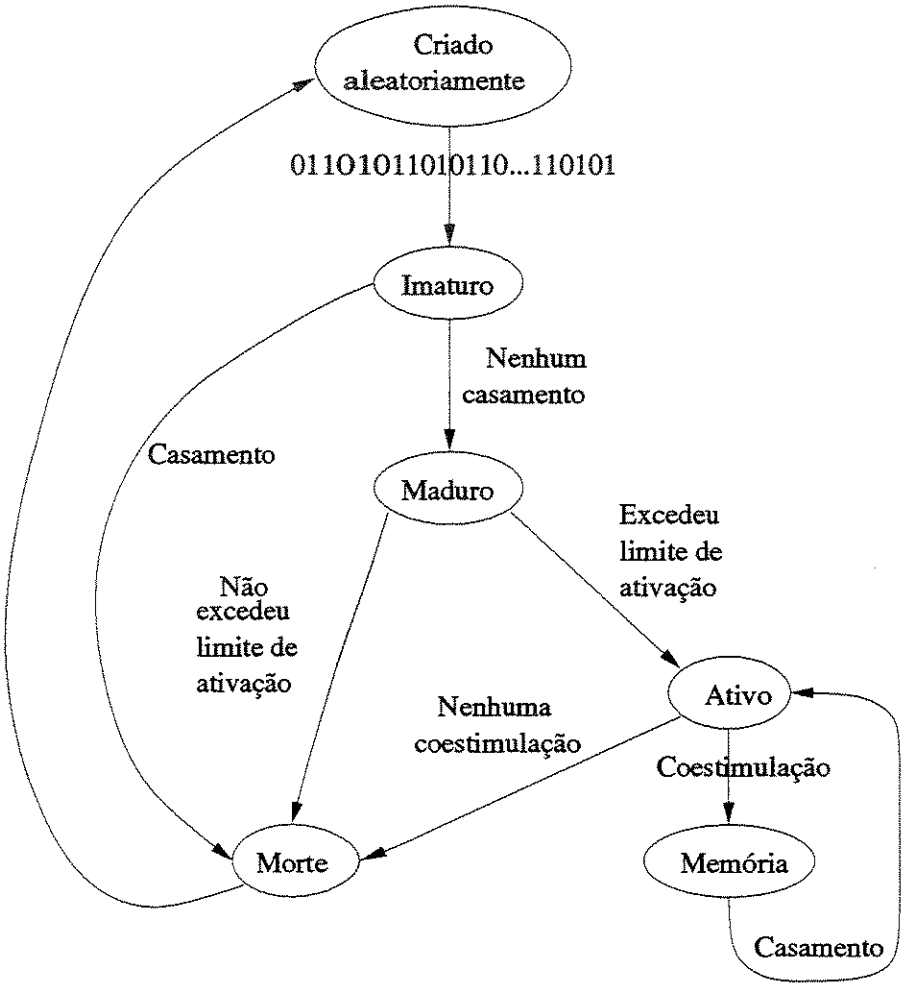


Figura 3.2: Ciclo de vida de um detector do sistema LISYS.

A Figura 3.2 resume o ciclo de vida de um detector. Um detector é inicialmente criado aleatoriamente e, então, permanece imaturo por um certo período de tempo. Se o detector casar, uma única vez enquanto imaturo, com alguma cadeia representando uma conexão ele é substituído por um novo (também gerado aleatoriamente). Se o detector sobreviver ao processo de seleção negativa, ele existirá por um tempo finito. Ao final de sua vida ele é substituído por um novo detector, a menos que ele tenha ultrapassado um limite de ativação (número mínimo de casamentos para tornar-se ativo) e tenha se tornado um detector de memória. No estágio maduro, o detector é ativado se exceder o limite de ativação. Se o detector estiver ativo e não receber coestimulação (do administrador de segurança, por exemplo), ele morre. Entretanto, se houver coestimulação o detector torna-se um detector de memória, com um tempo de vida indefinido. Um detector de

memória necessita de apenas um casamento para ser ativado.

Vários experimentos controlados foram conduzidos e os resultados práticos podem ser encontrados nas publicações mencionadas anteriormente.

3.3.1.4 Um algoritmo distribuído de detecção de mudanças

Forrest, Perelson, Allen e Cherukuri introduziram em [29] um algoritmo distribuído de detecção de mudanças, baseado na geração de células T do sistema imunológico. Embora os testes iniciais tenham sido aplicados na detecção de vírus de computador, o algoritmo pode ser estendido a outros problemas de detecção de mudanças.

Segundo o algoritmo, um conjunto de dados considerado *self* é monitorado contra mudanças ilegítimas, como segue:

1. Gere um conjunto de detectores que falham em reconhecer o *self*.

2. Use os detectores para monitorar os dados protegidos.

3. Sempre que um detector for ativado (reconhecer algo), uma mudança deve ter ocorrido em um dado protegido, e a localização da mudança é conhecida.

O *self* é definido como um conjunto de *strings* de mesmo tamanho, geradas pela segmentação dos dados protegidos em *substrings* de igual comprimento. Cada detector é também uma *string* de tamanho igual ao *self*.

O reconhecimento por parte dos detectores é modelado como o casamento entre pares de *strings*. Um casamento perfeito entre duas *strings* de mesmo tamanho indica que os símbolos são idênticos em cada posição da *string*. Entretanto, o algoritmo utiliza um tipo especial de casamento, chamado casamento de r-bits contíguos (mais plausível com o modelo imunológico). Desse modo, o algoritmo busca por r posições contíguas, cujos símbolos são idênticos, em um par de *strings*. Os detectores são gerados aleatoriamente, e passam pelo processo de seleção negativa, eliminando aqueles que casam com alguma *string* do conjunto *self*.

Segundo os autores, o algoritmo possui algumas propriedades interessantes, incluindo a facilidade de distribuição e a tolerância a ruídos (dependendo da regra utilizada no casamento das *strings*). Entretanto, a maior fraqueza do algoritmo original estava no alto custo para a geração dos detectores (de ordem exponencial). Essa fraqueza foi abordada em [22], onde dois algoritmos lineares de geração de detectores foram introduzidos.

Vários experimentos foram realizados com o algoritmo e os resultados podem ser encontrados em [29, 22, 23, 16].

3.3.1.5 Um método para introdução intencional de diversidade em sistemas computacionais para reduzir vulnerabilidades

A relevância da diversidade para a segurança de computadores foi reconhecida no início de 1989, após o incidente com o *worm* de novembro de 1988 (*Morris' Worm*), quando foi observado que apenas alguns tipos de máquinas eram vulneráveis à infecção [24].

Forrest, Somayaji e Ackley argumentam em [33] que os benefícios da diversidade em sistemas computacionais tem sido desprezados. Os autores buscam um paralelo com sistemas biológicos, principalmente o sistema imunológico humano, onde a diversidade ajuda a controlar a disseminação de doenças dentro de uma população. Segundo os autores, todas as vantagens da uniformidade em sistemas computacionais tornam-se potenciais fraquezas quando elas podem ser exploradas por um atacante, pois uma vez criado um método para penetrar a segurança de um sistema, todos os sistemas com as mesmas características tornam-se igualmente vulneráveis.

O objetivo da introdução de diversidade é prevenir a disseminação de ataques, tornando as intrusões mais difíceis de serem replicadas. Vários métodos são apresentados em [33], focados em consistências desnecessárias, incluindo a adição ou remoção de código não-funcional, a reordenação de código e a alocação de memória. Tal diversidade, no entanto, deve preservar a funcionalidade dos programas e causar um impacto mínimo na sua conveniência, usabilidade e eficiência.

Como uma simples demonstração das idéias, os autores implementaram um método simples para aleatorização da quantidade de memória alocada em uma pilha, mostrando que essa abordagem afeta a disseminação de um ataque simples de *buffer overflow*. Os resultados dessa demonstração prática são apresentados em [33].

3.3.2 Sistema imunológico computacional para detecção e análise de vírus de computador

Kephart apresenta em [52] um sistema imunológico computacional para detecção e análise de vírus de computador. Segundo o autor, o sistema proposto é utilizado para automatizar a análise de vírus de computador nos laboratórios da IBM e muitas de suas características são incorporadas no *software* anti-vírus da referida empresa.

O processo pelo qual o sistema proposto por Kephart determina se um novo *software* contém um vírus possui vários estágios, ilustrados na Figura 3.3. Primeiramente, o sistema imunológico computacional executa periodicamente ou continuamente um conjunto de detectores para determinar se alguma anomalia está presente no sistema monitorado. Esses detectores incluem os seguintes componentes:

- **monitores de integridade**: utilizam *checksums* para identificar mudanças em

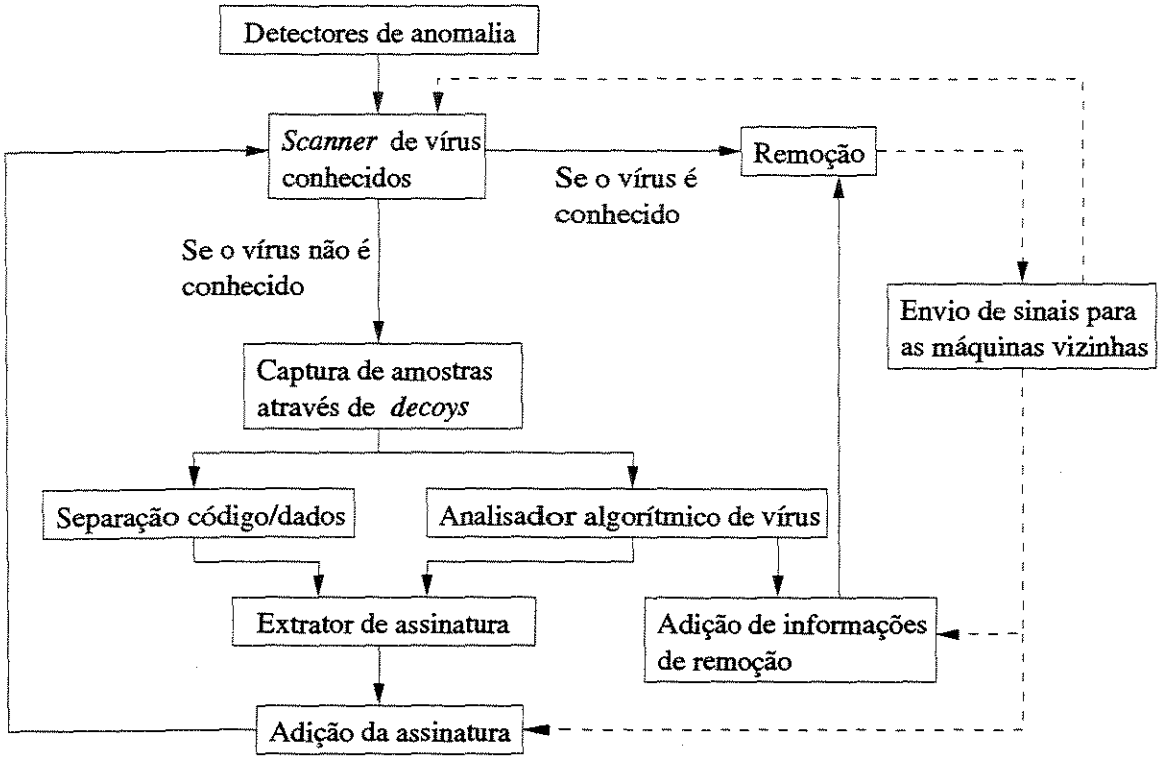


Figura 3.3: Sistema imunológico computacional de Kephart.

programas e arquivos de dados;

- **monitores de atividade**: conhecem os comportamentos dinâmicos típicos dos vírus;
- **heurísticas**: examinam a natureza estática das mudanças ocorridas para determinar se elas possuem características virais;

Se algum dos detectores é ativado, o sistema imunológico computacional executa um *scanner* para determinar se a anomalia pode ser atribuída a um vírus conhecido. Um vírus particular é reconhecido através da detecção exata ou *fuzzy* de uma sequência relativamente curta de bytes do vírus (chamada de assinatura). Se a anomalia é atribuída a um vírus conhecido, este é localizado e removido. Caso contrário, o sistema imunológico computacional executa uma série de atividades para adquirir conhecimento (assinatura e método de remoção) acerca do novo vírus.

Nesse sentido, o sistema imunológico computacional tenta fazer com que o provável vírus infecte um conjunto diverso de programas "armadilha", chamados de *decoys*. Os

decoys infectados (determinado através de *checksums*) são então processados pelo analisador algorítmico de vírus, que extrai informações úteis para a remoção do vírus (método de infecção, por exemplo). Em seguida, o extrator automático de assinatura recebe como entrada todas as sequências de bytes, determinadas como sendo código executável, contidas nos *decoys* infectados, seleciona uma assinatura e provê uma estimativa do número máximo de falso-positivos da assinatura aplicada a um *benchmark* de teste. A assinatura extraída e as informações de remoção do vírus são armazenadas nas bases de dados de vírus conhecidos.

O sistema imunológico computacional possui ainda um mecanismo de distribuição de informações, de modo que o conhecimento adquirido acerca de um novo vírus é disseminado entre as máquinas vizinhas de uma rede, impedindo a proliferação do vírus.

3.3.3 Modelo imunológico artificial para detecção de intrusão baseada em rede

Kim e Bentley apresentam em [54] uma revisão da analogia entre o sistema imunológico humano e sistemas de detecção de intrusão baseados em rede. Nesse trabalho, os autores identificam um conjunto de requisitos básicos para um bom IDS baseado em rede, derivando, a partir deles, três grandes princípios de modelagem: distributividade, auto-organização e baixo consumo de recursos. Nesse sentido, Kim e Bentley introduzem em [54] um conjunto de características do sistema imunológico humano que satisfazem esses três princípios de modelagem.

Em um trabalho posterior [53], Kim e Bentley propõem um modelo imunológico artificial para detecção de intrusão baseada em rede. Tal modelo consiste de três estágios evolucionários diferentes: evolução da biblioteca de genes (componentes que formam os detectores do modelo), seleção negativa e seleção clonal. A arquitetura geral desse modelo imunológico artificial é ilustrada na Figura 3.4.

O modelo imunológico artificial de Kim e Bentley é composto por um IDS primário, localizado na entrada da rede monitorada (roteador de entrada, por exemplo), e vários IDSs secundários, localizados nas máquinas da rede. O IDS primário é responsável pela geração de conjuntos de detectores (através da combinação dos genes). Cada conjunto individual de detectores descreve padrões anormais de tráfego de pacotes na rede. Os conjuntos de detectores são únicos e são transferidos para cada máquina. Em cada IDS secundário, os detectores atuam como processos que monitoram se padrões anormais de tráfego de rede são observados nos perfis de tráfego gerados localmente na máquina monitorada. O IDS primário e cada IDS secundário possuem comunicadores para permitir a transferência de informações entre eles. O funcionamento do modelo imunológico artificial de Kim e Bentley é ilustrado na Figura 3.5.

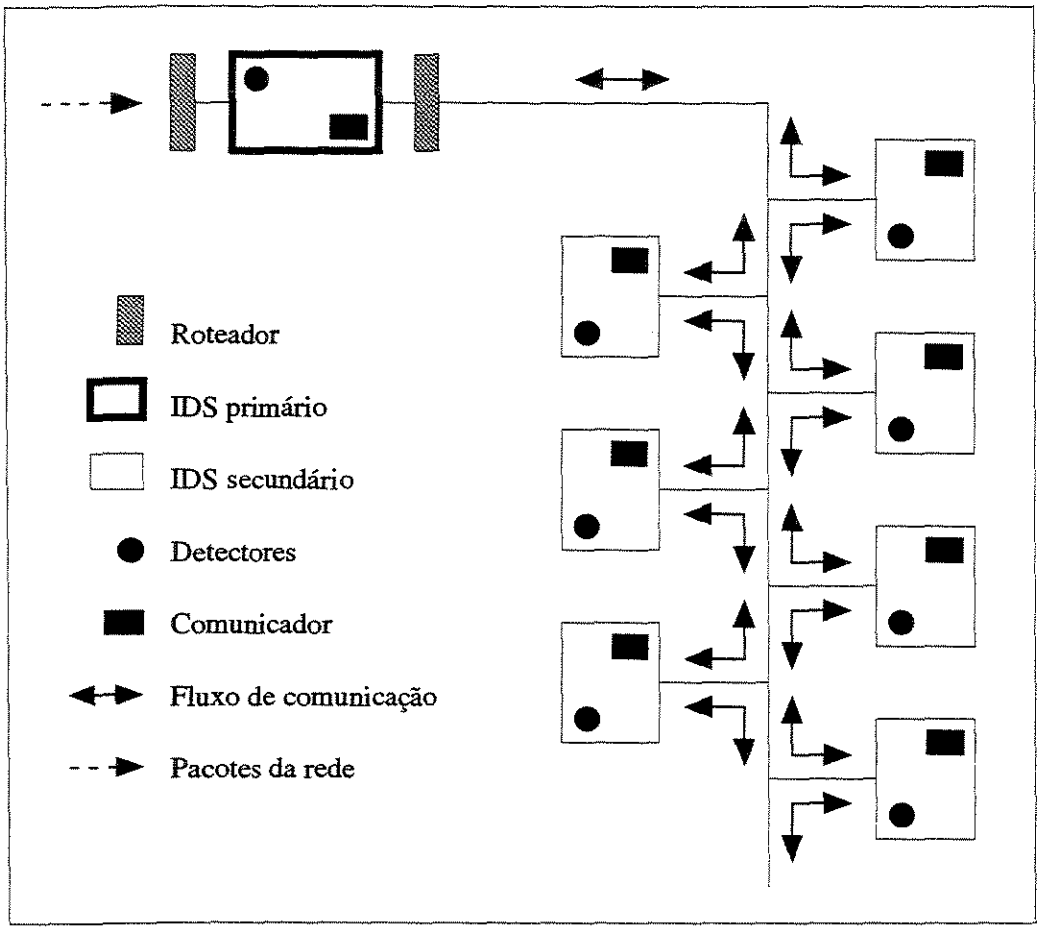


Figura 3.4: Arquitetura do modelo imunológico artificial de Kim e Bentley para detecção de intrusão baseada em rede.

O IDS primário realiza as dois primeiros processos evolucionários: evolução da biblioteca de genes e seleção negativa. No estágio de evolução da biblioteca de genes (estágio 1 na Figura 3.5) o objetivo é obter um conhecimento geral acerca dos detectores efetivos. No estágio de seleção negativa (estágio 2 na Figura 3.5) o objetivo é a geração de detectores diversos, que não reagem com o *self*, e a transferência de conjuntos desses detectores para as máquinas da rede.

No primeiro estágio, uma biblioteca de genes é gerada e mantida por um processo de evolução. A biblioteca de genes do modelo imunológico artificial armazena os genes potenciais dos detectores, aos quais são aplicados diversos mecanismos genéticos para geração de novos detectores. Esses genes potenciais são os campos selecionados para descrever padrões anormais nos perfis de tráfego. Os genes iniciais podem ser determinados pelos

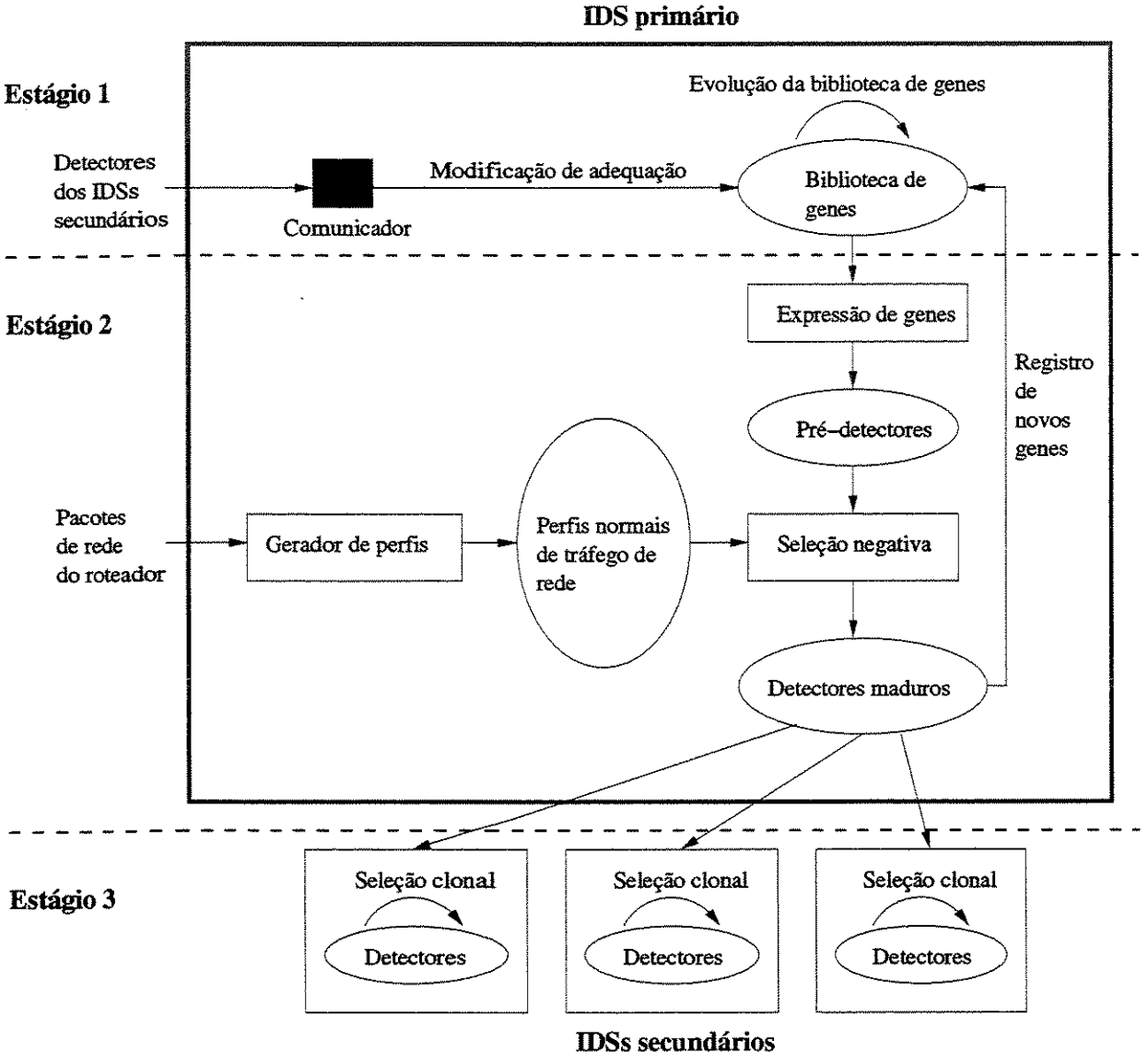


Figura 3.5: Funcionamento do modelo imunológico artificial de Kim e Bentley.

valores desses campos em uma simulação controlada de uma intrusão conhecida.

A partir desses genes, diversos pré-detectores são gerados e testados contra os perfis de tráfego normal da rede, eliminando aqueles que reagem com o *self*. Os detectores maduros, juntamente com os perfis de tráfego normal, são então transferidos para as máquinas da rede e usados para realimentar a biblioteca de genes.

Os IDSs secundários realizam o último processo evolucionário: seleção clonal. Suas principais tarefas são a detecção de intrusões com um número limitado de conjuntos de detectores (detecção aproximada) e a clonagem de detectores que estão operando satisfatoriamente, produzindo detectores de memória e orientando a evolução da biblioteca de genes do IDS primário. Para realizar a detecção de anomalias no tráfego da rede, os IDSs secundários comparam seus conjuntos de detectores com os perfis de tráfego normal, transferidos pelo IDS primário. Primeiramente o grau de casamento entre os campos de um detector e o perfil normal é medido. Quando esse grau de casamento excede um limiar pré-definido, o detector informa ao comunicador do IDS secundário e inicia o processo de seleção clonal. Quando um detector identifica um tráfego anormal na rede, ele permanece no respectivo IDS secundário como detector de memória e passa a gerar clones, que podem ser transferidos para os outras máquinas da rede. Um detector de memória detecta mais rapidamente a mesma intrusão que o gerou e os seus genes são adicionados à biblioteca de genes do IDS primário (ou seus valores de adequação são aumentados, caso eles já existam na biblioteca). A decisão final sobre a ocorrência de uma intrusão é feita de acordo com as decisões coletivas de várias máquinas da rede.

3.3.4 Sistema de detecção de intrusão baseado em agentes

Dasgupta apresenta em [14] um sistema de detecção de intrusão baseado em agentes que emprega uma série de características do sistema imunológico humano. Nessa abordagem, os agentes movem-se através das máquinas de uma rede, monitorando uma série de parâmetros em diferentes níveis (incluindo os pacotes da rede, os processos, os recursos do sistema e os usuários). Além disso, os agentes podem interagir entre si, de maneira hierárquica, permitindo a execução de complexas tomadas de decisões. Os agentes são divididos em três tipos, com funções específicas, descritos como segue:

- **agentes de monitoramento**: monitoram vários parâmetros simultaneamente em múltiplos níveis. Alguns desses agentes utilizam detecção por anomalia, através da identificação de desvios em relação aos perfis de comportamento normal. Outros possuem o conhecimento acerca de intrusões conhecidas;

- **agentes de comunicação**: atuam como distribuidores de mensagens, promovendo a comunicação entre os demais agentes;

- **agentes de decisão e ação**: são responsáveis pelas tomadas de decisões (determinação de quais agentes precisam ser ativados) e execução de tarefas específicas de acordo com a política de segurança adotada. Esses agentes podem ativar um conjunto de agentes de resposta (agentes ajudantes, eliminadores ou supressivos), dependendo da natureza e severidade da intrusão;

Durante a maior parte do tempo, os agentes de monitoração encontram-se em movimento através da rede, checando os diversos parâmetros monitorados. Se alguma situação é detectada por algum agente de monitoramento, uma ação coordenada, envolvendo outros agentes da vizinhança, é iniciada com o objetivo de entender o evento e tomar uma decisão. Uma vez tomada a decisão, os agentes de resposta são ativados para a execução de ações específicas.

Alguns detalhes da implementação de um protótipo inicial são apresentados em [14].

3.4 Conclusão

Este capítulo apresentou a analogia existente entre o sistema imunológico humano e a segurança de computadores, introduzindo o mecanismo de defesa humano como modelo para o desenvolvimento de um sistema de segurança computacional, além de alguns dos principais trabalhos em torno da analogia. A partir do que foi exposto neste capítulo, é possível identificar semelhanças entre características do sistema imunológico humano e técnicas de detecção de intrusão, quer seja por anomalia quanto por mau uso.

Capítulo 4

Arquitetura de um sistema de segurança imunológico

A partir da discussão apresentada no capítulo anterior, acerca do funcionamento do sistema imunológico humano e de sua relação com a segurança de computadores, este capítulo apresenta uma abordagem diferente para a aplicação dessa analogia no desenvolvimento de um sistema de segurança.

Grande parte das pesquisas em imunologia computacional aplicada à detecção de intrusão, tais como [31, 43, 29, 53], são baseadas na geração de receptores (de maneira aleatória ou através da permutação de genes) e no processo de seleção negativa daqueles receptores que não reagem com o *self*. Essa abordagem é usada no desenvolvimento de técnicas de detecção de intrusão por anomalia. Em linhas gerais, essas técnicas consistem na produção de uma base de dados daquilo que é considerado normal no sistema monitorado e na geração de receptores que são testados contra essa base de dados. Aqueles receptores que falham em detectar qualquer entrada da base de comportamento normal são usados para monitorar o sistema, assumindo que a ativação de algum desses receptores indica a ocorrência de uma situação anormal.

A nova abordagem apresentada neste capítulo, inicialmente proposta pelo grupo de imunologia computacional do LAS-IC-UNICAMP em [84], considera o fato de que o sistema imunológico humano possui várias características de detecção por mau uso em adição às características de detecção por anomalia exploradas nas pesquisas anteriores. Tais características de detecção por mau uso incluem, por exemplo, o reconhecimento de padrões específicos de antígenos, a memória de patógenos conhecidos e a detecção mais eficiente e de alta afinidade.

De fato, a memória imunológica é uma base de assinaturas de agentes patogênicos conhecidos e os anticorpos e receptores de células B representam padrões específicos de antígenos com os quais o sistema imunológico já teve contato. Esses componentes do

sistema imunológico, unidos ao processo de maturação de afinidade, permitem que ele responda mais eficientemente a novos ataques de um invasor conhecido. Além disso, tais componentes viabilizam uma capacidade interessante do sistema imunológico humano: a alteração autônoma da compreensão daquilo que é conhecidamente indesejável.

Baseado nessa abordagem, o grupo de imunologia computacional do LAS-IC-UNICAMP propôs [84] um modelo de IDS híbrido baseado no sistema imunológico humano (denominado modelo de IDS imunológico). Tal modelo conceitual possui a capacidade de detectar e caracterizar um ataque, elaborar um plano de resposta especializado e efetuar o contra-ataque. Além disso, o modelo agrega as características de aprendizado e adaptação do sistema imunológico humano, podendo reagir a ataques desconhecidos e melhorar sua reação a exposições subsequentes de um mesmo ataque.

O modelo de IDS imunológico é apresentado em detalhes na Seção 4.1 e sua relação com o sistema imunológico humano é discutida na Seção 4.2. Em seguida, na Seção 4.3, é apresentada a idéia inicial de uma implementação particular do modelo conceitual de IDS imunológico, denominada ADenoIdS. Por fim, a Seção 4.4 evidencia o papel da forense computacional no modelo de IDS imunológico.

4.1 Um modelo de IDS híbrido baseado no sistema imunológico humano

A Figura 4.1 ilustra o modelo de IDS híbrido baseado no sistema imunológico humano desenvolvido pelo grupo de imunologia computacional do LAS-IC-UNICAMP [84], apresentando seus componentes e o fluxo de informações entre eles. É importante mencionar que o modelo descreve a arquitetura geral de um sistema de segurança imunológico, em um nível conceitual, podendo ser implementado de várias formas (como o sistema ADenoIdS, apresentado na Seção 4.3), com diferentes estratégias de monitoramento e técnicas de análise. Os diversos componentes do modelo são detalhados como segue.

Fonte de dados

A fonte de dados é responsável pela coleta de informações e suprimento de um fluxo de registros de eventos ao sistema de filtragem. A natureza das informações coletadas pode variar de acordo com a estratégia de monitoramento adotada[1]: baseada em uma máquina, em rede, em aplicação ou em alvo [3]. O modelo de IDS imunológico é aplicável a qualquer uma dessas estratégias.

[1] Assume-se que o sistema de detecção por anomalia pode adotar uma estratégia de monitoramento diferente da usada pelo sistema de detecção por mau uso.

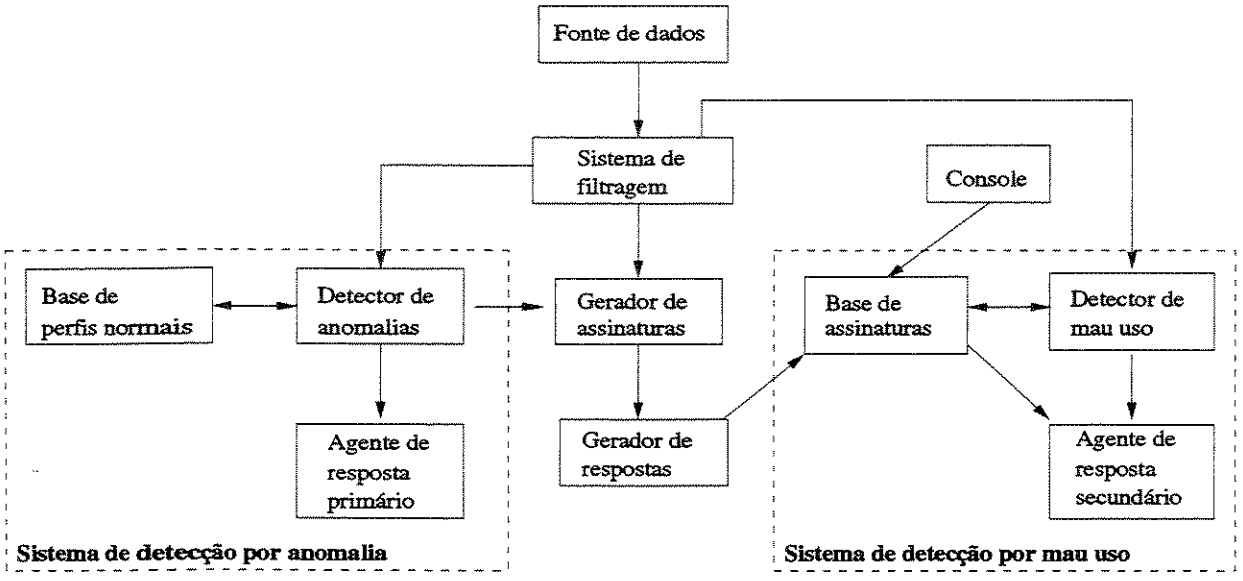


Figura 4.1: Modelo de IDS híbrido baseado no sistema imunológico humano.

Sistema de filtragem

O sistema de filtragem realiza o pré-processamento das informações (*audit reduction*) com o objetivo de identificar e remover informações reduntantes ou irrelevantes. Depois de filtradas, o fluxo de informações é passado aos sistemas de detecção e, quando requisitado, ao gerador de assinaturas.

Base de perfis normais

A base de perfis normais é responsável pelo armazenamento dos perfis que descrevem o comportamento normal do sistema monitorado. Esses perfis são definidos de acordo com a estratégia de monitoramento e a técnica de análise adotadas pelo detector de anomalias, sendo periodica e automaticamente atualizados para se adaptarem a mudanças legítimas no comportamento normal do sistema.

Detector de anomalias

O detector de anomalias recebe o fluxo de eventos do sistema de filtragem e verifica se ele representa algum comportamento anômalo, através da comparação das informações recebidas com o conjunto de perfis armazenados na base de perfis normais. Se algum sinal de comportamento anômalo é identificado, o detector de anomalias ativa o agente de resposta primário e repassa, para o gerador de assinaturas, as informações classificadas

como anormais.

Agente de resposta primário

Uma vez ativado, o agente de resposta primário inicia uma série de medidas de contenção para atrasar ou mesmo bloquear o suposto ataque. A reação do agente de resposta primário é limitada e genérica, uma vez que o ataque ainda não está especificamente identificado. O propósito principal dessas medidas de contenção primárias é a minimização dos efeitos da suposta intrusão, até que uma resposta específica e eficiente possa ser executada. Alguns exemplos de tais respostas primárias incluem a redução do nível de prioridade ou bloqueio de processos, a desabilitação de *logins* remotos, a proteção do sistema de arquivos e o disparo de alarmes.

Gerador de assinaturas

Uma característica inovadora do modelo de IDS imunológico é a conversão de informações consideradas anômalas em uma assinatura que especificamente identifique o ataque relacionado ao comportamento anormal. Essa conversão introduz uma capacidade de adaptação ao sistema de detecção por mau uso, provendo uma detecção futura mais precisa e eficiente do ataque. O gerador de assinaturas é responsável pela geração automatizada de assinaturas de ataques desconhecidos ao IDS. Depois de desenvolvida a assinatura, o gerador de assinaturas ativa o gerador de respostas.

Gerador de respostas

O gerador de respostas recebe a assinatura do ataque e elabora um conjunto de contra-medidas específicas para ele. Tanto a assinatura quanto a resposta relacionada são armazenadas na base de assinaturas.

Base de assinaturas

A base de assinaturas é responsável pelo armazenamento das assinaturas de ataques conhecidos, relacionando-os com as respectivas medidas de contenção específicas. As assinaturas são usadas pelo detector de mau uso, enquanto as contra-medidas são consultadas pelo agente de resposta secundário. A introdução de novas assinaturas e contra-medidas na base de assinaturas pode ser feita de duas formas: automaticamente pelo gerador de respostas ou manualmente pelo administrador do sistema.

Detector de mau uso

O detector de mau uso recebe o fluxo de eventos do sistema de filtragem e busca pelos padrões armazenados na base de assinaturas. Se algum padrão é encontrado no fluxo de

eventos, o detector de mau uso ativa o agente de resposta secundário.

Agente de resposta secundário

Uma vez ativado, o agente de resposta secundário recebe o padrão de mau uso detectado e busca na base de assinaturas pelas contra-medidas específicas relacionadas ao padrão. Então, o agente de resposta secundário executa as medidas de contenção.

Console

A interação entre o IDS e o administrador do sistema é possível através do console. Essa interface permite, por exemplo, a inclusão e remoção de assinaturas e contra-medidas na base de assinaturas.

4.2 Analogias entre o sistema de defesa humano e o modelo de IDS imunológico

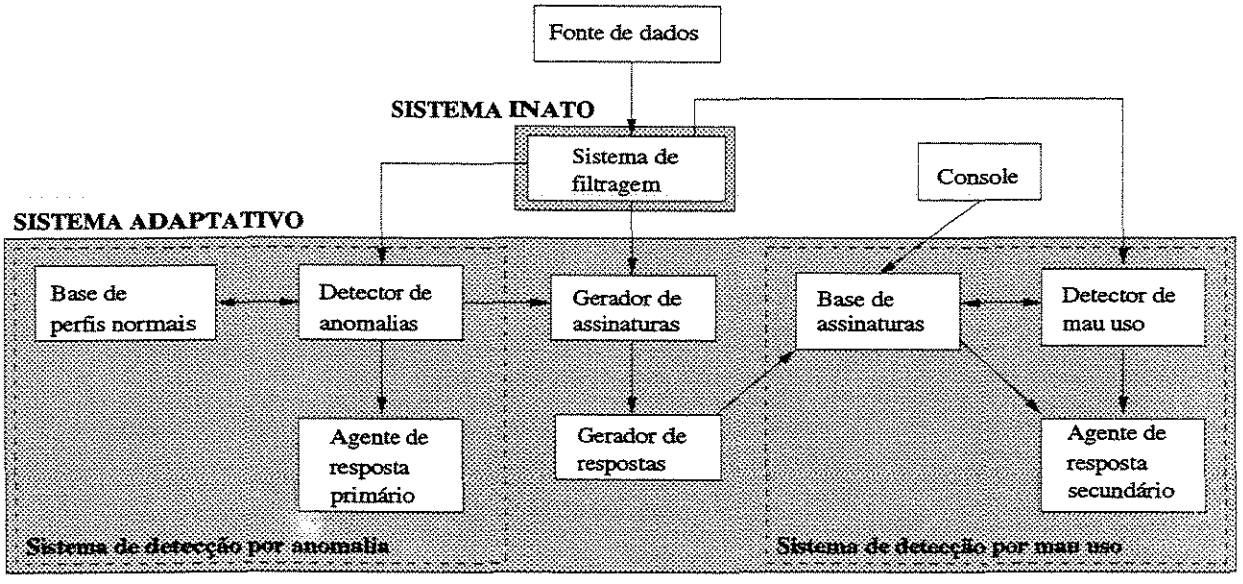


Figura 4.2: Analogia entre os sistemas inato e adaptativo e o modelo de IDS imunológico.

Como descrito no Capítulo 3, o sistema imunológico humano é composto pelos sistemas inato e adaptativo. Uma analogia entre esses sistemas e o modelo de IDS imunológico é ilustrada na Figura 4.2. O sistema inato é parcialmente representado no modelo pelo sistema de filtragem, cuja função remete ao processo de apresentação de antígenos.

Outras características principais do sistema inato, incluindo a detecção e a resposta não específicas, estão presentes no sistema de detecção por anomalia (respectivamente no detector de anomalias e no agente de resposta primário). Embora o sistema de detecção por anomalia possua características do sistema inato, ele é modelado como parte do sistema adaptativo, considerando a característica adaptativa da detecção por anomalia (relacionada principalmente à evolução dos perfis de comportamento normal).

O sistema adaptativo, por sua vez, é representado pelos componentes do modelo que implementam algum tipo de aprendizado e memória. Além disso, outras características importantes do sistema adaptativo, incluindo a detecção e a resposta específicas e eficientes, estão presentes no sistema de detecção por mau uso.

Outras analogias são apresentadas na Tabela 4.1, relacionando cada componente do modelo de IDS imunológico com as características do sistema de defesa humano.

Componentes do modelo	Sistema imunológico humano
Fonte de dados	Proteínas do corpo
Sistema de filtragem	Processo de apresentação de antígenos
Base de perfis normais	Proteínas *self*
Detector de anomalias	Detecção não específica dos fagócitos e receptores de células T
Agente de resposta primário	Resposta genérica do sistema inato
Gerador de assinaturas	Produção de células de memória e processo de maturação de afinidade
Gerador de respostas	Produção de anticorpos específicos
Base de assinaturas	Conjunto de células de memória com alta afinidade (memória imunológica)
Detector de mau uso	Detecção específica através das células com alta afinidade
Agente de resposta secundário	Resposta imunológica específica
Console	Imunidade adquirida artificialmente através de vacinas e soros

Tabela 4.1: Analogias entre os componentes do modelo de IDS imunológico e o sistema de defesa humano.

4.3 ADenoIdS: Uma implementação do modelo de IDS imunológico

Uma implementação do modelo de IDS imunológico é introduzida por Fabrício de Paula[2] em [75]. O IDS proposto [75], denomindado de ADenoIdS[3], é uma aplicação específica do modelo de IDS imunológico para proteção de um sistema computacional contra ataques de *buffer overflow*, que são considerados o problema de segurança mais importante e persistente [12, 96].

No contexto do sistema ADenoIdS assume-se que os atacantes não possuem acesso físico à máquina alvo, de modo que os ataques precisam ser efetuados de maneira remota. Para proteger uma máquina contra ataques de *buffer overflow*, o sistema ADenoIdS adota duas estratégias de monitoramento: baseada em uma máquina (empregada pelo sistema de detecção por anomalia) e baseada em rede (utilizada pelo sistema de detecção por mau uso). O detector de anomalias executa detecção no nível de chamada de sistemas, enquanto o detector de mau uso analisa o tráfego de rede destinado à máquina monitorada. Desse modo, cada máquina deve possuir um sistema ADenoIdS para estar protegido.

A implementação do sistema ADenoIdS, conduzida por Fabrício de Paula como parte de sua tese de doutorado, é baseada na plataforma Linux e consiste de *patches* para o *kernel* do sistema operacional e de um conjunto de ferramentas. Até o momento da escrita desta dissertação, o sistema ADenoIdS encontrava-se em fase de prototipação[4]. O funcionamento dos componentes do sistema ADenoIdS é descrito como segue.

Fonte de dados

A fonte de dados é responsável pela coleta de dois tipos de informações: chamadas de sistema dos processos monitorados e tráfego de rede destinado à máquina protegida. Além disso, a fonte de dados deve ser capaz de armazenar temporariamente as informações coletadas em um arquivo de *log*, permitindo a consulta por parte dos outros componentes do sistema.

Sistema de filtragem

O sistema de filtragem provê a seleção das chamadas de sistema feitas por um determinado processo, especificado pelo detector de anomalias ou pelo gerador de assinaturas,

[2]Como mencionado no Capítulo 1, Fabrício de Paula é um dos integrantes do grupo de imunologia computacional do LAS-IC-UNICAMP.

[3]No sistema imunológico humano, adenóides (*adenoids*) são nódulos linfáticos especializados, contendo células imunológicas que protegem o corpo humano contra invasores do sistema respiratório. O termo ADenoIdS é usado como acrônimo para *Acquired Defense System Based on the Immune System*.

[4]Maiores informações sobre o estágio de desenvolvimento do sistema ADenoIdS podem ser encontradas na URL http://www.ic.unicamp.br/~fabricio (disponível em janeiro de 2003).

além da filtragem do tráfego de rede, de acordo com as regras de filtragem determinadas pelo detector de mau uso ou pelo gerado de assinaturas. O sistema de filtragem pode ainda converter as informações recebidas pela fonte de dados para um formato canônico.

Sistema de detecção por anomalia

Uma abordagem para a detecção de ataques de *buffer overflow* consiste na análise das chamadas de sistema executadas por cada processo de interesse. A implementação do sistema de detecção por anomalia é baseada no sistema pH [97] (descrito na Seção 3.3.1.2) e seus componentes são apresentados como segue:

- **Base de perfis normais**: A base de perfis normais é responsável por armazenar, para cada processo monitorado, as sequências de chamadas de sistema que descrevem o comportamento normal de cada processo. Essas sequências são obtidas através da análise da execução dos processos, sob condições normais. Uma sequência de chamadas de sistema não prevista no perfil de um processo é interpretada como um possível sinal de ataque;

- **Detector de anomalias**: O papel do detector de anomalias é verificar se as chamadas de sistema executadas pelos processos monitorados representam um comportamento anormal. Se algum sinal de comportamento anômalo é detectado, o detector de anomalias ativa o agente de resposta primário e fornece uma série de informações ao gerador de assinaturas, incluindo a sequência de chamadas de sistema detectada como anômala;

- **Agente de resposta primário**: Uma vez ativado, o agente de resposta primário executa uma série de medidas de contenção sobre o processo suspeito, até que o IDS ou o administrador do sistema possa tomar uma decisão efetiva. Algumas medidas de contenção primárias incluem a inserção de atrasos na execução do processo, o bloqueio temporário de sua execução ou o bloqueio do tráfego de rede relacionado ao processo [97];

Gerador de assinaturas

O gerador de assinaturas executa um tipo de forense computacional automatizada (maiores informações são apresentadas no Capítulo 6). Baseado nas informações recebidas pelo detector de anomalias, o gerador de assinaturas inicia uma busca no tráfego de rede capturado (através de requisições ao sistema de filtragem), relacionado ao processo suspeito, com o intuito de identificar traços de código executável que possam ter gerado o comportamento anormal detectado.

Uma vez identificado o tráfego de rede que produziu o comportamento anômalo, o gerador de assinaturas pode obter diversas informações, incluindo a origem do ataque, o conteúdo malicioso dos pacotes e outras características específicas do ataque. Com base nessas informações, uma assinatura é criada e enviada ao gerador de respostas. Essa assinatura é definida no nível de rede[5], permitindo sua utilização pelo sistema de detecção por mau uso.

Se o gerador de assinaturas não pode encontrar sinais de ataque no tráfego de rede analisado, o administrador do sistema pode ser notificado e o provável ataque pode ser descartado. Desse modo, essa análise forense automatizada pode contribuir para diminuir a taxa de detecções falso-positivas emitidas pelo detector de anomalias.

Gerador de respostas

O gerador de respostas recebe a assinatura do ataque e elabora um conjunto de contra-medidas que responde de maneira específica ao tráfego de rede suspeito. No final, a assinatura do ataque e o conjunto de contra-medidas são inseridos na base de assinaturas, permitindo que o IDS adquira imunidade contra o ataque. Além disso, algum mecanismo de distribuição de informações pode se utilizado para disseminar, entre os outros IDSs da rede, o conhecimento adquirido.

Sistema de detecção por mau uso

O sistema de detecção por mau uso utiliza as informações contidas na base de assinaturas para monitorar o tráfego de rede destinado à máquina protegida. Dessa forma, o sistema ADenoIdS pode detectar e responder, no nível de rede, a ataques conhecidos, antes que o código malicioso chegue ao nível de aplicação. Os componentes do sistema de detecção por mau uso são apresentados como segue:

- **Base de assinaturas**: A base de assinaturas é responsável pelo armazenamento das assinaturas de ataques conhecidos e de suas respectivas contra-medidas. Cada assinatura representa um tráfego de rede considerado malicioso e o conjunto de contra-medidas relacionado indica as ações que devem ser tomadas quando esse tráfego é detectado na rede;

- **Detector de mau uso**: A detecção específica é feita pelo detector de mau uso, que analisa o tráfego de rede destinado à máquina protegida em busca dos padrões armazenados na base de assinaturas. Se algum padrão de ataque é detectado, o detector de mau uso ativa o agente de resposta secundário;

[5]Uma abordagem possivelmente adotada no protótipo inicial do sistema ADenoIdS é a geração de assinaturas no formato do sistema de detecção de intrusão Snort. Maiores informações sobre o IDS Snort podem ser encontradas na URL http://www.snort.org (disponível em janeiro de 2003).

- **Agente de resposta secundário:** Uma vez ativado, o agente de resposta secundário executa as medidas específicas relacionadas ao ataque detectado. Essa resposta ocorre principalmente no nível de rede;

Console

O console permite a interação do sistema ADenoIdS com o administrador, possibilitando, entre outras atividades, a manipulação da base de assinaturas de ataques.

4.4 O papel da forense computacional no modelo de IDS imunológico

Como visto no Capítulo 3, o sistema imunológico humano possui a capacidade de evoluir um conjunto de células de defesa, com o intuito de aumentar sua afinidade a um determinado agente patogênico. Esse exército altamente específico, produzido através do processo de maturação de afinidade, representa o conhecimento que o sistema imunológico humano possui acerca dos invasores com os quais já teve contato. Desse modo, as informações codificadas nas células adaptadas permitem uma reação mais eficiente contra novas exposições aos invasores conhecidos.

Essa característica adaptativa é introduzida no modelo de IDS imunológico através da análise forense dos ataques desconhecidos identificados pelo sistema de detecção por anomalia. As informações consideradas anômalas, juntamente com as evidências deixadas pelo suposto ataque (seja ele consumado, em andamento ou bloqueado), são minuciosamente analisadas buscando a geração de uma assinatura que especificamente identifique o ataque relacionado ao comportamento anormal detectado. Essa assinatura gerada é então acrescentada à base de assinaturas do sistema de detecção por mau uso, permitindo a detecção direta e eficiente de novas ocorrências do comportamento intrusivo.

Como visto na Seção 4.1, o gerador de assinaturas é o componente do modelo de IDS imunológico responsável pela análise minuciosa das evidências de um ataque e consequente geração de uma assinatura que caracterize o comportamento intrusivo. Sua funcionalidade permite que o modelo de IDS imunológico possa, de maneira análoga ao sistema de defesa humano, autonomamente alterar sua base de assinaturas de mau uso. Desse modo, em resumo, o papel da forense computacional é o de fornecer os conceitos e técnicas necessários para o funcionamento do gerador de assinaturas.

Uma revisão detalhada dos conceitos e técnicas empregados na área da forense computacional é apresentada no capítulo seguinte e, de modo a permitir o funcionamento automatizado do gerador de assinaturas, a questão da automatização do processo de análise forense é abordada no Capítulo 6.

4.5 Conclusão

Este capítulo apresentou uma nova abordagem para a aplicação da analogia com o sistema imunológico humano no desenvolvimento de um sistema de segurança computacional. Tal analogia considera o fato de que o sistema imunológico possui várias características de detecção por mau uso, em adição às características de detecção por anomalia exploradas nas pesquisas anteriores. Tais características de detecção por mau uso incluem, por exemplo, o reconhecimento de padrões específicos de antígenos, a memória de patógenos conhecidos e a detecção mais eficiente e de alta afinidade.

A partir da nova abordagem proposta, foi apresentado um modelo conceitual de IDS imunológico, desenvolvido como parte desta pesquisa, capaz de detectar e caracterizar um ataque, elaborar um plano de resposta especializado e efetuar o contra-ataque. Além disso, o modelo agrega as características de aprendizado e adaptação do sistema imunológico humano, podendo reagir a ataques desconhecidos e melhorar sua reação a exposições subsequentes de um mesmo ataque.

Para o funcionamento dessas características de adaptação e especialização foi proposta a utilização de conceitos e técnicas de forense computacional, com o intuito de gerar uma assinatura que especificamente identifique um ataque desconhecido, relacionado a algum comportamento anormal. Essa assinatura gerada pode ser utilizada na detecção direta e eficiente de novas ocorrências do comportamento intrusivo. Entretanto, essa geração de assinaturas de ataques deve ser um processo automatizado, permitindo sua aplicação em um sistema autônomo de segurança computacional. Para tal é necessário um estudo pioneiro acerca da automatização do processo de análise forense.

Capítulo 5

Forense computacional

Com o intuito de viabilizar a capacidade de geração automatizada de assinaturas de ataques do modelo de IDS imunológico, este capítulo discute os conceitos e técnicas aplicados na área da forense computacional. Embora o enfoque principal seja dado à análise de invasões em sistemas computacionais, muitos dos conceitos e técnicas apresentados neste capítulo podem ser estendidos à investigação forense de outros tipos de crimes envolvendo o uso de computadores.

Este capítulo é baseado na publicação [80] e seu objetivo principal é fornecer uma descrição detalhada sobre onde, como e o quê procurar em um sistema computacional invadido. Para tal, é apresentada, na Seção 5.1, uma introdução à área da forense computacional e, na Seção 5.2, são discutidos brevemente os objetivos e métodos de operação dos invasores. Na Seção 5.3 são apresentadas as principais fontes de informação de um sistema computacional e, na Seção 5.4, são abordadas algumas questões sobre o correlacionamento de evidências. Por fim, a estrutura geral do processo de investigação forense é discutida na Seção 5.5.

5.1 O que é forense computacional?

As últimas décadas foram marcadas pela integração dos computadores no modo de vida das pessoas. Infraestruturas básicas da sociedade, como redes financeiras, sistemas de comunicação, estações de energia e sistemas de saúde, dependem todas de sistemas computacionais para seu funcionamento eficiente e confiável. Além disso, é crescente o número de indivíduos que utilizam computadores pessoais por conveniência, educação e entretenimento. A conectividade oferecida pela Internet também introduziu uma série de novas facilidades no dia-a-dia das pessoas, como o correio eletrônico e a *World Wide Web*, permitindo o acesso anônimo a quase todo tipo de informação ou sistema.

Não é de se espantar o fato de que essa revolução computacional tenha atingido

também o mundo do crime. A tecnologia dos computadores está envolvida em um número crescente de atividades ilícitas. Além de serem utilizados como ferramentas para a consumação de alguns tipos de delitos (como, por exemplo, invasão de sistemas, disseminação de pornografia infantil e fraude), os computadores podem conter evidências relacionadas com qualquer tipo de atividade ilícita, incluindo homicídio e estupro.

O aumento dramático em crimes relacionados com computadores requer um maior entendimento de como se obter e utilizar evidências eletrônicas armazenadas em sistemas computacionais. Tal entendimento pode ser alcançado através dos conceitos e metodologias da forense computacional.

A forense computacional compreende a aquisição, preservação, identificação, extração, restauração, análise e documentação de evidências computacionais, quer sejam componentes físicos ou dados que foram processados eletronicamente e armazenados em mídias computacionais [47, 73].

O propósito do exame forense é a procura e extração de evidências relacionadas com o caso investigado, que permitam a formulação de conclusões acerca da infração [26]. Existem duas abordagens no que diz respeito ao objetivo final da análise forense. Na primeira, a análise forense busca obter informação de valor probante (coerente com as regras e leis das evidências e admissível em uma corte de justiça) a ser utilizada em um processo criminal. Na segunda abordagem, o exame é realizado dentro de uma corporação com o objetivo de determinar a causa de um incidente e assegurar que o mesmo não ocorra novamente, sem que haja preocupação com formalidades legais.

Mesmo que não haja intenção de se instituir um processo criminal, toda investigação deve considerar como prática padrão a utilização de metodologias e protocolos que garantam sua possível aceitação em uma corte de justiça [47]. Tratar todo caso com a formalidade de um processo criminal ajuda a desenvolver bons hábitos de investigação.

Nesse sentido, existem alguns aspectos chave que constituem as etapas do processo de análise forense de um sistema computacional [9]:

- coleta de informações (ou *information gathering*);
- reconhecimento das evidências;
- coleta, restauração, documentação e preservação das evidências encontradas;
- correlação das evidências;
- reconstrução dos eventos;

Toda informação relevante deve ser coletada para análise e, conforme as evidências digitais são encontradas, elas devem ser extraídas, restauradas quando necessário (evidências

danificadas ou cifradas, por exemplo), documentadas e devidamente preservadas. Em seguida, as evidências encontradas podem ser correlacionadas, permitindo a reconstrução dos eventos relacionados ao incidente investigado. Muitas vezes a análise das evidências (etapas de correlação e reconstrução dos eventos) resulta na descoberta de novas informações, formando um ciclo no processo de análise forense[1] [9].

A forense computacional é uma área de pesquisa relativamente recente, entretanto, é crescente a necessidade de desenvolvimento nesse campo, uma vez que a utilização de computadores em atividades criminosas tem se tornado uma prática comum. Os computadores atingiram crimes tradicionais, como extorsão, roubo e tráfico de drogas, e também originaram uma nova classe de crimes, conhecidos como "crimes da Internet", incluindo ataques de negação de serviço, invasão de sistemas e disseminação de vírus de computador. Com o advento da Internet, ataques remotos a sistemas computacionais tornaram-se mais comuns, tirando vantagem da crescente complexidade e vulnerabilidade dos serviços de rede [47]. O aumento na sofisticação e frequência com que esses ataques têm ocorrido representa um desafio crescente para os encarregados de investigar e responder a esses incidentes.

5.2 *Modus operandi* do invasor

Novas formas de penetrar e interferir no funcionamento de sistemas computacionais são desenvolvidas a cada dia. Com o mínimo de conhecimento de redes de computadores, praticamente qualquer um pode obter gratuitamente na Internet e utilizar ferramentas para invadir um sistema computacional e provocar todo tipo de estrago. O número de ataques externos a uma organização tem crescido consideravelmente, equiparando-se à quantidade de ataques cometidos por indivíduos de dentro da própria organização [9]. A intrusão de sistemas tem sido considerada um risco à segurança nacional em muitos países, de modo que a compreensão acerca dos objetivos e métodos empregados nesses incidentes tem se tornado alvo de muitos estudos [9, 47].

Entender a motivação e o comportamento de um intruso é um ponto chave para orientar a investigação, pois essa compreensão fornece pistas sobre onde e o quê procurar durante a análise forense [9]. Quanto maior a consciência acerca dos objetivos e *modus operandi* de um atacante, maior o preparo do investigador para analisar e responder a um incidente [47].

A invasão de sistemas computacionais ocorre com finalidades diversas, podendo ser destacadas as seguintes:

[1] Devido a sua importância, a estrutura geral do processo de análise forense é discutida detalhadamente na Seção 5.5.

- obtenção de informações (roubo de segredos, números de cartões de crédito, senhas e outros dados relevantes ao intruso);
- promover algum estrago ("pichação" de *sites*, destruição de informações e paralisação do sistema, por exemplo);
- utilização dos recursos do sistema (como repositório de dados, para disseminação de ataques distribuídos ou para provimento de algum serviço, por exemplo);

Dependendo da finalidade e da habilidade, o *modus operandi* de um invasor pode sofrer algumas variações. Entretanto, os passos tomados pelo atacante para comprometer um sistema computacional podem ser generalizados como segue [9, 47]:

- identificação do alvo;
- busca de vulnerabilidades no alvo;
- comprometimento inicial;
- aumento de privilégio;
- "invisibilidade";
- exploração do sistema;
- instalação de *back doors*;
- limpeza dos rastros;
- retorno por uma *back door*, inventário e comprometimento de máquinas vizinhas;

A primeira atitude do atacante é a escolha de um alvo em potencial. Após a localização, o atacante começa a reunir informações sobre o sistema alvo a fim de identificar vulnerabilidades no sistema operacional ou serviços de rede disponíveis. Se o invasor ainda não possui uma combinação de usuário e senha válida para o sistema alvo, ele utiliza métodos como *sniffing*, adivinhação de senhas, engenharia social ou *scanning* para encontrar um ponto de entrada.

Uma vez encontrado um ponto de entrada (conta de usuário ou vulnerabilidade em um serviço, por exemplo) o invasor realiza o comprometimento inicial do sistema. Essa primeira intrusão geralmente provoca muito "barulho", especialmente se o sistema alvo estiver devidamente guarnecido, e costuma ocorrer quando ninguém está presente para "ouvir" [47]. Tentativas de adivinhar senhas criam um número incomum de registros de falhas de *login*; o comprometimento de aplicativos, através de ataques de *buffer overflow*,

geralmente ficam registrados nos arquivos de *log* ou geram *core files*; e as várias tentativas de se invadir o sistema podem gerar mensagens de advertência [47].

Depois que o atacante ganha acesso ao sistema, ele busca por privilégios irrestritos (conta de *root*) — assumindo que o comprometimento inicial já não lhe forneceu acesso à conta de *root*. O invasor transfere para o sistema, uma série de programas maliciosos, conhecidos como *exploits*, e tenta explorar vulnerabilidades que possam fornecer o acesso à conta de *root*. Com acesso ilimitado, o atacante procura remover traços de sua presença, tornando-se "invisível", através da instalação de *rootkits* e *trojan horses*.

Quando o invasor obtém privilégios irrestritos e garante sua "invisibilidade", ele executa uma verdadeira varredura no sistema [47]. O atacante procura saber o quanto sua presença perturba o sistema invadido e, por conseguinte, pode ser descoberta (analisando, por exemplo, as configurações do sistema de *log*). Em seguida, ele investiga as medidas de segurança implementadas no sistema invadido — em alguns casos, o atacante até corrige vulnerabilidades existentes, para impedir que outro invasor faça uso do sistema.

Após compreender as configurações do sistema, o atacante instala *back doors* para facilitar seu retorno e tenta apagar os rastros deixados por sua presença no sistema. Utilizando uma *back door*, o invasor retorna de maneira mais discreta que o comprometimento inicial e faz um inventário acerca das informações existentes na máquina invadida e dos potenciais alvos da vizinhança.

Nível de habilidade	Habilidades	Evidências
Clueless	Nenhuma habilidade	Todas as atividades são bastante aparentes
Script Kiddie	Capaz de encontrar *exploits* prontos na Internet e executá-los seguindo instruções detalhadas. Não escrevem programas	Pode tentar cobrir rastros com o uso de *rootkits* prontos, mas com sucesso limitado. Pode ser detectado com esforço mínimo
Guru	Equivalente a um administrador experiente. Hábil em programação. Checa a existência de programas de segurança e esquemas de *log* seguros, evitando alvos protegidos	Cuidadosamente apaga evidências em arquivos de *log*. Não deixa traços óbvios de sua presença. Pode instalar *trojan horses* e *back doors* para um acesso futuro
Wizard	Possui um grande conhecimento do funcionamento interno de um sistema. Capaz de manipular *hardware* e *software*	Praticamente não deixa evidências úteis. Pode comprometer totalmente o sistema

Tabela 5.1: Relação entre a habilidade do invasor e a quantidade de evidências deixadas.

A habilidade do invasor em executar o *modus operandi* descrito anteriormente pode ser fundamental para o processo de análise forense, pois a quantidade de evidências deixadas depende diretamente do nível de conhecimento do atacante [47]. Para ilustrar essa relação, é possível classificar a habilidade do invasor em quatro classes, de acordo com [47]: *Clueless*, *Script Kiddie*, *Guru* e *Wizard*. Nesse sentido, a Tabela 5.1 apresenta a relação entre essas quatro classes e a quantidade de evidências deixadas no sistema invadido.

5.3 Evidências digitais

Um dos princípios fundamentais da forense é o Princípio da Troca de Locard [9]. De acordo com esse princípio, qualquer um, ou qualquer coisa, que entra em um local de crime leva consigo algo do local e deixa alguma coisa para trás quando parte [9]. No mundo virtual dos computadores, o Princípio da Troca de Locard ainda é válido (ou pelo menos parte dele): onde quer que o intruso vá ele deixa rastros [100]. Tais rastros podem ser extremamente difíceis ou praticamente impossíveis de serem identificados e seguidos, mas eles existem [100]. Nesses casos o processo de análise forense pode tornar-se extremamente complexo e demorado, necessitando do desenvolvimento de novas tecnologias para a procura de evidências.

O termo evidência digital refere-se a toda e qualquer informação digital que pode ser capaz de determinar que uma intrusão ocorreu ou que provê alguma ligação entre a intrusão e as vítimas ou entre a intrusão e o atacante[2].

A evidência digital não deixa de ser um tipo de evidência física, embora seja menos tangível [9]. Ela é composta de campos magnéticos e pulsos eletrônicos que podem ser coletados e analisados através de técnicas e ferramentas apropriadas. Entretanto, a evidência digital possui algumas características próprias:

- ela pode ser duplicada com exatidão, permitindo a preservação da evidência original durante a análise;
- com os métodos apropriados é relativamente fácil determinar se uma evidência digital foi modificada;
- por outro lado, a evidência digital é extremamente volátil, podendo ser facilmente alterada durante o processo de análise;

A busca de evidências em um sistema computacional constitui-se de uma varredura minuciosa nas informações que nele residam, sejam dados em arquivos ou em memória, "deletados" ou não, cifrados ou possivelmente danificados.

[2]Essa definição foi adaptada para o contexto de intrusão de sistemas a partir da definição apresentada em [9].

Dan Farmer e Wietse Venema introduziram um conceito denominado de ordem de volatilidade [27]. Tal conceito determina que o tempo de vida de uma evidência digital varia de acordo como o local onde ela está armazenada, de modo que a extração das informações contidas em um sistema computacional deve ser orientada no sentido decrescente de volatilidade, isto é, das informações mais voláteis até as menos voláteis. Desse modo, as principais fontes de informação de um sistema computacional são apresentadas, na ordem descendente de volatilidade, como segue [7, 27, 47]:

- dispositivos de armazenagem da CPU (registradores e *caches*);
- memória de periféricos (memória de vídeo, por exemplo);
- memória principal do sistema;
- tráfego de rede;
- estado do sistema operacional (como, por exemplo, estado das conexões de rede e dos processos em execução, usuários logados e tabelas e módulos do sistema);
- dispositivos de armazenagem secundária;

Quanto maior a volatilidade de uma informação, mais difícil se torna sua extração e menos tempo há para executá-la. O simples ato de observar informações altamente voláteis pode alterá-las, de modo que é pouco provável que alguém possa, por exemplo, utilizar o conteúdo dos registradores da CPU [47]. Entretanto, informações voláteis como o conteúdo da memória principal do sistema, o tráfego de rede e o estado do sistema operacional podem ser capturadas com relativa facilidade e podem conter pistas valiosas a respeito de intrusões em andamento.

As principais fontes de informação de um sistema computacional são discutidas brevemente na sequência. Alguns detalhes particulares sobre a extração e análise dos dados em cada uma são apresentados no Apêndice A.

5.3.1 Dispositivos de armazenagem da CPU

As informações contidas nos registradores da CPU são de mínima utilidade e sua captura é impraticável [47]. As *caches* podem conter informações que ainda não foram atualizadas na mémoria principal do sistema, entretanto sua captura também pode ser impraticável [47].

5.3.2 Memória de periféricos

Muitos dispositivos como modems, *pagers*, aparelhos de fax e impressoras, contêm memórias que podem ser acessadas e salvas [46]. Nelas podem estar armazenadas informações que não mais residem no sistema analisado, como documentos e mensagens de texto ou números de fax e telefone. A memória de vídeo também pode prover informações úteis no caso do invasor estar utilizando um console ou terminal gráfico, de modo que a tela corrente pode ser capturada e reproduzida [47], como detalhado na Seção A.1.

5.3.3 Memória principal do sistema

A memória principal contém todo tipo de informação volátil, como, por exemplo, informações dos processos que estão em execução, dados que estão sendo manipulados e muitas vezes ainda não foram salvos no disco e informações do sistema operacional [92, 99]. Tais dados podem ser capturados e analisados, como detalhado na Seção A.2, por meio de *dumps* da memória[3], pela geração de *core files* ou pela interface provida pelo diretório `/proc` [47].

Os arquivos denominados *core files* (ou *core dumps*) são gerados pelo sistema operacional quando certos sinais (SIGQUIT ou SIGSYS, por exemplo) são enviados a um processo em execução [68]. Um *core file* é uma imagem, em um formato especial, da memória utilizada pelo processo no momento em que o sinal foi recebido[4] [34].

Ataques de *buffer overflow* podem causar a geração de *core files*, permitindo a identificação do processo explorado e de características relacionadas à vulnerabilidade. Além disso, alguns *exploits* usados por atacantes geram propositalmente *core dumps* de programas que manipulam senhas, de modo que tais senhas podem ser resgatadas do arquivo gerado [47, 100]. A análise de um *core file* pode revelar, entre outras informações, as rotinas que estavam sendo executadas, os valores dos registradores e o conteúdo do espaço de endereçamento virtual do processo [34, 68]. Entre essas informações podem ser encontradas algumas evidências como, por exemplo:

- o programa relacionado ao *core file* é suspeito, podendo ser, por exemplo, um programa simplesmente desconhecido; um comando conhecido, mas que não deveria ter sido terminado; um programa com nome suspeito (por exemplo, que faz alusão a um código hostil, como *sniffer* ou *cracker*); ou ainda um programa que manipula algum tipo de senha;

[3] O processo de extração do conteúdo armazenado na memória principal do sistema é denominado de *dump* da memória.

[4] O pseudo-arquivo `/proc/kcore` fornece uma representação da memória física do sistema no formato de um *core file*, podendo ser examinado de maneira análoga.

- o sinal recebido pelo processo indica algum tipo de violação de memória, como resultado de um possível ataque de *buffer overflow*;

- o programa relacionado ao *core file* faz referência a arquivos suspeitos, incluindo, por exemplo, arquivos com nome ou localização suspeita, ou, ainda, arquivos que o programa não deveria estar acessando;

5.3.4 Tráfego de rede

A captura do tráfego de rede pode ser comparada à gravação em vídeo de um crime, sendo discutida em detalhes na Seção A.3. A partir dos pacotes capturados, é possível recontruir a comunicação entre o atacante e a máquina alvo, de modo que uma sequência de eventos pode ser estabelecida e comparada com as outras evidências encontradas na máquina invadida [10]. Entre as evidências mais comumente encontradas na análise do tráfego de rede podem ser citadas as seguintes:

- pacotes com endereço IP inválido ou suspeito (endereços reservados ou conhecidos de outros ataques, por exemplo);

- pacotes relacionados a portas suspeitas (como, por exemplo, servidores utilizando portas desconhecidas acima de 1024 ou pacotes destinados a um servidor que deveria estar desabilitado);

- tráfego não condizente com o padrão do protocolo[5] (pacotes com padrões inválidos de *flags*, opções, fragmentação e tamanho);

- tráfego TCP sem estabelecimento de conexão, sem *flags* ou com números de sequência inválidos (estáticos, aleatórios ou sobrepostos);

- tráfego intenso de pacotes incomuns à rede ou que deveriam estar desabilitados (ICMP *echo request* e *reply*, por exemplo);

- pacote ICMP *echo reply* sem correspondente *echo request* anterior;

- pacotes ICMP *echo request* e *reply* com número de sequência estático ou não incremental;

- pacotes contendo, na área de dados, comandos UNIX e códigos de NOPs;

[5]Informações sobre os protocolos de rede podem ser encontradas nas respectivas RFCs ou em [11, 101, 103].

- tráfego intenso de pacotes para um determinado serviço ou máquina (datagramas chegando com intervalo de tempo muito pequeno)[6];
- requisições HTTP suspeitas (URLs contendo referências a comandos UNIX ou a *scripts* suspeitos, por exemplo);
- tráfego característico de serviços como TELNET e FTP em portas incomuns;

5.3.5 Estado do sistema operacional

Informações relacionadas com o estado do sistema operacional podem fornecer pistas importantes quanto à existência, tipo e origem de um ataque em andamento [47]. Tais informações representam uma imagem do sistema operacional em um determinado instante e geralmente são perdidas quando o sistema é desligado. Entre as informações que descrevem o estado do sistema operacional incluem-se os processos em execução, as conexões de rede estabelecidas e os servidores aguardando conexão, o estado das interfaces de rede, os usuários "logados", as tabelas e *caches* mantidas e os módulos de *kernel* carregados.

Outras informações voláteis podem ser bastante úteis para reconstruir o estado do sistema, incluindo, por exemplo, as regras de filtragem em operação no *firewall*, os sistemas de arquivo montados e informações sobre serviços RPC. Através dessas informações é possível identificar vestígios que podem estar relacionados com a intrusão como, por exemplo, regras de filtragem inválidas ou ausentes, sistemas de arquivos montados indevidamente e serviços RPC ativados ou desativados impropriamente. Alguns aspectos do sistema operacional mais comumente analisados são discutidos na sequência.

5.3.5.1 Processos em execução

A coleta de informações acerca dos processos em execução no sistema analisado é de grande importância, pois tais informações podem revelar evidências de atividades não-autorizadas. Cada processo é executado em um ambiente com privilégios específicos que determinam quais recursos do sistema, programas e arquivos de dados podem ser acessados e de que modo [92]. Um invasor pode desvirtuar a execução de um programa ou serviço, causando sua falência, ou fazendo com que ele opere de maneira inesperada ao administrador ou usuário (acessando informações não-autorizadas ou consumindo recursos excessivos, por exemplo). Além disso, o atacante pode executar processos maliciosos como *rootkits*, *back doors* e *trojan horses*.

Existem várias formas de acessar as informações relacionadas aos processos em execução. Uma série de utilitários permitem, por exemplo, coletar uma lista de todos os

[6]Detalhes sobre ataques de negação de serviço podem ser encontrados em [34].

processos do sistema e visualizar detalhes sobre cada um, incluindo a data e hora de início do processo, o comando executado, os arquivos abertos e o consumo de recursos. Na Seção A.4 são apresentados alguns detalhes sobre as formas de acesso às informações dos processos em execução.

A partir das informações relacionadas aos processos em execução é possível identificar uma série de situações que podem representar evidências de uma intrusão como, por exemplo:

- existência de processos com nome suspeito (comandos não identificados ou conhecidamente maliciosos, por exemplo);
- ausência de processos que deveriam estar executando (principalmente relacionados a mecanismos de segurança);
- acesso a um arquivo suspeito ou não permitido;
- processos com *sockets* suspeitos relacionados (escutando em portas suspeitas, por exemplo);
- processos com horário de início suspeito ou executado por usuário incomum;
- processos com consumo de recursos incomum;
- existência de servidores executando fora do *super daemon* `xinetd` quando não deveriam (como `in.telnetd` e `in.fingerd`);
- existência de redirecionadores de porta, como o `fpipe` ou `datapipe`;
- existência de *unlinked files*[7] em `/proc`;
- inconsistências em `/proc`[8] como, por exemplo, os arquivos `exe` e `cmdline` indicando comandos diferentes;
- *file descriptors* suspeitos no sub-diretório `fd` em `/proc` (como, por exemplo, a saída padrão do processo `syslogd` direcionada para `/dev/null`);

[7]Alguns atacantes comumente iniciam a execução de um programa e "deletam-no" em seguida, com o intuito de esconder seus rastros no sistema. O arquivo "deletado"é denominado de *unlinked file* e só será apagado quando o processo relacionado terminar, podendo ser recuperado como detalhado na Seção A.4.4.

[8]O conteúdo do diretório `/proc` é detalhado na Seção A.4.4.

5.3.5.2 Conexões de rede

O estado da rede provê informações valiosas acerca das conexões de rede em andamento ou finalizadas e dos processos aguardando uma conexão, podendo ser acessadas como detalhado na Seção A.5. A partir dessas informações é possível determinar se o atacante instalou e ativou uma *back door* no sistema ou se existe alguma conexão não-autorizada (ou suspeita) em andamento. Os vestígios mais comumente encontrados em relação às conexões de rede incluem, por exemplo:

- endereços IP suspeitos (endereços conhecidos de outros ataques ou endereços incomuns a determinados serviços, por exemplo);
- serviços suspeitos aguardando conexão ou com conexões estabelecidas (portas incomuns e nomes suspeitos podem ser indicadores);
- serviços que deveriam estar desabilitados;
- ausência de servidores que deveriam estar em execução;
- *raw sockets*[9] suspeitos (*raw sockets* ICMP, por exemplo);

5.3.5.3 Estado das interfaces de rede

O estado das interfaces de rede pode revelar a existência de um possível *sniffer* no sistema analisado. Desse modo, é importante checar a presença de interfaces em modo promíscuo, como apresentado na Seção A.6.

5.3.5.4 Usuários logados

Determinar quais usuários estão logados no sistema é um processo bastante simples [67]. Além da identificação dos usuários logados, é possível obter, como detalhado na Seção A.7, informações sobre o endereço IP do sistema a partir do qual eles efetuaram o *login* (caso seja remoto), o horário do *login* e o comando que eles estão executando no sistema. A partir desses dados é possível identificar possíveis evidências como, por exemplo:

- nomes de usuários suspeitos (usuários que não deveriam ter acesso de *login* ou usuários desconhecidos, por exemplo);
- *logins* de origens suspeitas e/ou em horário suspeito;

[9] Um *raw socket* provê acesso direto a um protocolo de comunicação de baixo nível, permitindo tirar proveito de algumas características do protocolo não acessíveis pela interface normal [68].

- *logins* remotos não permitidos (dependendo da política de segurança, o *login* remoto para o usuário *root*, por exemplo, pode ser desabilitado);
- usuários executando comandos suspeitos ou não-autorizados;

5.3.5.5 Tabelas e *caches*

Embora raro, é possível que um atacante altere as tabelas de roteamento ou ainda desvirtue a resolução de endereços MAC [47]. Essas informações podem ser acessadas e verificadas como detalhado na Seção A.8.

Para analisar o conteúdo da tabela e *cache* de roteamento é necessário algum conhecimento sobre a topologia da rede invadida. Desse modo, é possível observar se existe algum tipo de inconsistência como, por exemplo:

- rotas alteradas ou incomuns;
- rotas que passam por redes distantes e/ou desconhecidas;
- rotas estáticas anormais;

Ocasionalmente, os atacantes falsificam endereços IP e MAC para transpor mecanismos de segurança como, por exemplo, listas de controle de acesso, regras de filtragem e configurações de *switches* [67]. Nesse sentido, a *cache* de resolução de endereços MAC pode ser últil no processo de análise forense, possibilitando determinar se algum tipo de falsificação foi utilizada. Um atacante pode ainda corromper o conteúdo da *cache* de DNS como parte de seu ataque [67], de modo que uma análise detalhada pode revelar informações valiosas.

5.3.5.6 Módulos de *kernel* carregados

Através dos chamados módulos de *kernel* carregáveis (LKMs — *Loadable Kernel Modules*), o comportamento do sistema operacional pode ser alterado sem a reinicialização o sistema. Módulos maliciosos, instalados pelos atacantes, podem comprometer totalmente o funcionamento do sistema operacional, interceptando comandos do sistema, como `netstat`, `ifconfig`, `ps` e `ls`, e produzindo resultados falsos [67]. Além da instalação de módulos de *kernel* maliciosos, alguns *rootkits* modificam os endereços na tabela de chamadas de sistema para que eles referenciem chamadas de sistema maliciosas, desvirtuando o funcionamento de determinados programas.

Os *rootkits* que instalam LKMs maliciosos representam um desafio aos investigadores, pois quando o atacante tem controle do sistema no nível do *kernel*, não é prudente confiar nas informações providas pelos programas executados no sistema comprometido, mesmo

que esses programas sejam confiáveis. Entretanto, pior do que descobrir a existência de um módulo malicioso no sistema analisado, é executar a coleta de informações sobre o estado do sistema operacional sem notar a presença do LKM do atacante. Por esse motivo, a investigação dos módulos de *kernel* é de grande importância para o processo de análise forense, uma vez que a própria identificação dos módulos maliciosos representa uma evidência de que a intrusão ocorreu. Essa investigação pode ser executada como apresentado na Seção A.9.

5.3.6 Dispositivos de armazenagem secundária

O disco representa a maior fonte de informação para o exame forense, pois além de ser uma memória não volátil, sua capacidade de armazenamento pode atingir terabytes de informações. Do ponto de vista da forense computacional, o disco é muito mais que um conjunto de partições e sistemas de arquivos, uma vez que cada porção do disco, por menor que seja, onde se possa ler e escrever um conjunto de bits[10], representa uma possível fonte de informação para a análise forense. Nesse sentido, determinadas áreas do disco não são acessíveis através de um sistema de arquivos e podem conter informações que o atacante mantém escondidas ou, ainda, vestígios que o mesmo tentou apagar.

A análise do disco inicia-se com a obtenção de informações sobre sua geometria e configuração, seguida pelo processo de geração da imagem bit-a-bit[11] do dispositivo, que é utilizada em substituição ao dispositivo original, evitando que este sofra qualquer tipo de alteração.

As informações armazenadas no disco podem ser divididas em duas grandes classes: aquelas acessíveis através de um sistema de arquivos (os arquivos propriamente ditos e seus atributos) e as informações armazenadas em áreas não acessíveis através do funcionamento normal de um sistema de arquivos (incluindo, por exemplo, os espaços não alocados aos arquivos e as áreas do disco que não contêm um sistema de arquivos). As diversas fontes de informação que constituem o disco são apresentadas na sequência, de acordo com essa classificação.

5.3.6.1 Áreas não acessíveis através de um sistema de arquivos

Como visto anteriormente, o disco pode ser utilizado sem as estruturas e facilidades de um sistema de arquivos. Desse modo, informações valiosas para o processo de análise forense podem estar armazenadas não somente nos arquivos, mas também em áreas do

[10] Através dos arquivos de dispositivo (*device files*) do UNIX ou diretamente pelo controlador do disco [87] é possível ler e escrever dados no dispositivo de armazenagem sem a abstração de arquivos, o que é chamado de *raw I/O* [92].

[11] Maiores detalhes sobre o procedimento de imagem são apresentados na Seção 5.5.4.

disco que não são acessíveis através do funcionamento normal de um sistema de arquivos. Tais áreas do disco incluem:

- os espaços não alocados dentro de um sistema de arquivos;
- os espaços alocados a arquivos, mas não totalmente utilizados, chamados de *file slacks*[12];
- e áreas do disco que não constituem uma partição ou que não contêm um sistema de arquivos;

Essas áreas do disco podem conter informações deliberadamente escondidas ou dados que foram "deletados"[13]. Quando um arquivo é "deletado", sua referência é removida das estruturas do sistema de arquivos, entretanto os dados contidos no arquivo não são apagados fisicamente do disco [9], permanecendo indefinidamente, até que sejam sobrescritos por outros arquivos. Desse modo, arquivos completos ou porções menores podem ser recuperados após terem sido "deletados" e parcialmente sobrescritos [9, 47].

Uma discussão detalhada sobre o acesso e recuperação de informações escondidas ou "deletadas", contidas nas áreas do disco não acessíveis através de um sistema de arquivos, é apresentada na Seção A.10.

5.3.6.2 Sistema de arquivos

Como um banco de dados, o sistema de arquivos é a parte do sistema operacional responsável por organizar as informações do disco na forma de arquivos e diretórios [92]. Para o processo de análise forense, o sistema de arquivos representa a fonte de informação onde o exame se concentra mais, podendo fornecer pistas valiosas a partir de suas estruturas internas e da análise dos arquivos e diretórios. Entretanto, é necessário inicialmente preparar o sistema de arquivos para o processo de análise forense, possibilitando o acesso controlado e confiável a suas informações, como apresentado em detalhes na Seção A.11.

Embora os dados contidos nas estruturas internas do sistema de arquivos, incluindo marcas de tempo e dados relativos a arquivos "deletados", possam representar informações valiosas, a análise dos arquivos e diretórios apresenta uma importância particular no processo de investigação forense, uma vez que eles contêm as principais informações do sistema computacional e de seus usuários. Uma discussão detalhada sobre os métodos de

[12]Os *file slacks* [92] ocorrem pelo fato do sistema de arquivos alocar blocos de tamanho fixo. Por exemplo, um arquivo com tamanho de 2460 bytes, em um sistema de arquivos com tamanho de bloco de 1024 bytes, ocupa três blocos de alocação, desperdiçando os últimos 612 bytes alocados.

[13]Os *file slacks* gerados nos sistemas de arquivos EXT2FS e FFS não contêm dados pertencentes a arquivos "deletados", pois cada novo bloco alocado a um arquivo é preenchido com zeros, sobrescrevendo as informações anteriores.

extração e análise das informações contidas nas estruturas internas do sistema de arquivos, incluindo a recuperação de dados "deletados", é apresentada na Seção A.12.

Existe uma alta probabilidade de muitos arquivos e diretórios conterem informações relacionadas com a intrusão [67]. Entretanto, o sucesso da análise forense em identificar todos os arquivos e diretórios relevantes é bem menos certo [67]. Felizmente, alguns tipos de arquivos e diretórios, resumidos na Tabela 5.2, comumente contêm ou representam evidências de uma intrusão, aumentando as chances de sucesso da investigação. Alguns desses arquivos e diretórios principais são detalhados nas Seções 5.3.7 e 5.3.8.

Tipo	Descrição
Arquivos de configuração	O sistema UNIX mantém certos arquivos de configuração que são comumente acessados e/ou alterados pelos atacantes. Em especial estão os *scripts* de inicialização, arquivos de informações sobre contas e senhas, configuração dos serviços de rede, tarefas agendadas, relações de confiança e do sistema de *log*
Filas	Incluem informações sobre processos em execução e atividades incompletas. Podem conter mensagens de correio eletrônico e documentos enviados para impressão que não existem em qualquer outro lugar do sistema
Caches	Arquivos de *cache* são usados para melhorar a performance do sistema ou de algum aplicativo. Podem conter informações valiosas como, por exemplo, um histórico de *sites Web* supostamente visitados pelo atacante (*sites* contendo informações e ferramentas para intrusões, por exemplo)

Tabela 5.2: Resumo dos principais arquivos e diretórios que comumente contêm ou representam evidências de uma intrusão (continua na próxima página).

Tipo	Descrição
Arquivos de *swap*	Quando a demanda do sistema excede a capacidade da memória, algumas informações são retiradas e armazenadas temporariamente nos arquivos de *swap*. Tais arquivos podem conter fragmentos de dados ou até mesmo um arquivo completo que nunca foi salvo no disco
Diretórios temporários	Diretórios temporários (`/tmp` e `/usr/tmp`) servem como diretórios de "rascunho" para todo o sistema. Como eles são limpos periodicamente (em teoria), acabam se tornando locais apropriados para armazenar dados que não serão usados no futuro. Por essa razão, muitos atacantes usam-nos como diretórios de serviço, não se importando em apagar os rastros deixados. Além disso, é possível encontrar nesses diretórios arquivos temporários de aplicativos (editores de texto, por exemplo), cujos originais foram cifrados ou nunca foram salvos no disco
Diretório de *device files* (`/dev`)	Com exceção de alguns poucos arquivos especiais, o diretório `/dev` deve conter apenas os *device files*. Muitos atacantes exploram a complexidade do conteúdo desse diretório para esconder suas informações
Arquivos e diretórios ocultos ou não usuais	Diferentes combinações de pontos, espaços e caracteres de controle são frequentemente usadas pelos atacantes para esconder seus arquivos
Executáveis e bibliotecas	Os atacantes comumente instalam *trojan horses*, *rootkits* e *exploits* no sistema invadido. Esses programas geralmente modificam algum executável (binário ou *script*) do sistema ou alguma biblioteca carregada dinamicamente. Além disso, muitos atacantes deixam para trás seus códigos maliciosos (executáveis ou códigos-fonte) que podem ser analisados como detalhado na Seção A.14
Arquivos de *log*	Os arquivos de *log* representam um papel crucial na análise do sistema de arquivos, pois permitem a reconstituição de fatos que ocorreram no sistema. Tais arquivos podem registrar, entre outras informações, as atividades dos usuários, dos processos e do sistema, as conexões e atividades da rede, e informações específicas dos aplicativos e serviços
Arquivos e diretórios de usuários	Arquivos de texto, mensagens de correio eletrônico, registros de conversas em salas de bate papo, arquivos de imagens gráficas, entre outros tipos de dados, podem conter informações úteis relacionadas ao atacante

Tabela 5.2 (continuação): Resumo dos principais arquivos e diretórios que comumente contêm ou representam evidências de uma intrusão.

5.3.7 Arquivos de configuração

Alguns arquivos de configuração representam excelentes locais para se encontrar evidências de uma intrusão [67]. Um atacante experiente pode facilmente modificar a configuração do sistema e aplicativos, com o intuito de instalar *back doors*, esconder sua presença no sistema ou apenas causar estragos. Algumas ferramentas podem facilitar a análise das configurações do sistema, como apresentado na Seção A.13.

Em geral, a configuração de algumas partes do sistema tende a permanecer constante após um determinado momento, salvo algumas pequenas alterações esporádicas para a inclusão de novas máquinas, aplicativos e usuários. Desse modo, os arquivos de configuração podem ser verificados, quanto a modificações inesperadas, através da comparação com cópias de segurança e registros de alterações mantidos pelo administrador do sistema. Nesse sentido, algumas evidências podem ser facilmente encontradas como, por exemplo:

- modificação das permissões de acesso aos arquivos de configuração, facilitando alterações futuras;
- os tempos de modificação dos arquivos de configuração diferem da última modificação legítima;
- os *hashes* criptográficos dos arquivos diferem das cópias de segurança, indicando uma possível alteração indevida;
- remoção de alguns arquivos de configuração (relacionados com controle de acesso, por exemplo);

Entre as configurações mais frequentemente alteradas pelos atacantes estão os *scripts* de inicialização, o controle de acesso à máquina, as relações de confiança, as contas e senhas de usuários, os serviços de rede, as tarefas agendadas e os sistemas de *log* e monitoramento. Alguns desses principais alvos são detalhados na sequência.

5.3.7.1 *Scripts* de inicialização

No sistema Linux, alguns serviços e o próprio sistema operacional são especialmente inicializados e terminados com base nas configurações dos *scripts* localizados no diretório `/etc/rc.d`. Um atacante pode facilmente adicionar comandos a esses *scripts*, permitindo, por exemplo, a execução de programas maliciosos durante a inicialização do sistema ou aplicativos. Outros *scripts* de inicialização encontram-se nos diretórios dos usuários (como, por exemplo, `.bash_profile`, `.bashrc` e `.cshrc`), sendo consultados automaticamente quando o usuário acessa o sistema ou quando determinados programas são

executados. É necessário checar cada *script* de inicialização em busca de comandos maliciosos e verificar se os programas executados a partir deles são legítimos. Como exemplos de comandos maliciosos, encontrados nos *scripts* de inicialização, podem ser citados os seguintes:

- modificar as permissões de acesso de arquivos sensíveis (por exemplo, o comando `chmod 0 /var/log/*` torna inacessíveis todos os arquivos de *log* em `/var/log`);
- colher informações sobre o sistema e armazenar em algum lugar acessível (por exemplo, `cat /etc/shadow >> /tmp/arquivo`);
- enviar informações sensíveis por correio eletrônico;
- adicionar usuários e senhas em `/etc/passwd` e `/etc/shadow`;
- remover arquivos sensíveis;

5.3.7.2 Relações de confiança e controle de acesso

Relações de confiança podem ser convenientes para os administradores da rede, mas introduzem falhas de segurança que são comumente utilizadas pelos atacantes [34]. Alguns utilitários de acesso remoto, como `rsh`, `rlogin` e `rcp` (notórios por sua insegurança), podem se configurados para não utilizarem autenticação de usuário, de modo que os atacantes podem tentar aumentar o conjunto de máquinas com permissão para utilizá-los dessa forma [47]. Essa configuração é feita através dos arquivos `.rhosts`, presentes nos diretórios dos usuários[14], e `/etc/hosts.equiv`. Outras formas de relação de confiança podem ainda ser estabelecidas através do programa `ssh` (por meio dos arquivos `.shosts` e `/etc/ssh/shosts.equiv`), por compartilhamentos através de NFS e pelo serviço NIS [67].

O controle de acesso à máquina pode ser feito, por exemplo, através dos arquivos de configuração `/etc/hosts.allow` e `/etc/hosts.deny`, por meio de *firewalls* e do sistema de autenticação PAM. Os atacantes podem modificar ou "deletar" os arquivos que configuram o controle de acesso e as relações de confiança para permitir seu retorno facilitado à máquina invadida. Desse modo, todas as possíveis relações de confiança e permissões de acesso devem ser investigadas, para determinar se elas estão relacionadas com o incidente. Entre as principais evidências encontradas, com relação ao controle de acesso e às relações de confiança, podem ser citadas:

[14]Os arquivos de configuração localizados nos diretórios dos usuários podem ser encontrados através do comando `find /home -type f -name ''.?hosts'' -print0 | xargs -0 ls -l`. Deve-se prestar atenção especial àqueles encontrados no diretório de usuários com privilégios administrativos.

- permissão de acesso para máquinas desconhecidas (especialmente para máquinas fora da organização);
- permissão de acesso para usuários desconhecidos ou suspeitos;
- permissão de acesso a serviços desconhecidos ou suspeitos (inseguros ou desativados);
- habilitação de relações de confiança quando deveriam estar desativadas;
- presença do caracter "+" nos arquivos de configuração das relações de confiança, permitindo a conexão remota, sem autenticação, proveniente de qualquer máquina;
- compartilhamentos via NFS suspeitos em `/etc/exports` (inseguros ou desconhecidos);
- servidor NIS desconhecido ou diferente do normal em `/etc/yp.conf`;

5.3.7.3 Usuários e grupos

Os arquivos de configuração de contas e senhas de usuários e grupos (`/etc/passwd`, `/etc/shadow` e `/etc/group`) frequentemente mostram sinais de comprometimento em sistemas invadidos [47]. Tais sinais podem vir, por exemplo, na forma de novas contas ou aumento de privilégios de contas existentes, com o objetivo usual de criar *back doors* para futuros acessos. As informações sobre as contas e senhas de usuários e grupos devem ser investigadas em busca de inconsistências ou características suspeitas, como, por exemplo:

- contas criadas sem o conhecimento do administrador do sistema;
- contas sem senha;
- contas com diretório de usuário suspeito (por exemplo, `/tmp`);
- contas com *shell* de *login* suspeito (como, por exemplo, algum outro tipo de programa ou algo diferente de `/bin/false`, quando for o caso);
- entradas nos arquivos de configuração com número impróprio de campos ou com características diferentes das demais entradas;
- contas com USERID e/ou GROUPID suspeitos (principalmente aquelas com USERID ou GROUPID 0 ou 1);
- contas ativas, mas que deveriam estar desabilitadas ou indisponíveis para acesso como, por exemplo, as contas *daemon*, *sync* e *shutdown*;

- usuários suspeitos em grupos de alto privilégio;

Alguns utilitários como `pwck` e `grpck` podem ser utilizados para checar as entradas contidas em `/etc/passwd`, `/etc/shadow` e `/etc/group`. Essas ferramentas apenas verificam se as entradas estão no formato apropriado e se apresentam dados válidos em cada campo, não fazendo nenhuma avaliação quanto a características suspeitas ou inseguras.

5.3.7.4 Serviços de rede

Uma instalação padrão do sistema Linux oferece um grande conjunto de serviços de rede, incluindo NFS, TELNET, FTP, SMTP e muitos outros. Qualquer um desses serviços de rede pode potencialmente permitir algum tipo de acesso remoto a usuários indesejáveis [67]. Alguns do pontos de acesso mais comumente explorados pelos atacantes incluem, por exemplo, servidores X Window, FTP, TELNET, TFTP, DNS, SMTP, SNMP, IMAP, POP, HTTP, HTTPS e RPC [47].

Durante a investigação dos arquivos de configuração dos serviços de rede, todos devem ser examinados como potenciais pontos de acesso ao sistema, considerando que alguns serviços podem apresentar vulnerabilidades ou ter sido substituídos por *trojan horses*. Nesse sentido, é preciso verificar se os pontos de acesso apresentam-se configurados de maneira segura e possuem os *patches* e versões mais recentes dos programas. Como exemplos das principais evidências encontradas em relação à configuração dos serviços de rede, podem ser citadas as seguintes:

- habilitação inesperada de serviços que estavam previamente desativados;
- adição de entradas suspeitas em `/etc/xinetd.conf`, permitindo acesso a portas e serviços incomuns ou desativados;
- adição de portas e serviços suspeitos ou modificação das entradas legítimas em `/etc/services`;
- alterações inesperadas no serviço de resolução de nomes;
- configurações inseguras no servidor FTP (como, por exemplo, permissão para usuários anônimos executarem comandos "perigosos" como `delete` e `chmod` [34]);
- diretório de FTP com permissões indevidas;

5.3.7.5 Tarefas agendadas

O serviço de agendamento de tarefas é comumente disponibilizado através dos programas `crond`, `anacron` e `at`, apresentando diversos arquivos e diretórios de configuração, como, por exemplo, `/var/spool/cron`, `/var/spool/at`, `/var/spool/anacron`, `/etc/crontab`, `/etc/cron.d` e `/etc/anacrontab`. Assim como os administradores do sistema utilizam tarefas agendadas para automatizar suas atividades, os atacantes também fazem uso dessa facilidade. As tarefas agendadas são executadas com os privilégios do proprietário do respectivo arquivo de configuração, que costumam ser explorados pelos atacantes para executar programas maliciosos com os privilégios de outro usuário (geralmente o usuário *root*) [67]. É preciso localizar todos os diretórios e arquivos contendo tarefas agendadas e examinar cuidadosamente cada uma, bem como os executáveis referenciados por elas. Entre as principais evidências encontradas com relação a tarefas agendadas, podem ser citadas as seguintes:

- tarefas desconhecidas ao administrador do sistema;
- referência a executáveis desconhecidos ou incomuns;
- referência a executável com permissão de escrita para todos os usuários;
- arquivos de configuração de tarefas agendadas com permissão de escrita para todos os usuários;

5.3.7.6 Sistemas de segurança

Os sistemas de segurança implementados na máquina invadida também podem ser alvos dos atacantes. Os sistemas de *log* e de detecção de intrusão podem ser alterados com o intuito de esconder os rastros do invasor e permitir que ele retorne sem ser descoberto. Ao analisar a máquina invadida, é importante verificar a configuração do esquema de *log* no arquivo `/etc/syslog.conf`, bem como determinar se as opções de *log* dos diversos serviços e aplicativos estão devidamente configuradas. No caso da máquina analisada hospedar algum tipo de sistema de detecção de intrusão, é necessário verificar a configuração do mesmo, pois o atacante pode ter desabilitado o monitoramento de alguma parte do sistema ou rede.

5.3.8 Arquivos de *log*

Os arquivos de *log* potencialmente representam a fonte de informação mais valiosa sobre as atividades do sistema [47]. Tais arquivos podem registrar, entre outras informações, as atividades dos usuários, dos processos e do sistema, as conexões e atividades da rede,

e informações específicas dos aplicativos e serviços. A maioria dos arquivos de *log* está localizada, no sistema Linux, em um diretório comum, usualmente `/var/log`. Os principais arquivos de *log* que podem ser examinados durante a análise forense são listados na Tabela 5.3.

Considerações gerais

Existem algumas observações importantes que devem ser consideradas durante a análise dos arquivos de *log*, incluindo, por exemplo:

- os arquivos de *log* binários geralmente contêm mais informações do que o mostrado na saída dos programas que os manipulam [67];
- nem todos os programas registram uma entrada nos arquivos `utmp` e `wtmp`, de modo que podem existir mais usuários usando o sistema;
- o comando su não altera os arquivos `utmp` e `wtmp`, de modo que o usuário acessado pelo comando su não é listado na saída dos programas que processam esses *logs* (o atacante pode entrar no sistema, sem levantar suspeitas, através da conta de um usuário sem privilégios e, então, executar o comando su para acessar a conta de *root*);
- os arquivos de *log* podem ter nomes e funcionalidades diferentes daqueles apresentados na Tabela 5.3 ou podem estar localizados em outras partes do sistema, em outras máquinas ou em cópias de segurança. É importante consultar o administrador do sistema e o arquivo de configuração do serviço *syslog*[15] (`/etc/syslog.conf`), para entender a organização do esquema de *log* do sistema invadido;
- a capacidade de *log* de alguns serviços ou aplicativos pode estar desabilitada ou, ainda, configurada para um nível de detalhes insuficiente;
- os arquivos de *log* podem sofrer alterações deliberadas, principalmente aqueles armazenados em formato texto. Se o atacante conseguir acesso de *root* no sistema, ele pode facilmente modificar os arquivos de *log*, removendo, alterando ou acrescentando entradas nos arquivos [67];

[15] O *syslog* é uma facilidade que utiliza um processo de *log* centralizado chamado `syslogd`. Os programas individuais que desejam registrar mensagens de *log* enviam-nas para o processo `syslogd`, que se encarrega de registrá-las. Tais mensagens podem ser armazenadas em vários arquivos locais ou em sistemas remotos, dependendo da configuração do serviço *syslog*. Maiores informações sobre o *syslog* podem ser encontradas em [34] ou na *man page* do programa `syslogd`.

Arquivo de *log*	Descrição e características
`utmp`	Registra os usuários atualmente utilizando o sistema. Apresenta-se no formato binário e pode ser acessado pelos comandos `w`, `who`, `users` e `finger`. Encontra-se no diretório `/var/run`
`wtmp`	Registra todos os `login`s, `logout`s, `reboot`s, `shutdown`s e mudanças no tempo do sistema. Apresenta-se no formato binário e pode ser acessado pelos comandos `last` e `ac`
`btmp`	Registra as tentativas mal sucedidas de `login`. Apresenta-se no formato binário e pode ser acessado pelo comando `lastb`
`lastlog`	Registra o momento e origem do `login` mais recente de cada usuário. Apresenta-se no formato binário e pode ser acessado pelo comando `lastlog`
`boot.log` e `dmesg`	Registram as mensagens relativas ao processo de inicialização do sistema. Apresentam-se no formato texto
`messages` ou `syslog`	Registra vários eventos e informações do sistema e aplicativos. Representa o principal arquivo de `log` do sistema e geralmente contém informações encontradas também em outros arquivos de `log` como, por exemplo, tentativas mal sucedidas de `login`
`secure`	Registra mensagens privadas de programas relativos a autorização de usuários. Apresenta-se no formato texto
`sulog`	Registra o uso do comando `su`
`access_log`	Registra os acessos ao servidor HTTP. Apresenta-se no formato texto
`xferlog`	Registra os acessos ao servidor FTP. Apresenta-se no formato texto
`cron`	Registra a execução de tarefas agendadas. Apresenta-se no formato texto
`maillog`	Registra mensagens relativas ao serviço de correio eletrônico. Apresenta-se no formato texto
`aculog`	Registra o uso de conexões `dial-out`
`pacct` ou `acct`	Registram os processos executados por cada usuário. Apresentam-se no formato binário e podem ser acessados pelos comandos `lastcomm` e `sa`
`History files` (`.history`, `.sh_history`, `.bash_history`)	Registram os comandos recentemente executados por cada usuário. Apresentam-se no formato texto e encontram-se nos diretórios de cada usuário
`Log`s de `firewall`s e sistemas de detecção de intrusão	Registram conexões permitidas e/ou rejeitadas e eventos caracterizados como possíveis intrusões

Tabela 5.3: Principais arquivos de *log* encontrados durante a análise forense.

- a falta de autenticação no uso do *syslog* permite que qualquer usuário registre mensagens de *log* [34]. Essa característica abre a possibilidade de inserção de mensagens falsas nos arquivos de *log* mantidos pelo *syslog*;

- o *syslog* utiliza um mecanismo não confiável para o envio de mensagens de *log* para uma máquina remota (através do protocolo UDP), de modo que não há como determinar se as mensagens de *log* foram realmente registradas na máquina remota [10];

- a falta de sincronização dos relógios dos sistemas que utilizam o *syslog* pode modificar totalmente a ordem dos eventos registrados. O *syslog* marca as entradas de *log* recebidas com a data e hora da máquina onde a entrada será armazenada, ao invés dos tempos do sistema que produziu a mensagem de *log*. Além disso, mesmo que as mensagens de *log* sejam armazenadas localmente, o *syslog* pode introduzir algum atraso entre o tempo em que a mensagem é gerada e o momento em que ela é gravada no respectivo arquivo de *log* [10];

Circunstâncias suspeitas relacionadas aos arquivos de *log*

Os arquivos de *log* podem conter evidências importantes para o processo de análise forense, permitindo recontruir os eventos relacionados ao comportamento intrusivo, levantar possíveis suspeitos e, até mesmo, identificar o ponto de entrada e o momento da intrusão. Algumas das principais circunstâncias suspeitas encontradas com relação aos arquivos de *log* são resumidas a seguir:

- remoção do arquivo de *log* ou seu redirecionamento para `/dev/null`;

- indicações de modificações deliberadas nos arquivos de *log* como, por exemplo, o primeiro registro contendo uma data posterior à data esperada para início do serviço de *log*, extensos períodos de tempo sem qualquer registro, ausência de alguma mensagem de *log* específica que deveria estar presente (proveniente de algum serviço que certamente deveria estar gerando mensagens de *log* ou algum registro de *logout* sem o registro de *login* correspondente, por exemplo), registros com alguma informação inconsistente ou ausente, ou ainda alguma desordenação nos tempos cronológicos das entradas (as entradas devem ter sempre um incremento maior ou igual a zero nas marcas de tempo);

- registros de atividades em horários ou datas não usuais (datas e horários sem expediente em uma organização, por exemplo);

- registros de atividades não usuais (podem ser apenas visíveis ao administrador do sistema);
- registro de *login* provenientes de origens suspeitas ou não usuais (originados, por exemplo, de fora da organização, de outros países ou de origens conhecidamente mal-intencionadas);
- registros de falhas de *login* (principalmente se forem várias sucessivas);
- mensagens relacionadas ao uso não-autorizado ou mal sucedido do comando su (principalmente tentando acessar a conta de *root*);
- mensagens de erro provenientes de serviços de rede (a maioria dos serviços de rede produzem mensagens de erro quando são comprometidos pelos atacantes [47]);
- registros de *reboots* e *shutdowns* suspeitos (datas e horários imprevistos ou não usuais);
- registros de alterações indevidas na data e hora do sistema;
- entradas de *log* fora do padrão normal, contendo caracteres não usuais, embaralhados ou não legíveis, podem indicar tentativas de ataques de *buffer overflow*. Além disso, entradas contendo NOPs e comandos UNIX também podem estar relacionadas com esse tipo de ataque;
- mensagens de serviços que deveriam estar desabilitados;
- registros de conexões de rede provenientes de origens desconhecidas ou suspeitas;
- mensagens relacionadas a conexões de rede estabelecidas no intervalo de tempo em que se suspeita ter acontecido a intrusão;
- registros de transmissão de arquivos suspeitos via FTP;
- mensagens de pacotes rejeitados pelo *firewall*;
- mensagens de advertência produzidas por um sistema de detecção de intrusão;

Além das possíveis evidências relacionadas anteriormente, algumas circunstâncias suspeitas podem ser encontradas exclusivamente nos arquivos de *log* que registram os comandos executados pelos usuários (*history files* e `pacct`), como, por exemplo:

- obtenção de informações sobre o sistema (execução de comandos como `ps`, `ifconfig`, `netstat`, `what` e visualização de arquivos de configuração);

- remoção ou adição não-autorizada de usuários;
- transferência de arquivos suspeitos ou provenientes de origens suspeitas;
- criação ou acesso a arquivos e diretórios ocultos ou com nomes pouco usuais;
- execução de programas não-autorizados ou desconhecidos;
- remoção de arquivos ou diretórios sensíveis (arquivos de *log* ou de configuração, por exemplo);

A análise dos arquivos de *log* representa um verdadeiro desafio à medida que eles crescem substancialmente. Existem ferramentas automatizadas que permitem varrer um arquivo de *log* em busca de padrões específicos definidos pelo usuário. Como exemplo desse tipo de ferramenta pode ser citado o Swatch[16], que busca por padrões definidos no formato de expressões regulares da linguagem Perl. Já os arquivos de *log* binários, listados na Tabela 5.3, podem ser processados e analisados com a ajuda dos respectivos utilitários apresentados na referida tabela.

5.4 Correlação de evidências

A recontrução dos eventos e formulação de conclusões acerca de um incidente muitas vezes requer mais do que a simples identificação de evidências isoladas nas diversas fontes de informações do sistema comprometido. Na maioria dos casos é preciso correlacionar as informações extraídas do sistema invadido, quer seja para corroborar uma suspeita ou para identificar novas pistas, levando a um maior entendimento sobre o incidente.

O relacionamento entre duas ou mais informações pode ser estabelecido através de vários parâmetros como, por exemplo, registros de tempo, relações de causa e efeito, e números de identificação (PID, USERID, GROUPID, endereços IP e portas de serviços de rede, por exemplo). Por esse motivo, é importante que as evidências encontradas no sistema comprometido sejam descritas minuciosamente, contendo os dados necessários para um possível correlacionamento.

A construção de uma linha de tempo e de gráficos relacionais pode ajudar o investigador a visualizar com maior clareza a ordem dos eventos e suas relações, permitindo a identificação de padrões e a descoberta de novas evidências. Muitas vezes os processos de busca e correlacionamento de evidências se misturam, de modo que a descoberta de uma evidência gera informações necessárias para a identificação de outras.

[16]Maiores informações sobre a ferramenta Swatch podem ser encontradas em [34] e o programa é acessível através da URL http://coast.cs.purdue.edu/pub/tools/unix/logutils/swatch/ (disponível em janeiro de 2003).

5.5 Estrutura geral do processo de investigação forense

O processo de investigação forense, seja para fins judiciais ou corporativos, deve garantir a autenticidade e integridade das evidências coletadas e dos resultados produzidos [9]. Em outras palavras, a investigação forense deve assegurar que as informações obtidas existem nas evidências analisadas e não foram alteradas ou contaminadas pelo processo de investigação. Isso pode ser particularmente difícil em se tratando de evidências digitais, devido ao seu alto grau de volatilidade. O simples ato de ligar ou desligar o computador pode alterar ou destruir evidências. Por esse motivo, é importante que a investigação seja conduzida de maneira metódica, bem organizada e em sintonia com a tecnologia envolvida [9].

Desse modo, para suportar os resultados da investigação forense, são necessários procedimentos e protocolos que assegurem que as evidências são coletadas, preservadas e analisadas de maneira consistente, minuciosa e livre de contaminações [10]. Tais procedimentos são comumente definidos pelo termo *Standard Operating Procedures* (SOP) e representam um guia prático, documentado, de controle de qualidade, que deve ser aplicado toda vez que um sistema é analisado [10, 74]. Os procedimentos e protocolos de análise forense devem ser detalhados, documentados, revisados e aceitos pela comunidade científica relevante [73], sendo essenciais para evitar erros durante o processo de investigação, para assegurar que as melhores técnicas disponíveis sejam utilizadas e para aumentar a probabilidade de dois investigadores chegarem às mesmas conclusões ao examinarem as mesmas evidências [10].

Com o objetivo de garantir que as evidências digitais sejam coletadas, preservadas e examinadas de maneira a resguardar sua autenticidade e integridade, os procedimentos e protocolos usados no processo de investigação forense devem ser coerentes com um conjunto de princípios básicos, que representam conceitos e práticas importantes entre os profissionais da área [73, 74]. Alguns princípios e boas práticas são sugeridos pela Associação de Oficiais Chefes de Polícia do Reino Unido (ACPO) [1] e pelo Grupo de Trabalho Científico em Evidências Digitais (SWGDE)[17] [74]. Alguns desses princípios são apresentados como segue:

- as ações tomadas durante a investigação forense não devem alterar as evidências;
- qualquer ação que tenha o potencial de alterar, danificar ou destruir qualquer aspecto da evidência original deve ser conduzida por uma pessoa qualificada (por exemplo,

[17]Maiores informações sobre o SWGDE podem ser encontradas na URL http://www.for-swg.org/swgdein.htm (disponível em janeiro de 2003).

quando a evidência original precisa ser acessada para coleta de informações voláteis ou para a criação de imagens dos discos suspeitos);

- o investigador não deve confiar cegamente no sistema invadido, nem nos programas e bibliotecas dinâmicas nele encontrados;

- cópias das evidências originais devem ser produzidas e, sempre que possível, a investigação deve ser conduzida sobre as cópias. Tais cópias devem ser idênticas às evidências originais, contendo toda a informação em seu estado original;

- *hashes* criptográficos de todas as evidências digitais coletadas e cópias produzidas devem ser gerados, permitindo a verificação de autenticidade e integridade;

- toda evidência coletada deve ser identificada, contendo o número do caso investigado, uma breve descrição da evidência e a data e horário da coleta;

- toda evidência coletada deve ser preservada em local de acesso controlado e livre de alterações;

- todas as informações relativas à investigação (atividades relacionadas à aquisição, armazenamento, análise e transferência das evidências, anotações e observações do investigador e os resultados produzidos) devem ser documentadas de maneira permanente e devem estar disponíveis para revisão. A documentação das ações executadas e dos resultados obtidos deve ser feita em relatórios minuciosos, contendo detalhes que incluem as versões das ferramentas utilizadas, os métodos empregados para coleta e análise das evidências e explicações que fundamentem a utilização desses métodos. Desse modo, outro investigador deve ser capaz de examinar as informações documentadas e chegar às mesmas conclusões;

- a cadeia de custódia das evidências coletadas deve ser mantida, documentando a jornada completa de cada evidência durante a investigação. Devem ser relatadas, entre outras informações, o nome da pessoa que coletou a evidência, como, onde e quando foi feita a coleta, o nome do investigador que está de posse da evidência, data e horário de retirada e devolução da evidência e as atividades executadas pelo investigador;

- as ferramentas usadas na investigação (*hardware* e *software*) devem ser amplamente aceitas na área e testadas para garantir sua operação correta e confiável. Além disso, elas devem ser conhecidas em cada detalhe, evitando implicações inesperadas na evidência analisada;

- os procedimentos devem ser aceitos pela comunidade científica relevante ou suportados por demonstrações da precisão e confiabilidade das técnicas aplicadas;
- os procedimentos devem ser revistos periodicamente para garantir sua contínua adaptação às evoluções tecnológicas;
- o investigador deve ser responsável pelos resultados da investigação e pelas evidências enquanto estiverem em sua posse;
- a pessoa responsável pela investigação deve assegurar o cumprimento dos procedimentos e protocolos estabelecidos;

A estratégia de investigação forense deve fornecer um guia passo-a-passo para se conduzir a análise do sistema invadido [100]. Entretanto, existem múltiplas variantes que tornam uma investigação particularmente diferente das outras, como, por exemplo, a tecnologia envolvida, as configurações do sistema, as condições em que o sistema é encontrado e restrições impostas à investigação. Desse modo, a estratégia de análise forense em sistemas computacionais pode ser definida em linhas gerais, permitindo que o investigador possa utilizá-la, em todo ou em parte, como um roteiro na grande maioria das investigações. Nesse sentido, a estrutura geral do processo de investigação forense pode ser definida em termos dos seguintes passos:

- considerações e inteligência preliminares;
- planejamento;
- estabilização do sistema e decisões iniciais;
- coleta, autenticação, documentação e preservação de material para análise;
- análise;

Como as chances de sucesso aumentam conforme o processo de investigação é melhor definido [100], alguns procedimentos específicos utilizados na estratégia geral podem ser detalhados, incluindo, por exemplo, o uso de determinadas ferramentas e técnicas. Existem pesquisas no campo da forense computacional que visam o desenvolvimento de procedimentos e protocolos segundo uma abordagem hierárquica, onde no topo dessa hierarquia encontram-se os princípios e boas práticas que o processo de investigação deve obedecer, seguidos da estratégia geral de análise forense e terminando, na base da hierarquia, com os procedimentos detalhados sobre o uso de ferramentas e técnicas específicas [82]. O Apêndice C apresenta uma discussão sobre o desenvolvimento e padronização de

procedimentos e protocolos de análise forense, segundo a abordagem hierárquica resumida anteriormente.

As diversas fases da estrutura geral do processo de investigação forense são discutidas na sequência.

5.5.1 Considerações e inteligência preliminares

Antes de abordar o sistema computacional comprometido (ou suspeito de comprometimento) é preciso considerar uma série de questões e fazer um trabalho inicial de inteligência [9]. Algumas dessas considerações e tarefas de inteligência preliminares são apresentadas como segue:

- entrar em contato com o administrador do sistema, que comumente é a pessoa mais indicada a responder questões sobre o sistema e, geralmente, é o primeiro a perceber e relatar o incidente;
- entender o problema. A compreensão acerca dos motivos e suspeitas que originaram a investigação pode ser vital para o andamento do processo;
- compreender as condições iniciais em que o sistema será entregue para a investigação. Questões comumente abordadas neste momento incluem, por exemplo, determinar se o sistema já foi desligado ou encontra-se em operação, se foram tomadas medidas de contenção e resposta a incidentes e se foi coletado algum tipo de informação;
- conhecer as políticas de segurança e privacidade aplicadas. É importante identificar as responsabilidades sobre cada parte do sistema e as relações de propriedade sobre as informações nele contidas. Desse modo, o investigador sabe a quem recorrer em busca de esclarecimentos ou permissões de acesso. Além disso, é importante observar questões, abordadas na política de segurança, com relação ao controle de acesso ao sistema e à implementação de esquemas de *log* e monitoramento;
- determinar se alguma lei ou direito individual pode ser violado. Algumas informações podem estar protegidas por leis rigorosas de privacidade, havendo necessidade de cuidados extremos em relação ao acesso ou disponibilização acidental desse tipo de informação. Em alguns casos, principalmente em investigações criminais, um mandado de busca e apreensão será necessário para a realização da investigação [9]. Nesse caso, a investigação deve observar rigorosamente o disposto no mandado;
- conhecer as premissas e restrições impostas à investigação como, por exemplo, impossibilidade de desligamento do sistema ou remoção de qualquer componente físico e cuidados com qualquer interrupção no fornecimento dos serviços;

- levantar informações sobre o alvo da investigação como, por exemplo, o tipo de equipamento e tecnologia envolvidos, tipo e versão do sistema operacional, conexões e topologia da rede, serviços e configurações implementados e dados sobre os usuários do sistema;

5.5.2 Planejamento

Com base nas informações adquiridas na etapa anterior, a investigação pode ser planejada com o maior nível de detalhes possível. Entre as principais atividades da etapa de planejamento estão as seguintes:

- definir a melhor abordagem para a investigação, identificando as principais atividades que precisarão ser executadas (considerando as condições iniciais em que o sistema será entregue para a investigação);

- preparar o conjunto de ferramentas e o sistema de análise, com a mais adequada configuração de *hardware* e *software*. Pode ser necessário gerar uma mídia removível (CDROM ou disquete, por exemplo) contendo o conjunto de ferramentas, permitindo sua utilização no sistema comprometido. É importante que as ferramentas e o sistema de análise sejam confiáveis, que tenham sua integridade checada e seu funcionamento testado. No caso de utilitários que fazem uso de bibliotecas dinânicas, estas devem fazer parte do conjunto de ferramentas ou devem ser compiladas juntamente com os programas (desse modo, apenas o sistema operacional da máquina suspeita é utilizado pelas ferramentas de análise, no caso da coleta de informações voláteis, como será visto adiante nesta seção);

Na maioria das ocasiões uma operação urgente e imediata é necessária, com o objetivo de reunir o maior número de informações sobre o incidente e diminuir os estragos causados pelo atacante [87]. Desse modo, um planejamento detalhado e considerações preliminares sobre o incidente, embora importantes para o processo de análise forense, podem não ser possíveis [87]. Isso reforça a necessidade de adoção, por parte das organizações, de políticas de resposta a incidentes, bem como de treinamento e manutenção de times de especialistas responsáveis pelas primeiras medidas quando da ocorrência de uma invasão [67].

5.5.3 Estabilização do sistema e decisões iniciais

Esta etapa só se aplica se o sistema comprometido ainda estiver em funcionamento. Com a máquina ainda em operação, o investigador pode deparar-se com situações do tipo:

- o atacante ainda se encontra no sistema;
- processos em execução ou conexões abertas foram deixadas pelo atacante, representando evidências importantes para a investigação;
- o atacante preparou armadilhas (*booby traps*) que visam a destruição das evidências deixadas ou a danificação do sistema;

Desse modo, em um primeiro contato com o sistema suspeito, o investigador deve estabilizar a condição inicial da máquina para a etapa seguinte de coleta de informações, com o objetivo de preservar o máximo de evidências e proteger os sistemas e dados fora do escopo da investigação. É importante, também, descrever o sistema suspeito em sua condição inicial e tomar nota de todas as atividades executadas.

Nesta etapa, algumas decisões importantes devem ser tomadas, por exemplo, com relação ao método mais adequado de coleta de informações, à necessidade de coleta de dados voláteis (conteúdo da memória e informações sobre o estado do sistema operacional, por exemplo), ao método de desligamento do sistema (caso seja necessário e possível), à necessidade de coleta de tráfego de rede e possibilidade de rastreamento do atacante e à necessidade de remoção do sistema para um ambiente controlado de análise. A avaliação da necessidade e método de execução dessas e outras atividades é melhor conduzida após uma análise inicial do sistema em funcionamento e, por esse motivo, é discutida nesta etapa da estratégia de análise forense.

Decidir entre manter a máquina em funcionamento, desligá-la imediatamente através da interrupção do fornecimento de energia, ou proceder ao desligamento administrativo normal do sistema é uma das questões mais amplamente discutidas no campo da forense computacional [47]. Muitos investigadores alegam que o desligamento imediato pelo cabo de energia é a melhor opção para congelar o sistema em seu estado corrente, considerando aceitável que algumas evidências sejam perdidas em face à preservação da integridade daquelas presentes nos dispositivos de armazenagem secundária [10, 47, 100]. O principal argumento a favor dessa abordagem é que qualquer erro cometido em um sistema em funcionamento não pode ser remediado, de modo a desfazer todas as suas consequências, que podem incluir, por exemplo, a alteração ou destruição de informações fundamentais à investigação [47]. Além disso, a manutenção do sistema em funcionamento e o seu desligamento administrativo normal podem expor o investigador a situações potencialmente catastróficas, como, por exemplo:

- muitos atacantes instalam armadilhas que destroem algumas evidências importantes ou danificam o sistema, quando este é desligado. No ambiente UNIX existem diversos arquivos envolvidos no processo de desligamento administrativo que podem ser facilmente alterados [100];

- o próprio processo de desligamento administrativo pode alterar ou remover arquivos como parte do procedimento normal;
- um atacante ainda no sistema pode desconfiar da investigação e iniciar uma limpeza das evidências deixadas, podendo até danificar o sistema;

Por outro lado, o desligamento imediato pelo cabo de energia, antes da coleta das informações voláteis, pode resultar em uma grande perda de evidências (principalmente aquelas associadas a um ataque em andamento), que talvez possam ser encontradas apenas enquanto o sistema estava em operação [10, 47]. Além disso, um desligamento abrupto do sistema pode corromper dados importantes ou danificar algum dispositivo físico [100]. Em muitos casos, ainda, o desligamento do sistema pode não ser permitido pela gerência da organização.

De fato, cada abordagem relacionada ao desligamento do sistema apresenta vantagens e desvantagens, que devem ser consideradas pelo investigador de acordo com situações particulares da investigação em questão. O importante é conhecer as implicações de cada abordagem e manter-se flexível quanto à utilização de cada uma.

Igualmente dependente de situações particulares da investigação em questão, é a decisão sobre o que fazer quando se descobre que o intruso ainda está presente no sistema. Basicamente três opções podem ser consideradas [100]: o investigador pode mantê-lo no sistema para coletar informações e iniciar um rastreamento[18], a conexão do atacante pode ser interrompida (por *software* ou pela desconexão do cabo de rede) ou o investigador pode tentar repreender o intruso a deixar o sistema e não mais retornar.

Em resumo, esta etapa da estrutura geral do processo de investigação envolve decisões importantes que afetarão na escolha do método a ser utilizado para a coleta e análise das informações do sistema suspeito. A escolha desse método está relacionada com o nível de esforço empregado para proteger as evidências e evitar a execução de código hostil, como apresentado na Tabela 5.4.

[18] Maiores informações sobre o rastreamento do atacante (*backtracking*) podem ser encontradas em [67, 100].

Método	Vantagens	Desvantagens
Usar um sistema de análise dedicado para examinar o disco suspeito (protegido contra escrita) ou uma imagem do mesmo	Não requer preocupação quanto a validade do *software* ou *hardware* da máquina suspeita	Pode depender do desligamento do sistema suspeito. Pode resultar na perda de informações voláteis
Inicializar a máquina comprometida usando um *kernel* e ferramentas verificados e protegidos contra escrita, contidos em uma mídia removível	Conveniente e rápido. Nesse caso, os discos da máquina suspeita devem ser montados somente para leitura	Assume que o *hardware* da máquina suspeita não foi comprometido (o que é raro). Resulta na interrupção dos serviços disponibilizados pelo sistema suspeito e pode causar a perda de informações voláteis
Reconstruir o sistema suspeito a partir de sua imagem e, então, examiná–lo	Recria completamente o ambiente operacional do sistema suspeito, sem o risco de alterar as informações originais	Requer disponibilidade de *hardware* idêntico ao do sistema suspeito. Pode resultar na perda de informações voláteis
Examinar o sistema suspeito através de mídias removíveis contendo ferramentas verificadas	Conveniente e rápido. Permite acessar informações voláteis	Se o *kernel* estiver comprometido, os resultados podem ser inconsistentes
Verificar o *software* contido no sistema suspeito e, então, utilizá–lo para conduzir o exame	Requer uma preparação mínima. Permite acessar informações voláteis e pode ser realizado remotamente	Pode tomar muito tempo e o programa usado para verificação de integridade pode estar comprometido. A falta de proteção contra escrita nos discos do sistema suspeito pode resultar na alteração ou destruição de informações
Examinar o sistema suspeito usando o *software* nele contido, sem qualquer verificação	Requer nenhuma preparação. Permite acessar informações voláteis e pode ser realizado remotamente	Método menos confiável e representa exatamente aquilo que os atacantes esperam que seja feito. Na maioria das vezes é uma completa perda de tempo

Tabela 5.4: Relação entre o método de coleta e análise e o nível de esforço empregado para proteger as evidências e evitar a execução de código hostil.

5.5.4 Coleta, autenticação, documentação e preservação de material para análise

Existem dois grandes perigos ao se coletar material para análise: perda e alteração. Se os cuidados necessários não forem tomados, dados importantes podem ser sobrescritos e perdidos totalmente, ou apenas parte deles, alterando seu significado ou apagando informações críticas [100]. Uma coleta conduzida de maneira inadequada pode comprometer totalmente a investigação forense, em particular aquela com fins judiciais [100]. Desse modo, alguns aspectos fundamentais devem ser considerados durante a etapa de coleta de material para análise:

- tratar cada informação como se ela fosse ser usada para fins judiciais;
- coletar o máximo de material possível, observando sua ordem de volatilidade. É importante observar que uma vez terminada a etapa de coleta, geralmente não há retorno, de modo que algumas das informações inicialmente consideradas desnecessárias podem posteriormente não estar disponíveis [47]. O termo "material" é usado no sentido amplo de "tudo que pode ser coletado", seja fisicamente tangível ou não, como por exemplo, informações digitais, *hardware*, documentação, configuração das conexões externas da máquina (cabos de rede, impressoras e outros dispositivos externos) e anotações em geral;
- autenticar, identificar, catalogar e preservar cada item coletado;
- produzir cópias exatas e autenticadas das informações digitais coletadas;
- qualquer atividade executada diretamente no sistema suspeito não deve utilizar os programas nele encontrados (com exceção do *kernel* do sistema operacional, com o risco deste também estar comprometido), pois o atacante pode ter antecipado uma possível investigação, alterando alguns dos executáveis do sistema [47];

A coleta de informações digitais de um sistema suspeito pode ser tecnicamente desafiadora, mas o procedimento ideal pode ser resumido como segue: coletar as informações voláteis, desligar a máquina suspeita, instalar o disco da máquina suspeita no sistema de análise e produzir sua imagem [9]. Como visto anteriormente, nem todas essas atividades podem ser executadas, de modo que a coleta de informações pode apresentar variações de acordo com situações particulares da investigação em questão.

A etapa de coleta envolve uma série de atividades relacionadas, incluindo, por exemplo, a documentação, transporte e armazenamento dos itens coletados; a autenticação das informações digitais; a coleta de dados voláteis; e a produção de imagens dos discos suspeitos. As principais atividades da etapa de coleta de material para análise são detalhadas como segue.

Documentação, transporte e armazenamento

A documentação dos itens coletados é uma das atividades de grande importância realizadas durante a etapa de coleta de material para análise. Cada item coletado deve ser unicamente identificado e minuciosamente descrito em seu estado original. Além disso, essa descrição deve identificar a localização original do item, data e hora da coleta e a identificação da pessoa responsável. Não menos importantes são as atividades de transporte e armazenamento dos itens coletados, que devem zelar pela integridade dos mesmos,

tomando especial cuidado, por exemplo, com campos magnéticos, quedas e acessos não-autorizados.

Autenticação das informações digitais

A autenticidade e integridade das informações digitais coletadas pode ser estabelecida através de *hashes* criptográficos, como o MD5 e SHA [34, 88]. É possível determinar que uma informação digital coletada é autêntica, através da simples comparação do seu *hash* criptográfico (produzido, por exemplo, pelo comando `md5sum`) com o *hash* criptográfico da informação original (produzido de maneira idêntica ao da cópia) [47]. Entretanto, muitas vezes essa comparação não é possível, pois a informação original não existe mais ou foi alterada (principalmente no caso de informações voláteis ou quando o sistema suspeito não pôde ser desligado). Já no caso de checagem de integridade, o *hash* criptográfico produzido no momento da coleta (tal procedimento é exemplificado adiante) permite provar, a qualquer momento, que os dados usados durante a análise são idênticos às informações inicialmente coletadas [47].

Coleta de dados voláteis

Durante a coleta de informações voláteis, o sistema suspeito precisa ser acessado ainda em funcionamento. Nesse caso, é importante observar duas questões fundamentais: não utilizar os programas contidos no sistema suspeito e não escrever em seus discos. A primeira questão pode ser abordada através da utilização de algum tipo de mídia removível (protegida contra escrita), contendo todo o conjunto de ferramentas de coleta (confiáveis, compiladas estaticamente e testadas). Essa abordagem considera que o sistema operacional da máquina suspeita não foi comprometido pelo atacante, o que pode ser investigado, através das técnicas apresentadas na Seção A.9, antes de se proceder à coleta das informações.

Para impedir escritas acidentais nos discos da máquina analisada, é importante conhecer a fundo o funcionamento das ferramentas executadas no sistema suspeito. Os dados produzidos por essas ferramentas representam as informações voláteis coletadas, que precisam ser armazenadas para a etapa seguinte de análise. Como elas não podem ser salvas nos discos do sistema suspeito, as informações geradas pelas ferramentas devem ser direcionadas para uma mídia removível ou transmitidas para o sistema de análise, por exemplo, através da rede. A transmissão das informações pela rede pode ser feita como detalhado na Seção A.15.

Imagem dos discos suspeitos

Outro procedimento importante da etapa de coleta de material para análise é a produção de imagens dos discos da máquina comprometida. O princípio por trás desse

procedimento é a obtenção de toda informação contida no disco, seja ela pertencente a um sistema de arquivos ou não, de tal forma que a imagem possa ser examinada como se o disco original estivesse sendo analisado [87]. Para produzir uma imagem, é necessária alguma ferramenta que leia cada bit do disco suspeito, do início ao fim, sem qualquer alteração na ordem ou conteúdo [87]. Portanto, programas normais de cópia e *backup* (como `cp` e `tar`, por exemplo) não devem ser usados, pois eles copiam apenas os dados reconhecidos pelo sistema de arquivos [10].

Existem diversos utilitários que podem ser usados para a produção de imagens, inclusive equipamentos especiais dedicados como, por exemplo, o Image MaSSter Solo Forensic Unit[19]. O procedimento mais comumente utilizado é a instalação do disco suspeito no sistema de análise, onde a imagem é produzida e armazenada em um único arquivo. Dependendo da situação, a imagem pode ser dividida em arquivos menores, contendo separadamente cada partição do disco suspeito, ou, ainda, utilizada para espelhar o disco suspeito em um outro disco de capacidade similar ou maior [87]. Essa última abordagem, chamada de espelhamento, permite reconstituir exatamente o disco investigado, de modo que os dados contidos em um determinado setor do disco suspeito aparecem no mesmo setor do disco espelhado [87]. Caso o sistema suspeito não possa ser desligado[20], as imagens dos discos podem ser feitas a partir da máquina comprometida e enviadas para o sistema de análise pela rede, de maneira análoga à coleta de informações voláteis descrita anteriormente. As diversas abordagens para o procedimento de imagem são detalhadas na Seção A.16.

5.5.5 Análise

A etapa de análise representa o objetivo principal do processo de investigação forense. É o momento em que todo o material coletado é minuciosamente examinado em busca de evidências, permitindo formular conclusões acerca do incidente que originou a investigação. Durante a análise, é importante investigar todas as fontes de informação do sistema suspeito, buscando identificar características anormais e indevidas, possivelmente provocadas pelo atacante. Conhecer o *modus operandi* dos atacantes e manter um contato próximo com o administrador do sistema comprometido são requisitos essenciais para a eficácia e precisão do processo de análise, pois aumentam a capacidade do investigador de reconhecer possíveis evidências.

A documentação das tarefas executadas e evidências encontradas, bem como a manutenção da cadeia de custódia dos itens analisados, devem ser atividades rotineiras durante

[19]Maiores detalhes sobre o Image MaSSter podem ser encontrados na URL http://www.ics-iq.com (disponível em janeiro de 2003).

[20]Com o sistema suspeito em funcionamento, é importante observar que os *hashes* criptográficos de seus discos podem apresentar valores diferentes a cada vez que forem gerados.

a etapa de análise. Outra atividade importante é a correlação das evidências encontradas, permitindo, entre outras conclusões, determinar se houve realmente um incidente de segurança; reconstruir as atividades do atacante; e identificar causas, suspeitos e consequências da invasão. Com base nos resultados obtidos pela investigação, um relatório final pode ser elaborado (o laudo pericial, no caso de uma perícia criminal), servindo de base para amparar uma ação judicial ou para auxiliar na reconstituição do sistema comprometido e revisão das soluções de segurança da corporação.

5.6 Conclusão

Este capítulo introduziu os principais conceitos e técnicas aplicados na área da forense computacional, fornecendo, em conjunto com o Apêndice A, uma descrição detalhada sobre onde, como e o quê procurar em um sistema computacional invadido.

Por se uma área de pesquisa bastante recente e relacionada a um ambiente em constante evolução, a forense computacional necessita manter-se atualizada em relação aos desenvolvimentos técnico-científicos. A revisão apresentada neste capítulo constitui um esforço no sentido de suprir essa necessidade de desenvolvimento na área da forense computacional.

Do ponto de vista prático, a discussão acerca das várias fontes de informação em um sistema computacional invadido, detalhando as técnicas de extração dos dados e as principais evidências comumente encontradas, fornece um guia resumido para aqueles que estão se iniciando na área. De maneira análoga, a estrutura geral apresentada para o processo de investigação forense constitui um ponto de partida para o desenvolvimento de bons procedimentos de análise, podendo ser aplicada a qualquer tipo de investigação envolvendo sistemas computacionais. Por fim, a apresentação de diversas ferramentas, com exemplos práticos de utilização, permite orientar o investigador na escolha e manipulação de seus utilitários de análise.

Capítulo 6

Automatização do processo de análise forense

Com base na revisão apresentada no capítulo anterior, detalhando os conceitos e técnicas aplicados na investigação forense de intrusões em sistemas computacionais, este capítulo discute a automatização do processo de análise forense, fornecendo subsídios para o desenvolvimento do componente gerador de assinaturas do modelo de IDS apresentado no Capítulo 4.

Com o intuito de viabilizar essa automatização, é apresentada, na Seção 6.2, uma arquitetura extensível para o desenvolvimento de um sistema automatizado de análise forense. Parte dessa arquitetura é implementada em um protótipo inicial, detalhado na Seção 6.3. Por fim, a aplicação, no modelo de IDS imunológico, dos subsídios aqui apresentados é discutida na Seção 6.4.

6.1 Introdução

Conforme o apresentado no Capítulo 5, a análise forense de um sistema computacional invadido pode ser resumida em três grandes etapas: coleta de informações para análise (também conhecida como *information gathering*), busca e extração de evidências nas informações coletadas e análise dos vestígios encontrados.

A etapa de coleta de informações busca reunir, de maneira livre de alterações, o máximo de dados sobre o sistema invadido, incluindo, por exemplo, imagens dos discos do sistema e dados sobre o estado do sistema operacional. Em seguida, durante a etapa de busca e extração de evidências, as informações coletadas são minuciosamente analisadas pelo investigador, que se baseia em seu conhecimento para reconhecer os indícios possivelmente relacionados com a invasão. Por fim, o investigador analisa e correlaciona as evidências encontradas, buscando corroborar suas suspeitas e formular conclusões acerca

da intrusão (indicando causas, responsabilidades e efeitos, por exemplo).

De fato, a experiência do investigador faz-se necessária durante todo o processo de análise forense do sistema invadido. Na etapa de *information gathering* o investigador deve decidir quais informações são relevantes e qual a maneira mais adequada de coletá-las. Durante a busca de evidências, a experiência do investigador permite identificar, entre uma quantidade grande de informações, os indícios possivelmente relacionados com a intrusão. A análise das evidências encontradas é certamente a etapa mais dependente da experiência do investigador. Durante essa etapa, o investigador deve interpretar os vestígios encontrados e, a partir daí, relacioná-los com outras informações (buscando corroborar uma suspeita ou estabelecer uma sequência de eventos, por exemplo), tentar responder alguma questão acerca da invasão (como, por exemplo, sua origem, seu mecanismo de entrada e seus efeitos) e decidir qual o próximo passo a ser tomado (como, por exemplo, buscar uma nova informação ou investigar mais a fundo alguma questão relativa à evidência analisada).

A idéia de automatização está presente em todas as áreas que empregam o uso de computadores. A forense computacional, cujo mister está intimamente relacionado à computação, não haveria de ser diferente. Entretanto, o que se observa atualmente é que apenas a primeira etapa do processo de análise forense (coleta de informações), conta com ferramentas automatizadas [46]. Das ferramentas descritas nos Apêndices A e B, apenas o programa `grave-robber` (componente do conjunto de ferramentas TCT) aproxima-se da idéia de automatização completa da etapa de coleta de informações, pois seu objetivo principal é a coleta automatizada de informações básicas sobre o sistema comprometido, incluindo, por exemplo, os atributos dos arquivos contidos no sistema, dados sobre processos e conexões de rede e o conteúdo de arquivos e diretórios críticos para as etapas seguintes do processo de análise forense. As demais ferramentas, apresentadas nos Apêndices A e B, limitam-se à automatização de algum procedimento específico da etapa de coleta, como, por exemplo, a extração de uma determinada informação do sistema de arquivos. Já as ferramentas como o ForensiX e o EnCase, detalhadas no Apêndice B, apenas integram um conjunto de sub-funções, de modo que a decisão quanto à utilização dessas funcionalidades e a interpretação de seus resultados necessita da intervenção constante do investigador que opera o programa. Poucas ou inexistentes são as ferramentas automatizadas que se destinam especificamente à busca e análise de evidências digitais [46].

Para se automatizar o processo de análise forense é necessário transferir parte do conhecimento do investigador para um sistema automatizado capaz de coletar informações do sistema invadido, identificar e analisar as evidências digitais. A etapa de coleta de informações pode ser traduzida na execução de uma série de ferramentas para a extração de dados voláteis do sistema suspeito e para a geração de imagens de seus discos. Desse

modo, o conhecimento do investigador utilizado nessa etapa pode ser resumido na determinação das informações relevantes e das ferramentas e técnicas utilizadas para sua extração, o que pode ser facilmente codificado em um sistema automatizado de análise forense.

A resposta para a identificação das evidências digitais pode ser encontrada nos sistemas de detecção de intrusão, que utilizam uma série de técnicas (como, por exemplo, detecção por limiar, redes neurais, sistemas especialistas e abordagens baseadas em transição de estados) que permitem "instruir" o IDS a analisar e reconhecer situações intrusivas [3]. A forense computacional pode ser considerada uma instância *post-mortem* de detecção de intrusão, pois um dos objetivos da análise forense, que é realizada após o suposto comprometimento do sistema, é determinar se ocorreu de fato uma invasão. Nesse sentido, os mesmos mecanismos empregados nos sistemas de detecção de intrusão podem ser utilizados em tal sistema automatizado de análise forense, para definir e detectar um conjunto de evidências que descrevem os diversos cenários intrusivos. Esse conjunto de possíveis evidências, representando os vestígios mais prováveis de serem encontrados em um sistema invadido, pode ser definido a partir das experiências anteriores do investigador.

Por fim, o conhecimento do investigador relacionado à etapa de análise das evidências encontradas, que envolve uma série de raciocínios e tomadas de decisões, pode ser transferido para esse sistema automatizado de análise forense através de técnicas de inteligência artificial.

No sentido de viabilizar essa transferência de conhecimento do investigador, permitindo a automatização do processo de análise forense, é apresentada, na sequência, uma arquitetura extensível para um sistema automatizado capaz de coletar informações, identificar e analisar as evidências de uma intrusão. Como resultado da análise forense executada por esse sistema, pode ser gerado um relatório detalhado, contendo a descrição das evidências encontradas, a exposição dos eventos relacionados com a intrusão e a determinação de características particulares do ataque como, por exemplo, origem, serviço explorado e alterações deixadas no sistema comprometido. Além disso, as informações geradas por tal sistema podem ser utilizadas na concepção de uma assinatura do ataque, fornecendo dados para a base de conhecimento de um sistema de detecção de intrusão.

Além do propósito específico de aplicação no modelo de IDS imunológico, descrito no Capítulo 4, a automatização do processo de análise forense possui consequências mais amplas. À medida que a quantidade de informações armazenadas nos sistemas computacionais aumenta, essa automatização torna-se uma necessidade. Muitas vezes uma investigação completa e detalhada é inviabilizada pela quantidade maciça de informações a serem analisadas. Outra consequência relacionada à automatização do processo de análise forense é a possibilidade de implementação de procedimentos e protocolos[1] devidamen-

[1]Maiores detalhes sobre protocolos de análise forense são discutidos no Capítulo 5 e no Apêndice C.

te testados e avaliados, impedindo que o investigador cometa erros que comprometam a investigação.

6.2 Uma arquitetura extensível de análise forense

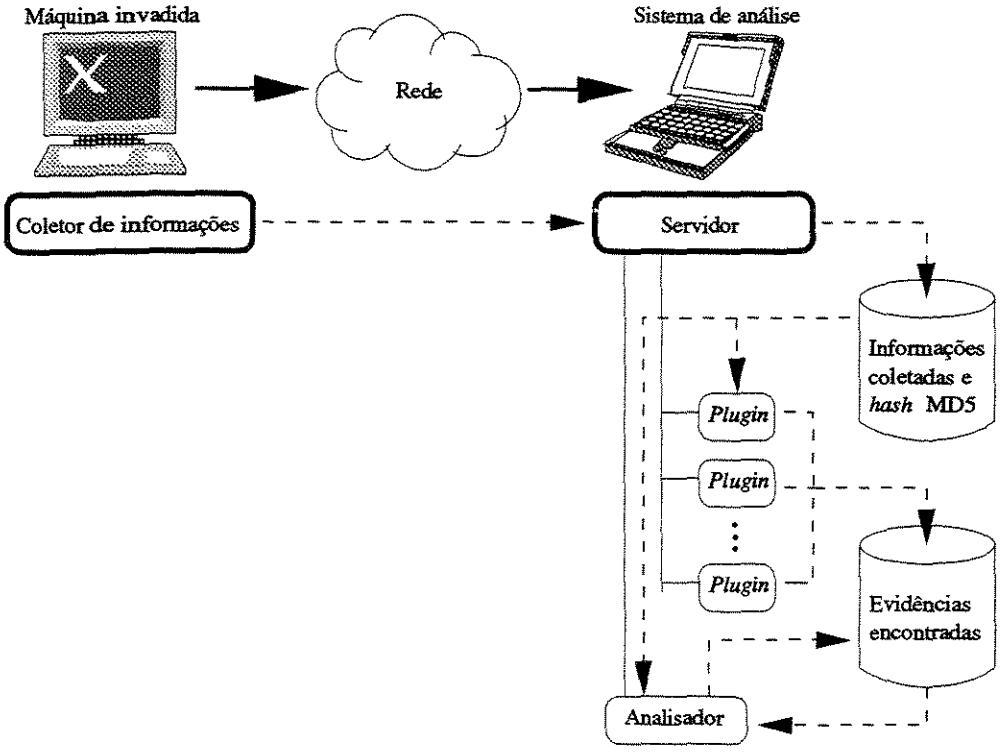


Figura 6.1: Arquitetura extensível para um sistema automatizado de análise forense.

O desenvolvimento de um sistema automatizado de análise forense requer uma arquitetura que permita ao investigador configurar, de acordo com suas experiências de investigações anteriores, todo o processo de análise forense. O investigador deve ser capaz de configurar cada etapa do processo de investigação, incluindo, por exemplo:

- a determinação de quais informações devem ser coletadas para análise e qual a maneira mais adequada de realizar a coleta;
- a definição das informações que podem ser consideradas possíveis evidências de uma intrusão e das técnicas mais adequadas para sua identificação;
- e a determinação de como as evidências podem ser interpretadas e analisadas;

Outra característica importante dessa arquitetura deve ser a consistência com os princípios básicos[2] da investigação forense, incluindo, por exemplo, a preservação da integridade do sistema e dados analisados, a geração de cópias e imagens das informações coletadas, a realização da análise sobre essas cópias e a utilização de ferramentas confiáveis.

Nesse sentido, a Figura 6.1 ilustra uma arquitetura extensível, inicialmente proposta em [81], para o desenvolvimento de um sistema automatizado de análise forense. Tal arquitetura é composta por um servidor, presente no sistema de análise, e um cliente (o coletor de informações) executado na máquina suspeita. O servidor é o componente principal, encarregado de receber as informações provenientes da máquina analisada, organizá-las devidamente no sistema de análise, buscar as evidências e analisá-las. A parte cliente é responsável pela coleta de informações na máquina invadida, enviando-as para o servidor através da rede. Maiores detalhes sobre essa arquitetura são apresentados na sequência, de acordo com as etapas do processo de análise forense.

6.2.1 Coleta de informações

A coleta de informações na máquina suspeita, realizada pelo coletor de informações, consiste na execução de uma série de ferramentas para a extração dos dados contidos nas várias fontes de informações do sistema comprometido. A configuração do coletor de informações pode ser feita com base no apresentado no Capítulo 5 e no Apêndice A, onde são detalhadas as diversas fontes de informações de um sistema computacional e apresentadas as principais ferramentas utilizadas para extração dos dados contidos em cada uma. Nesse sentido, o investigador deve determinar ao coletor de informações quais dados precisam ser coletados e como eles devem ser obtidos, indicando a ferramenta a ser utilizada para a coleta.

Além da coleta de informações voláteis, o procedimento de imagem dos discos da máquina analisada também é realizado nesta etapa, podendo ser executado de duas maneiras. Caso o sistema invadido não possa ser desligado, o coletor de informações pode gerar as imagens, na máquina analisada, e enviá-las para o sistema de análise através da rede. Caso contrário, os discos a serem copiados podem ser instalados no sistema de análise e duplicados diretamente através do servidor.

É importante mencionar que alguns princípios básicos da investigação forense são observados pela arquitetura de análise forense proposta, incluindo, por exemplo:

- a preservação da integridade do sistema analisado — toda informação coletada é enviada, juntamente com seu *hash* criptográfico, para o sistema de análise através da

[2]Maiores detalhes sobre alguns princípios básicos da investigação forense são apresentados no Capítulo 5 e no Apêndice C

rede (utilizando opcionalmente um canal cifrado), de modo que nenhuma informação é salva nos discos da máquina analisada;

- a checagem da autenticidade e integridade das informações coletadas — *hashes* criptográficos são gerados no momento da coleta das informações na máquina analisada, sendo enviados para o sistema de análise, juntamente com os dados coletados, onde são comparados com os *hashes* produzidos a partir das informações recebidas;

- e a utilização de ferramentas confiáveis — toda ferramenta executada no sistema suspeito, incluindo o próprio coletor de informações, os programas auxiliares de coleta e as configurações dessas ferramentas, são acessadas a partir de uma mídia removível e compiladas estaticamente, evitando a utilização de componentes possivelmente comprometidos do sistema suspeito (à exceção do sistema operacional).

6.2.2 Busca e extração de evidências

A etapa de busca e extração de evidências representa uma instância *post-mortem* de detecção de intrusão, sendo equivalente à detecção de intrusão em modo *batch* (descrita no Capítulo 2). Desse modo, as diversas técnicas de análise empregadas na detecção de intrusão podem ser utilizadas na identificação das evidências digitais contidas em um sistema invadido.

Esta etapa é executada na arquitetura proposta através de um conjunto de *plugins*, que buscam evidências em locais específicos do sistema, utilizando as técnicas que melhor se adequam para a detecção de cada evidência. De acordo com o conhecimento do investigador e características particulares do sistema analisado, cada *plugin* é configurado com um conjunto de possíveis evidências que devem ser procuradas. Esse conjunto de possíveis evidências pode ser representado inicialmente por aquelas evidências mais comumente encontradas nas diversas fontes de informações, conforme o apresentado no Capítulo 5. Alguns exemplos dessas possíveis evidências são ilustrados, de maneira genérica, na Figura 6.2.

Diante da inexistência de um método único de detecção que possa ser aplicado com total eficácia em todos os casos [2] e considerando a existência de várias técnicas [3], cada qual com suas vantagens, essa abordagem de *plugins* introduz uma característica extensível à arquitetura de análise forense, permitindo incorporar as diversas soluções existentes e suportar, sem grandes esforços, a adição de novos métodos, através da implementação de novos *plugins*.

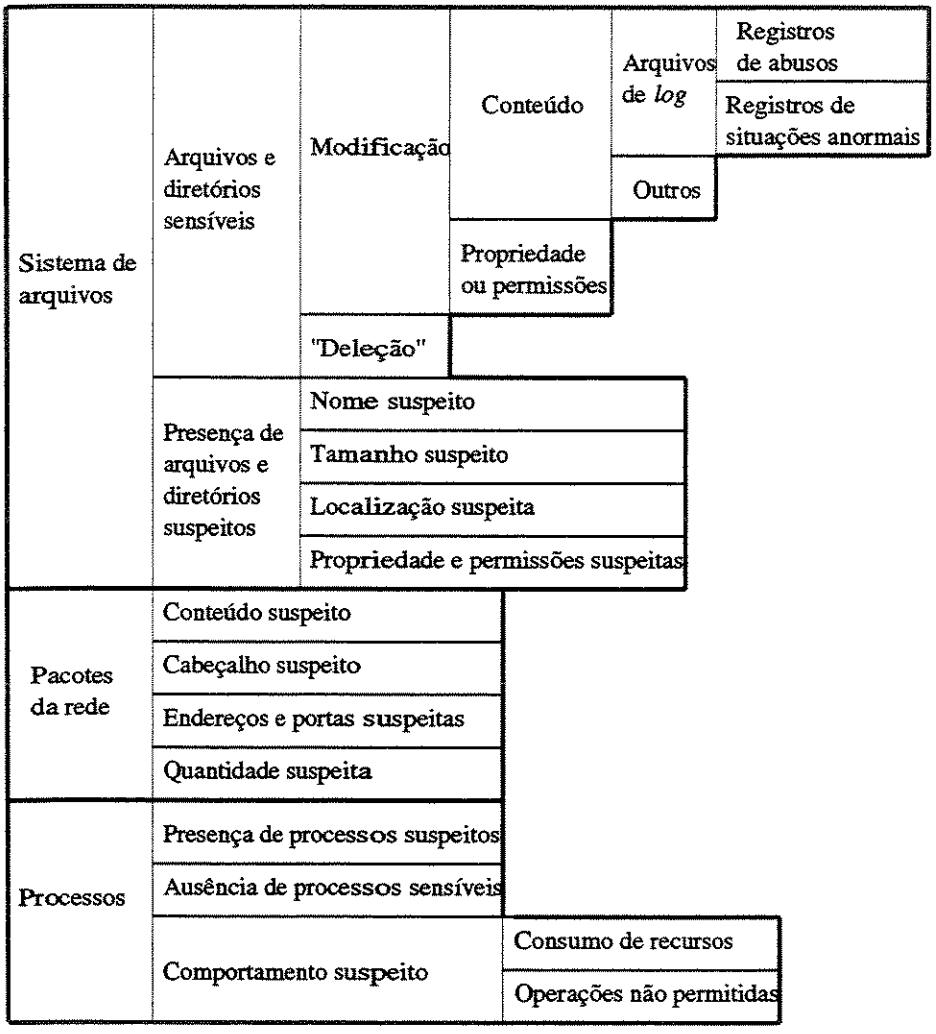


Figura 6.2: Exemplos gerais de possíveis evidências a serem procuradas.

6.2.3 Análise das evidências encontradas

O analisador é o componente da arquitetura proposta responsável pela análise das evidências encontradas pelos *plugins*. Segundo a Figura 6.2, os vestígios encontrados são armazenados em uma base de dados, que é consultada pelo analisador à medida que ele processa cada evidência. Esse processamento envolve a interpretação da evidência (no sentido de se identificar as informações que ela representa), seu correlacionamento com outros vestígios encontrados e outras informações coletadas do sistema suspeito (o que pode causar a identificação de novas evidências ou o descarte do vestígio processado), o estabelecimento de relações entre os vestígios (incluindo, por exemplo, agrupamentos segundo algum parâmetro específico, relações de causa e efeito e ordenação dos eventos segundo uma linha de tempo) e a tentativa de formulação de conclusões acerca da intrusão (como, por exemplo, a determinação da real ocorrência da invasão, a identificação de sua origem, ponto e momento de entrada e consequências no sistema comprometido).

O processamento realizado pelo analisador é baseado nas principais atividades executadas pelo investigador durante a etapa de análise das evidências. Essas atividades podem ser resumidas nos sequintes passos:

1. interpretação dos vestígios encontrados;
2. correlacionamento com outros vestígios e informações, buscando corroborar uma suspeita ou estabelecer uma relação entre os fatos;
3. tentativa de responder alguma questão a respeito da invasão, como, por exemplo, sua origem, seu mecanismo de entrada e seus efeitos;
4. decisão acerca do próximo passo a ser tomado durante a análise, incluindo, por exemplo, a busca de uma nova informação ou a investigação mais aprofundada de alguma questão relativa à evidência analisada;

Esses passos podem ser codificados na arquitetura de análise forense através das seguintes medidas. As evidências encontradas pelos *plugins* podem ser armazenadas e descritas através de uma meta-linguagem que permita a interpretação, por parte do analisador, das informações relacionadas às evidências, como, por exemplo, registros de tempo, números de identificação e o contexto de cada vestígio. A partir dessa descrição, alguns parâmetros podem ser definidos para o correlacionamento das evidências, incluindo, por exemplo, identificação de usuários e grupos, endereços de rede, portas de serviços, identificação de processos, intervalos de tempo e relações de causa e efeito. Com base nos parâmetros definidos, as evidências podem ser agrupadas e organizadas, permitindo relacionar os eventos e ordená-los segundo uma linha de tempo.

Os passos que envolvem raciocínio e tomada de decisões, incluindo a formulação de conclusões sobre a intrusão e o planejamento das atividades de análise, podem necessitar de mecanismos mais complexos que permitam a codificação das capacidades de raciocínio e decisão do investigador. A resposta para tal pode estar nos conceitos e técnicas aplicados na área da inteligência artificial, em particular aqueles relacionados ao planejamento automatizado e ao raciocínio sobre incertezas [65].

A etapa de análise pode ser compreendida como um processo que envolve incertezas, uma vez que o plano de análise, na maioria dos casos, somente é traçado à medida que o processo se desenrola e novas informações são agregadas ao problema. No início da análise das evidências, não se sabe o rumo que ela poderá seguir, nem o conjunto de atividades que precisarão ser executadas e os resultados que cada uma irá produzir. Além disso, as evidências encontradas na etapa anterior representam, na maioria das vezes, informações incertas ou incompletas, de modo que é preciso combinar um conjunto delas para se chegar a alguma conclusão.

Nesse sentido, algumas técnicas de inteligência artificial podem ser sugeridas, como segue, de modo a auxiliar na automatização dos processos decisórios envolvidos na etapa de análise das evidências de uma intrusão.

Raciocínio por evidência (*evidential reasoning*)

O raciocínio por evidência, desenvolvido a partir do trabalho de John Lowrance[3] [59] acerca da teoria de funções de confiança de Dempster e Shafer [20, 90, 91], é uma metodologia para a representação e raciocínio a partir de informações potencialmente incertas, incompletas ou imprecisas [64]. O instituto SRI International[4] é pioneiro na aplicação de raciocínio por evidência na formulação de conclusões, a partir de múltiplas fontes de informações, acerca de situações dinâmicas do mundo real. Entre essas aplicações estão a integração de múltiplos sensores, análise de situações, planejamento de rotas, diagnóstico, monitoramento da execução de planos e controle de processos [65].

Uma base formal [64] e uma arquitetura [62] para a implementação de sistemas automatizados de raciocínio sobre incertezas foram desenvolvidas pela SRI International, envolvendo tanto modelos probabilísticos (como o Bayesiano [40] e o Dempster-Shafer [20, 90, 91]) quanto modelos de possibilidade (como a lógica proposicional e a lógica *fuzzy*) [109]. Todas essas técnicas foram incorporadas em uma única ferramenta de raciocínio por evidência, denominada Gister [63, 60].

A idéia por trás do raciocínio por evidência, tanto do ponto de vista formal quanto

[3]John Lowrance é membro do Centro de Inteligência Artificial do instituto de pesquisa SRI International. Maiores informações podem ser obtidas na URL http://www.ai.sri.com/~lowrance (disponível em janeiro de 2003).

[4]Maiores detalhes sobre o instituto de pesquisa SRI International podem ser encontrados na URL http://www.sri.com (disponível em janeiro de 2003).

prático, pode ser dividida em quatro partes, como segue (detalhados em [65]):

1. definição de um conjunto de espaços proposicionais distintos, cada qual delimitando um conjunto de possíveis situações do domínio em questão. Suponha que a resposta para uma determinada questão A está contida em um conjunto finito Θ_A, denominado de quadro de discernimento. Em outras palavras, cada elemento a_i de Θ_A corresponde a uma possível resposta distinta para a questão A, onde apenas um elemento de Θ_A pode ser verdadeiro em um determinado instante. Nesse contexto, um espaço proposicional A_i, acerca da resposta para a questão A, corresponde a um subconjunto de Θ_A;

2. especificação de interrelacões entre esses espaços proposicionais, denominadas de relações de compatibilidade. Se algo é sabido sobre uma determinada situação A, pode-se tentar tirar proveito dessa informação no sentido de estreitar as possibilidades relacionadas a uma situação B. Para tal, é necessário definir uma relação de compatibilidade entre os dois quadros de discernimento Θ_A e Θ_B. Uma relação de compatibilidade simplesmente descreve quais elementos de ambos os quadros podem ser verdadeiros simultaneamente, ou seja, quais elementos são compatíveis. Os quadros de discernimento e sua relações de compatibilidade podem ser usados para calcular os efeitos de qualquer sequência planejada de ações;

3. representação das evidências (informações incertas) na forma de distribuições de confiança sobre esses espaços proposicionais. Quando uma informação é inconclusiva, probalidades parciais substituem a certeza. Distribuições probabilísticas sobre os espaços proposicionais discernidos por um determinado quadro substituem proposições de valor Booleano. Essas distribuições são chamadas de distribuições em massa. Cada evidência é representada como uma distribuição em massa que distribui uma unidade de confiança sobre os espaços proposicionais discernidos por um quadro;

4. estabelecimento de caminhos para as evidências se moverem através desses espaços proposicionais, por meio de operações sobre evidências (incluindo a fusão de duas distribuições em massa relativas a um mesmo quadro de discernimento e a tradução de uma evidência de uma quadro de discernimento para outro, através das relações de compatibilidade entre eles). Em algum momento as evidências convergem em espaços proposicionais onde as questões alvo podem ser respondidas;

Os passos descritos anteriormente especificam um mecanismo para o questionamento a partir de múltiplas evidências, na direção de uma conclusão particular (probabilística) [65].

Uma abordagem baseada em aprendizado, para o raciocínio por evidência, é utilizado por Terrance Goan no sistema de detecção de intrusão ICE (*Intelligent Correlation of Evidence*)[5], que, entre outras características, é capaz de avaliar o valor relativo de cada evidência no suporte a uma determinada hipótese e tomar decisões com base no correlacionamento de evidências provenientes de diversas fontes independentes (incluindo, por exemplo, ferramentas de exame de configuração, dados de auditoria e *sniffers* de rede) [37].

O sistema ICE suporta essas características através da utilização de raciocínio por evidência baseado em redes Bayesianas e de técnicas associadas de aprendizado [37]. A partir de uma teoria de caso sobre como a execução de um plano de ataque resulta em determinadas atividades, o sistema ICE raciocina sobre as possíveis causas dos efeitos observados dessas atividades. Através da utilização de um conhecimento prévio sobre o estado e dinâmica da rede monitorada (representado por um distribuição de probabilidade condicional), o mecanismo de detecção pode agendar a coleta de evidências e usar observações parciais de ações de usuários e mudanças de estado para avaliar as hipóteses de ataque. A aplicação dessa abordagem permite ao sistema ICE raciocinar sobre evidências incertas ou incompletas, formular estratégias de teste de hipóteses e adequar-se às preferências do operador [37].

Sistemas especialistas

Um sistema especialista encapsula o conhecimento adquirido de um especialista humano e o aplica automaticamente para tomar decisões. Apesar de não poderem lidar com incertezas [3], os sistemas especialistas podem representar uma forma intuitiva de codificar a capacidade decisória do investigador, na forma de regras "se-então". De um lado da regra são especificadas as condições necessárias para a tomada de uma decisão (o lado "se"). Quando essas condições são satisfeitas, as ações definidas do outro lado (a parte "então") são executadas. Como exemplos de regras de decisão, codificadas na sitaxe "se-então", podem ser citadas as seguintes:

- Se achar algum processo com nome suspeito **então** busque portas de serviço relacionadas a tal processo;
- Se achar portas suspeitas **então** busque conexões com tais portas;
- Se achar conexões com portas suspeitas **então** determine endereço de origem;

[5]Maiores informações sobre o sistema ICE podem ser encontradas na URL http://www.shai.com (disponível em janeiro de 2003).

6.3 Um protótipo inicial

Devido à extensão e complexidade do problema de automatização do processo de investigação forense e ao aspecto pioneiro relacionado à análise automatizada de evidências de intrusões, o escopo deste trabalho foi restringido ao tratamento das etapas de coleta de informações e busca de evidências. Nesse sentido, foi desenvolvido um protótipo inicial que implementa a estrutura geral da arquitetura proposta e as duas primeiras etapas do processo de análise forense. A etapa de *information gathering* encontra-se totalmente implementada, enquanto que a etapa de busca de evidências conta com apenas dois *plugins* desenvolvidos, responsáveis pela análise das informações relacionadas aos processos em execução no sistema analisado e ao tráfego de rede capturado.

O protótipo inicial, denominado AFA (*Automated Forensic Analyser*), foi implementado na linguagem Perl e limita-se à análise de intrusões em sistemas Linux. A escolha da linguagem Perl deve-se a sua facilidade em lidar com *strings* e arquivos e a sua popularidade na implementação de ferramentas de administração e segurança de sistemas, permitindo a reutilização de bibliotecas contidas em outras ferramentas (como o TCT, por exemplo).

O AFA é composto por dois *scripts* principais: o `afa_server` e o `gather_target`. De acordo com a arquitetura proposta, ilustrada na Figura 6.1, o *script* `afa_server` implementa o componente servidor, executado no sistema de análise, sendo responsável por receber as informações provenientes da máquina analisada, armazená-las no sistema de análise, buscar as evidências através dos *plugins* e analisá-las (capacidade ainda não disponível). Já o *script* `gather_target` implementa o coletor de informações, que é executado na máquina invadida para a coleta de informações voláteis e imagens dos discos (quando a máquina comprometida não pode ser desligada). Fazem parte do AFA, ainda, os *plugins* responsáveis pela busca de evidências nas informações coletadas (foram implementados os *plugins* `ps_analyser` e `packet_analyser`) e uma série de programas auxiliares necessários à coleta de informações (como, por exemplo, os comando `ps`, `netstat` e `dd`). Maiores detalhes sobre a utilização e configuração dos componentes do AFA são apresentados no Apêndice D.

Apesar de seu estágio embrionário, o AFA permite colocar em prática a arquitetura proposta, de modo a consolidar os conceitos e entender a fundo questões que só aparecem durante a implementação, incluindo, por exemplo, detalhes de configuração, interface e funcionamento. Nenhum teste quantitativo foi realizado, no sentido de avaliar a eficiência e precisão do sistema, devido ao estágio inicial em que se encontra o protótipo.

6.4 Aplicação da arquitetura de análise forense no modelo de IDS imunológico

Certamente a automatização completa do processo de análise forense está longe de ser uma realidade. Entretanto, o esforço inicial apresentado neste capítulo, traduzido na arquitetura proposta e no protótipo desenvolvido, constitui um importante subsídio para o desenvolvimento de um sistema de segurança baseado no modelo de IDS imunológico descrito no Capítulo 4, no sentido de buscar a viabilização da funcionalidade proposta ao componente gerador de assinaturas do referido modelo.

De acordo com a abordagem adotada para a implementação do modelo conceitual de IDS imunológico, a arquitetura de análise forense, aplicada ao gerador de assinaturas, pode necessitar de pequenas adaptações em seu modo de funcionamento e na organização de seus componentes, como exemplificado nas seguintes situações:

- No caso de uma abordagem distribuída, onde cada máquina da rede possui um sistema de segurança imunológico completo (como é o caso do sistema ADenoIdS, apresentado no Capítulo 4), o gerador de assinaturas deve implementar todos os componentes da arquitetura de análise forense. Nesse caso, o funcionamento da arquitetura deve ser local, de modo que o coletor de informações e o servidor são executados na mesma máquina (talvez como módulos separados do gerador de assinaturas), não havendo necessidade de envio de informações pela rede;

- Considerando uma situação centralizada, onde apenas uma máquina é responsável por monitorar e defender toda a rede, cada máquina deve ser capaz de executar o coletor de informações, permitindo que os dados necessários à análise forense possam ser enviados ao servidor, contido no gerador de assinaturas do sistema de defesa centralizado. Nesse caso, cada máquina da rede pode conter um coletor de informações ou pode ser utilizada uma abordagem baseada em agentes móveis[106], que executam o coletor de informações onde e quando for necessário;

No entanto, qualquer que seja a abordagem adotada para o desenvolvimento do sistema de segurança imunológico, é importante observar que os componentes da arquitetura de análise forense (incluindo o coletor de informações, o servidor, os *plugins*, os programas auxiliares de coleta e os arquivos de configuração) podem ser executados em ambientes potencialmente comprometidos, além de não poderem ser acessados a partir de uma mídia removível confiável, como proposto anteriormente, pois idealiza-se um processo totalmente automatizado. Desse modo, é necessário garantir que os componentes envolvidos na análise forense não possam ser comprometidos pelo atacante[6].

[6]Embora sejam implementados mecanismos que aumentem a confiabilidade dos componentes envol-

Os componentes relacionados à etapa de coleta de informações precisam ser executados no sistema suspeito, de modo que eles podem estar contidos previamente na máquina invadida ou em um outro sistema (nesse caso particular, eles podem ser executados, por exemplo, através de uma abordagem baseada em agentes móveis). No caso da primeira abordagem, pode-se aumentar a confiabilidade dos componentes envolvidos na análise forense através da utilização de uma partição de disco com proteção contra escritas, para armazenar tais componentes. Entretanto, essa partição deve estar sempre acessível e a proteção contra escritas só pode ser desabilitada sob certas condições (mediante algum tipo de autenticação, por exemplo). Tais características podem ser implementadas, por exemplo, através de modificações no *kernel* do sistema operacional.

Outra forma de aumentar a confiabilidade da análise forense é a utilização de uma máquina confiável e de acesso restrito, dedicada à execução do servidor da arquitetura de análise forense. Desse modo, apenas os componentes de coleta de informações são executados nas máquinas potencialmente comprometidas e todo o processo de busca e análise das evidências é realizado na máquina confiável. Entretanto, é necessária a implementação de mecanismos de autenticação e retransmissão, para garantir a confiabilidade na comunicação entre o coletor de informações e o servidor.

6.4.1 Aplicação específica no sistema ADenoIdS

Como visto no Capítulo 4, o gerador de assinaturas do sistema ADenoIdS executa uma busca no tráfego de rede capturado, com o intuito de identificar traços de código executável que possam ter gerado um determinado comportamento anormal (possivelmente relacionado a um ataque de *buffer overflow*).

Nesse sentido, o *plugin* `packet_analyser` do AFA (apresentado em detalhes no Apêndice D) pode ser empregado diretamente no desenvolvimento do gerador de assinaturas do sistema ADenoIdS. Esse *plugin* tem por objetivo principal a identificação de pacotes de rede através da busca de padrões em seu conteúdo (definidos por meio de um conjunto de regras).

6.5 Conclusão

Este capítulo discutiu a automatização do processo de análise forense, apresentando, como proposta de solução, uma arquitetura extensível de análise forense e um protótipo inicial que implementada parte dessa arquitetura.

vidos na análise forense, um ataque que comprometa o funcionamento do sistema operacional pode desvirtuar o processo de investigação.

As etapas de coleta de informações e busca de evidências constituem processos onde a automatização pode ser implementada com relativa facilidade. Por outro lado, a análise das evidências representa a etapa onde a automatização torna-se uma questão bastante complexa, uma vez que ela depende totalmente das capacidades de raciocínio e decisão do investigador. Nesse sentido, foi sugerida uma abordagem de solução através de técnicas de inteligência artificial, em particular aquelas relacionadas ao planejamento automatizado e ao raciocínio sobre incertezas, uma vez que as evidências, na maioria dos casos, podem ser caracterizadas como informações incertas ou incompletas.

Capítulo 7

Conclusão

O presente capítulo tece algumas conclusões sobre este trabalho, apresentando uma síntese da pesquisa realizada, algumas considerações finais sobre o trabalho e a experiência adquirida ao longo de seu desenvolvimento.

7.1 Conclusões gerais

Do ponto de vista lógico, este trabalho pode ser dividido em duas partes principais. A primeira parte (representada pelos Capítulos 2, 3 e 4) está relacionada à área da detecção de intrusão e ao paralelo estabelecido entre a segurança de computadores e o sistema imunológico humano. Nessa parte do trabalho, foi apresentada a pesquisa realizada em conjunto com os demais integrantes do grupo de imunologia computacional do LAS-IC-UNICAMP, no sentido do desenvolvimento de um sistema de segurança, baseado no sistema imunológico humano, capaz de detectar e caracterizar um ataque, elaborar um plano de resposta especializado e efetuar o contra-ataque, podendo reagir a ataques desconhecidos e melhorar sua reação a exposições subsequentes de um mesmo ataque. Nesse contexto, foi desenvolvido um modelo conceitual de IDS híbrido, baseado no sistema imunológico humano, que agrega as características desejadas para o referido sistema de segurança.

O modelo de IDS proposto representa uma abordagem inovadora para a detecção de intrusão baseada na analogia com o sistema imunológico, pois ele reconhece e incorpora as características de detecção por mau uso presentes no sistema de defesa humano. Além da estratégia de detecção por anomalia, focalizada em pesquisas anteriores, o modelo proposto integra uma abordagem de detecção por mau uso, permitindo modelar, no nível conceitual, as capacidades de identificação de ataques desconhecidos, aprendizado e reação eficiente a ataques com os quais o sistema já teve contato. Essa integração explora as principais vantagens de cada paradigma de detecção, no sentido de viabilizar as capacida-

des descritas anteriormente. Como visto no Capítulo 2, a detecção por anomalia permite a identificação de cenários intrusivos desconhecidos, enquanto que a detecção por mau uso apresenta-se como uma abordagem eficiente para a detecção de ataques previamente conhecidos e codificados [3].

No entanto, a integração adotada no modelo de IDS proposto é baseada na cooperação ativa entre as duas abordagens de detecção. Nesse sentido, quando um ataque desconhecido é identificado pelo detector de anomalias, um mecanismo de aprendizado é aplicado sobre as informações consideradas anômalas e os rastros deixados no sistema invadido, com o intuito de adquirir conhecimento sobre o ataque e gerar uma assinatura que o identifique. Desse modo, a base de dados do detector de mau uso é atualizada autonomamente, permitindo a identificação futura mais eficiente do mesmo ataque. Além disso, esse mecanismo de aprendizado pode ser utilizado no amparo à hipótese de que a anomalia detectada realmente está associada a um comportamento intrusivo, permitindo reduzir a taxa de falso-positivos relacionada ao detector de anomalias.

O processo de aprendizado e geração automatizada de assinaturas de ataques desconhecidos é executada, no modelo de IDS proposto, através da aplicação de técnicas e ferramentas de análise forense. É nesse contexto em que se insere a segunda parte lógica deste trabalho[1] (representada pelos Capítulos 5 e 6), relacionada à área da forense computacional e sua utilização na identificação e caracterização automatizada de um ataque.

Nessa última parte do trabalho, foi apresentada uma revisão detalhada sobre onde, como e o quê procurar durante a análise forense de um sistema invadido, representando um importante subsídio para o funcionamento do mecanismo de aprendizado do modelo de IDS proposto. Com o intuito de viabilizar a geração de assinaturas de maneira automatizada, foi desenvolvida uma arquitetura de análise forense, capaz, no nível conceitual, de coletar informações do sistema invadido, identificar possíveis evidências relacionadas à intrusão e analisar os vestígios encontrados, visando a caracterização do incidente. É importante observar que a arquitetura desenvolvida possui características extensíveis e incorpora uma série de princípios básicos da investigação forense, necessários ao suporte dos resultados produzidos pela análise.

Do ponto de vista prático, este trabalho apresentou a implementação, na forma do protótipo inicial AFA, de parte da arquitetura de análise forense proposta. O protótipo desenvolvido implementa as funcionalidades de coleta de informações e busca de evidências nos dados coletados, representando as etapas do processo de investigação forense onde se limitou o escopo deste trabalho. A automatização da etapa de análise das evidências encontradas — uma questão de cunho pioneiro na área da forense computacional [46] — foi caracterizada como um problema de planejamento automatizado e raciocínio sobre

[1] Essa segunda parte não contou com a participação direta dos demais integrantes do grupo de imunologia computacional do LAS-IC-UNICAMP.

incertezas, podendo ser abordado através de técnicas aplicadas na área da inteligência artificial. Nesse sentido, foram sugeridas algumas técnicas que podem auxiliar na automatização dos processos decisórios envolvidos nessa etapa da investigação forense. No entanto, é necessário um estudo mais detalhado dessas técnicas e de suas aplicações no contexto da análise forense de intrusões em sistemas computacionais.

Embora a automatização completa do processo de análise forense não seja uma realidade prática, o esforço inicial apresentado neste trabalho representa um passo adiante no sentido de viabilizar a geração automatizada de assinaturas de ataques, podendo ser aplicado, com algumas adaptações e extensões, no desenvolvimento de um sistema de segurança imunológico.

7.2 Considerações finais e experiência adquirida

7.2.1 Forense computacional

A geração automatizada de assinaturas de ataques pode apresentar uma aplicação prática semelhante à adotada por Kephart [52], relacionada à extração de assinaturas de vírus de computador (descrita no Capítulo 3). Assim que um novo ataque é identificado, o mecanismo de análise forense pode ser aplicado em laboratório para gerar sua assinatura, que é posteriormente fornecida aos usuários de um determinado sistema de detecção de intrusão. Nesse sentido, o AFA pode se tornar uma ferramenta bastante útil.

Além do proposto neste trabalho, a automatização do processo de análise forense possui algumas aplicações mais amplas. Através dessa automatização é possível viabilizar análises completas e detalhadas quando a quantidade de informações a ser examinada é extremamente grande. Além disso, a implementação de protocolos e procedimentos devidamente testados e avaliados pode restringir a possibilidade do investigador de cometer erros que comprometam a investigação.

A pesquisa na área da forense computacional leva à conclusão de que o sucesso e eficiência da investigação acerca de um incidente de segurança estão intimamente relacionados aos aspectos de configuração do ambiente envolvido. Com os objetivos de maximizar a utilidade das informações relacionadas ao incidente e minimizar os custos envolvidos na análise desses dados, é importante a adoção de uma política de segurança atenta às necessidades da prática forense [102], incluindo, por exemplo, a implementação de um esquema de *log* seguro e detalhado, a utilização de um mecanismo de sincronização de tempo, a implantação de uma política de resposta a incidentes [67, 47] e a manutenção e treinamento de profissionais capacitados a executar as primeiras medidas no tratamento de um incidente de segurança, com o objetivo de preservar o máximo de informações e proteger os demais sistemas envolvidos.

Além da análise forense para fins corporativos, esta pesquisa também considerou a forense computacional sob o enfoque da persecução criminal, sendo, inclusive, publicado e apresentado um trabalho em congresso específico da área de perícia criminal [80].

Infelizmente, aqueles que cometem crimes não estão alheios à revolução computacional que tem ocorrido nas últimas décadas, de modo que um número crescente de criminosos fazem uso, por exemplo, de *pagers*, telefones celulares, computadores *laptop* e servidores de rede no curso de suas atividades ilícitas. O aumento dramático em crimes relacionados com computadores requer que os operadores de justiça invistam em novas técnicas de abordagem e combate aos crimes, através de treinamento constante e parcerias com entidades técnico-científicas, a fim de se entender como obter e utilizar evidências eletrônicas armazenadas em computadores [49].

A forense computacional constitui um ramo da criminalística bastante recente e relacionado a uma das áreas científicas que mais evolui atualmente, necessitando manter-se atualizada em relação aos desenvolvimentos técnico-científicos, no sentido de se ampliar o uso de evidências digitais no amparo à Justiça.

O estudo apresentado neste trabalho representa um esforço no sentido de suprir essa necessidade de desenvolvimento científico na área da forense computacional. A discussão acerca das várias fontes de informação em um sistema computacional invadido, detalhando as técnicas de extração dos dados e as principais evidências comumente encontradas, fornece um guia prático para aqueles que estão se iniciando na área. De maneira análoga, a estrutura geral apresentada para o processo de investigação forense constitui um ponto de partida para o desenvolvimento de bons procedimentos de análise, podendo ser aplicada a qualquer tipo de investigação envolvendo sistemas computacionais. Por fim, a apresentação de diversas ferramentas, com exemplos práticos de utilização, permite orientar o investigador na escolha e manipulação de seus utilitários de análise.

7.2.2 Imunologia computacional

A analogia entre segurança de computadores e o sistema imunológico humano constitui uma rica fonte de inspiração para o desenvolvimento de novos mecanismos de defesa, sejam algoritmos e técnicas de detecção de intrusão, políticas de segurança mais atentas à existência de falhas ou sistemas completos de segurança. Um sistema de defesa, modelado segundo o sistema imunológico, certamente teria uma noção mais sofisticada de identidade [98], o que a maioria dos administradores de redes e sistemas não possui (poucos são os que conhecem o comportamento usual de seu sistema ou rede de computadores). Embora a analogia seja valorosa, alguns aspectos precisam ser devidamente considerados no desenvolvimento de um sistema de segurança imunológico, incluindo, por exemplo:

- o sistema imunológico garante a sobrevivência de um indivíduo, mas o termo "sobre-

vivência", em um sistema computacional, possui um sentido mais amplo. Garantir a "sobrevivência" de um sistema computacional geralmente implica em garantir confidencialidade, integridade, disponibilidade e corretude [98];

- algumas soluções biológicas podem não ser diretamente aplicáveis em sistemas computacionais [98];
- é possível, através da analogia, herdar características indesejáveis do sistema imunológico (incluindo aspectos falhos ou suposições inapropriadas) [98];
- o sistema imunológico não é capaz de reconhecer todos os antígenos *nonself* em um determinado instante, de modo que a cobertura provida é incompleta. Porém, como cada indivíduo tem um sistema imunológico peculiar, nem todos são vulneráveis aos mesmos patógenos em um mesmo grau. Essa diversidade de sistemas imunológicos garante a sobrevivência da população como um todo. Além disso, a manutenção constante do repertório de receptores dinamiza a capacidade de proteção de um indivíduo particular;
- o processo de seleção negativa não é perfeito, uma vez que as células imaturas podem não ser expostas a todas as proteínas *self* do corpo, gerando doenças de auto-imunidade;

Apesar de existirem vários trabalhos relacionados à analogia, muito ainda pode ser explorado, incluindo, por exemplo, os mecanismos de comunicação entre as células do sistema imunológico, o processo de inflamação, a imunização através de vacinas e soros, a atuação das proteínas complementares, o processo Darwiniano de aperfeiçoamento dos receptores e o papel dos órgãos do sistema linfático.

7.3 Contribuições efetivas do trabalho

Algumas contribuições efetivas desta pesquisa podem ser evidenciadas como segue:

- nova abordagem de pesquisa em torno da analogia entre o sistema imunológico humano e a segurança de computadores. Tal abordagem considera a existência de características de detecção por mau uso no mecanismo de defesa humano;
- proposta de um modelo conceitual de IDS híbrido baseado em características do sistema imunológico humano;
- revisão, definição e documentação de técnicas de forense computacional, fornecendo um guia prático para a extração e análise de evidências em ambientes Linux;

- proposta de uma arquitetura extensível para o desenvolvimento de um sistema automatizado de análise forense;
- implementação inicial da ferramenta AFA;
- publicações científicas [80, 75, 82, 84, 81, 79, 78, 83]

7.4 Trabalhos futuros

Certamente a continuidade deste trabalho envolve a investigação dos aspectos relacionados à automatização da etapa de análise das evidências de uma intrusão. Nesse sentido, é preciso definir a meta-linguagem de representação dos vestígios encontrados e especificar o conjunto de parâmetros de correlacionamento das informações. Além disso, a representação e automatização das capacidades de raciocínio e decisão do investigador constituem aspectos importantes para a continuidade desta pesquisa. Esses aspectos podem ser abordados através de um estudo detalhado de técnicas de inteligência artificial que se adequem ao contexto do problema de automatização do processo de análise forense.

Após a definição do mecanismo de processamento das evidências, é possível modelar o funcionamento do componente analisador, da arquitetura de análise forense proposta, mapeando e integrando as diversas atividades executadas nessa etapa do processo de investigação. Como consequências imediatas dessa modelagem incluem-se a implementação do componente analisador e sua integração no protótipo AFA. Nesse sentido, é preciso adequar o funcionamento dos *plugins* implementados, incorporando a tradução das evidências para o formato da meta-linguagem definida. Além disso, o desenvolvimento de novos *plugins*, idealmente um para cada fonte de informação, também constitui um importante quesito para a extensão das funcionalidades do AFA.

Com a extensão do protótipo AFA, é possível efetuar testes quantitativos de eficiência e precisão, através de análises em laboratório de sistemas invadidos. Uma vez estabilizado o funcionamento da arquitetura de análise forense, ela pode ser integrada em uma instância do modelo conceitual de IDS imunológico, como, por exemplo, o sistema ADenoIdS, que encontra-se em fase de prototipação. Nesse sentido, é necessário o desenvolvimento de um módulo adicional para a arquitetura de análise forense, responsável por converter as informações resultantes da investigação em uma assinatura que caracterize os ataques. Além disso, as adaptações necessárias para essa integração e a implementação de características de segurança devem ser avaliadas em detalhes.

Por fim, o trabalho em torno de técnicas e ferramentas de análise forense deve ser um esforço contínuo, uma vez que elas devem se adequar à constante evolução inerente ao ambiente computacional. Por ser uma disciplina criminalística relativamente recente, a forense computacional clama por padrões e pesquisas científicas, principalmente nos

aspectos relacionados ao acesso a informações cifradas ou protegidas por senhas; ao desenvolvimento de políticas de segurança e resposta a incidentes que maximizem a utilidade das informações coletadas e minimizem o custo de uma investigação; à falta de ferramentas de análise e correlacionamento de evidências; ao desenvolvimento e avaliação de procedimentos e protocolos de análise forense; e ao teste e certificação de ferramentas para a prática forense.

Bibliografia

[1] Good Practices Guide for Computer Based Evidence. Association of Chief Police Officers of England, Wales and Northern Ireland, Junho de 1999. ACPO Crime Committee.

[2] Ross Anderson e Abida Khattak. The Use of Information Retrieval Techniques for Intrusion Detection. Em *First Workshop on the Recent Advances in Intrusion Detection*, Louvain-la-Neuve, Belgium, Setembro de 1998.

[3] Rebecca G. Bace. *Intrusion Detection.* Macmillan Technical Publishing, Indianapolis, IN, 2000.

[4] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, Eugene H. Spafford, e D Zamboni. An Architecture for Intrusion Detection Using Autonomous Agents. Relatório Técnico 98/05, COAST, Purdue University, Junho de 1998.

[5] Justin Balthrop, Stephanie Forrest, e Matthew Glickman. Revisiting LISYS: Parameters and Normal Behavior. Em *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.

[6] Adriano M. Cansian. *Desenvolvimento de um Sistema Adaptativo de Detecção de Intrusos em Redes de Computadores.* Tese de Doutorado, Instituto de Física, Universidade de São Paulo, São Carlos, SP, 1997.

[7] Adriano M. Cansian. Crime na Internet: Conhecendo e Observando o Inimigo. Notas de aula e slides de micro-curso durante o *SSI'2000*: Simpósio Segurança em Informática, São José dos Campos, SP, Outubro de 2000.

[8] Brian Carrier. Performing an Autopsy Examination on FFS and EXT2FS Partition Images: An Introduction to TCTUTILs and the Autopsy Forensic Browser. Disponível *online* em agosto de 2001 na URL http://www.cerias.purdue.edu/homes/carrier/forensics.html.

[9] Eoghan Casey. *Digital Evidence and Computer Crime.* Academic Press, San Diego, California, 2000.

[10] Eoghan Casey, editor. *Handbook of Computer Crime Investigation.* Academic Press, San Diego, California, 2002.

[11] Douglas E. Comer. *Internetworking with TCP/IP*, volume 1. Prentice Hall, Upper Saddle River, New Jersey, 3ª edição, 1995.

[12] C. Cowan, P. Wagle, C. Pu, S. Beattie, e J. Walpole. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. Em *DARPA Information Survivability Conference and Exposition*, Janeiro de 2000.

[13] Dipankar Dasgupta, editor. *Artificial Immune Systems and Their Applications.* Springer-Verlag, 1999.

[14] Dipankar Dasgupta. Immunity-Based Intrusion Detection System: A General Framework. Em *Proceedings of the* 22^{nd} *National Information Systems Security Conference (NISSC)*, Outubro de 1999.

[15] Dipankar Dasgupta e Nii Attoh-Okine. Immunity-Based Systems: A Survey. Em *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, Outubro de 1997.

[16] Dipankar Dasgupta e Stephanie Forrest. Artificial Immune Systems in Industrial Applications. Em *Proceedings of the IPMM: International Conference on Intelligent Processing and Manufacturing Material*, Honolulu, HI, Julho de 1999.

[17] Dipankar Dasgupta, Nivedita Majumdar, e Fernando Nino. Artificial Immune Systems: A Bibliography. Relatório Técnico CS-02-001, Computer Science Division, University of Memphis, Julho de 2002.

[18] Leandro Nunes de Castro e Fernando José Von Zuben. Artificial Immune Systems: Part II A Survey Of Applications. Relatório Técnico DCA-RT 02/00, Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, Campinas, SP, Fevereiro de 2000.

[19] H. Debar, M. Becker, e D. Siboni. A Neural Network Component for an Intrusion Detection System. Em *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 240–250. IEEE Computer Society Press, 1992.

[20] Arthur P. Dempster. A generalization of Bayesian inference. *Journal of the Royal Statistical Society*, 30:205–247, 1968.

[21] D. E. Denning. An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232, Fevereiro de 1987.

[22] Patrik D'haeseleer, Stephanie Forrest, e Paul Helman. An Immunological Approach to Change Detection: Algorithms, Analysis, and Implications. Em *Proceedings of the 1996 IEEE Sysmposium on Security and Privacy.* IEEE Computer Society Press, 1996.

[23] Patrik D'haeseleer, Stephanie Forrest, e Paul Helman. A Distributed Approach to Anomaly Detection. Disponível *online* em setembro de 2002 na URL http://www.cs.unm.edu/ forrest/papers.html, 1997.

[24] M. W. Eichin e J. A. Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. Em *Proceedings of the IEEE Symposium on Research in Computer Security and Privacy*, Los Alamitos, CA, 1989. IEEE Computer Society Press.

[25] H. N. Eisen e K. L. Knight. Nature of the Immune System. *Report of the NIAID Task Force on Immunology*, Setembro de 1998. U.S. Department of Health and Human Services, National Institutes of Health. NIH Publication No. 99-2414.

[26] Alberi Espindula. A Função Pericial do Estado. Disponível *online* em agosto de 2001 na URL http://www.espindula.com.br/artigo1.htm.

[27] Dan Farmer e Wietse Venema. Computer forensics analysis class handouts. Disponível *online* em agosto de 2001 na URL http://www.fish.com/forensics/class.html, Agosto de 1999.

[28] Daniel Farmer e Eugene H. Spafford. The COPS Security Checker System. Em *Proceedings of the Summer USENIX Conference*, pp. 165–170, Anaheim, CA, Junho de 1990.

[29] S. Forrest, A. S. Perelson, L. Allen, e R. Cherukuri. Self-nonself discrimination in a computer. Em *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pp. 202–212, Los Alamos, CA, 1994. IEEE Computer Society Press.

[30] Stephanie Forrest e Steven A. Hofmeyr. Immunology as Information Processing. Em L.A. Segel e I. Cohen, editores, *Design Principles for the Immune System and Other Distributed Autonomous Systems.* Santa Fe Institute Studies in the Sciences of Complexity, Oxford University Press, 2001.

[31] Stephanie Forrest, Steven A. Hofmeyr, e Anil Somayaji. A Sense of Self for UNIX Processes. Em *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pp. 120–128, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[32] Stephanie Forrest, Steven A. Hofmeyr, e Anil Somayaji. Computer Immunology. *Communications of the ACM*, 40(10):88–96, 1997.

[33] Stephanie Forrest, Anil Somayaji, e David H. Ackley. Building Diverse Computer Systems. Em *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems*, pp. 67–72, Los Alamitos, CA, 1997. Computer Society Press.

[34] Simson Garfinkel e Gene Spafford. *Practical UNIX and Internet Security*. O'Reilly & Associates, Sebastopol, California, 2ª edição, 1996.

[35] T. D. Garvey e Teresa F. Lunt. Model-based Intrusion Detection. Em *Proceedings of the* 14^{th} *National Computer Security Conference*, pp. 372–385, Baltimore, MD, Outubro de 1991.

[36] Joseph C. Giarratano. *CLIPS Version 5.1 User's Guide*. NASA, Lyndon B. Johnson Space Center, Information Systems Directorate, Software Technology Branch, Março de 1992.

[37] T. Goan. A Cop on the Beat: Collecting and Appraising Intrusion Evidence. *Communications of ACM*, Julho de 1999.

[38] N. Habra, B. Le Charlier, A. Mounji, e I. Mathieu. ASAX: Software Architecture and Rule-based Language for Universal Audit Trail Analysis. Em *Proceedings of ESORICS'92*, Toulouse, France, Novembro de 1992.

[39] R. Heady, G. Luger, A. Maccabe, e M. Servilla. The Architecture of a Network Level Intrusion Detection System. Relatório técnico, Department of Computer Science, University of New Mexico, Agosto de 1990.

[40] David Heckerman. A Tutorial on Learning Bayesin Networks. Relatório Técnico MSR-TR-95-06, Microsoft Research, Advanced Technology Division, Maicrosoft Corporation, Redmond, WA, Março de 1995.

[41] J. Hochberg, K. Jackson, C. Stallings, J.F. McClary, D. DuBois, e J. Ford. NADIR: An Automated System for Detecting Network Intrusion and Misuse. *Computers & Security*, 12(3):235–248, 1993.

[42] Steven A. Hofmeyr. *A Immunological Model of Distributed Detection and its Application to Computer Security.* Tese de Doutorado, Department of Computer Sciences, University of New Mexico, Abril de 1999.

[43] Steven A. Hofmeyr e Stephanie Forrest. Immunity by Design: An Artificial Immune System. Em *Proceedings of 1999 GECCO: Genetic and Evolutionary Computation Conference*, 1999.

[44] Steven A. Hofmeyr e Stephanie Forrest. Architecture for an Artificial Immune System. *Evolutionary Computation*, 7(1):45–68, 2000.

[45] Steven A. Hofmeyr, Stephanie Forrest, e Anil Somayaji. Intrusion Detection Using Sequences of System Calls. *Journal of Computer Security*, 1998.

[46] Chet Hosmer, John Feldman, e Joe Giordano. Advancing crime scene computer forensic techniques. WetStone Technologies Inc, 2000. Disponível *online* em agosto de 2001 na URL http://wetstonetech.com/crime.htm.

[47] Warren G. Kruse II e Jay G. Heiser. *Computer Forensics: Incident Response Essentials.* Addison-Wesley, Reading, Massachusetts, 2002.

[48] Koral Ilgun. USTAT, A Real-Time Intrusion Detection System for UNIX. Tese de Mestrado, Computer Science Department, University of California, Santa Barbara, CA, Novembro de 1992.

[49] Jorilson S. Rodrigues (Perito Criminal Federal INC/DF). Crimes por Computador. *Perícia Federal*, Ano I(1):17, Março de 1999.

[50] H. S. Javitz e A. Valdes. The SRI IDES Statistical Anomaly Detector. Em *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, Maio de 1991. IEEE Computer Society Press.

[51] Kurt Jensen. *Coloured Petri Nets - Basic Concepts I.* Springer Verlag, New York, 1992.

[52] J. O. Kephart. A Biologically Inspired Immune System for Computers. Em R. A. Brooks e P. Maes, editores, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Sysnthesis and Simulation of Living Systems*, pp. 130–139, Cambridge, MA, 1994. MIT Press.

[53] Jungwon Kim e Peter Bentley. An Artificial Immune Model for Network Intrusion Detection. Em *Proceedings of 7^{th} European Congress on Intelligent Techniques and Soft Computing (EUFIT 99)*, Aachan, Germany, Setembro de 1999.

[54] Jungwon Kim e Peter Bentley. The Human Immune System and Network Intrusion Detection. Em *Proceedings of 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT 99)*, Aachan, Germany, Setembro de 1999.

[55] Sandeep Kumar. *Classification and Detection of Computer Intrusions.* Tese de Doutorado, Department of Computer Sciences, Purdue University, West Lafayette, IN, Agosto de 1995.

[56] Linda Lankewicz e Mark Benard. Real Time Anomaly Detection Using a Nonparametric Pattern Recognition Approach. Em *Proceedings of the Seventh Annual Computer Security Applications Conference*, San Antonio, TX, Dezembro de 1991.

[57] W. Lee, S. J. Stolfo, e K. W. Mok. A Data Mining Framework for Building Intrusion Detection Models. Em *Proceedings of the Twentieth IEEE Symposium on Security and Privacy*, Oakland, CA, 1999. IEEE Computer Society Press.

[58] G. Liepins e H. Vaccaro. Intrusion Detection: Its Role and Validation. *Computers & Security*, 11:347–355, 1992.

[59] John D. Lowrance. *Dependency-Graph Models of Evidential Support.* Tese de Doutorado, Department of Computer and Information Science, University of Massachusetts, Amherst, MA, Setembro de 1982.

[60] John D. Lowrance. *Evidential Reasoning with Gister: A Manual.* Artificial Intelligence Center, SRI International, 333 Ravenswood Avenue, Menlo Park, CA, Abril de 1987.

[61] John D. Lowrance. Automated Argument Construction. *Journal od Statistical Planning and Inference*, 20:369–387, 1988.

[62] John D. Lowrance, Thomas D. Garvey, e Thomas M. Strat. A Framework for Evidential-Reasoning Systems. Em *Proceedings of the National Conference on Artificial Intelligence*, pp. 896–903, Menlo Park, CA, Agosto de 1986. American Association for Artificial Intelligence.

[63] John D. Lowrance, Thomas M. Strat, e Thomas D. Garvey. Application of Artificial Intelligence Techniques to Naval Intelligence Analysis. Relatório Técnico Final, Contrato SRI 6486, SRI International, Artificial Intelligence Center, Menlo Park, CA, Junho de 1986.

[64] John D. Lowrance, Thomas M. Strat, Leonard P. Wesley, Thomas D. Garvey, Enrique H. Ruspini, e David E. Wilkins. The Theory, Implementation, and Practice of

Evidential Reasoning. Relatório Técnico Final, Contrato SRI 5701, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Junho de 1991.

[65] John D. Lowrance e David E. Wilkins. Plan Evaluation under Uncertainity. Em Katia P. Sycara, editor, *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pp. 439–449. Morgan Kaufmann Publishers Inc., San Mateo, CA, Novembro de 1990.

[66] Ludovic Mé. GASSATA, A Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis. Em *First International Workshop on the Recent Advances in Intrusion Detection*, Louvain-la-Neuve, Belgium, Setembro de 1998.

[67] Kevin Mandia e Chris Prosise. *Incident Response: Investigating Computer Crime.* McGraw-Hill, Berkeley, California, 2001.

[68] Marshall K. McKusick, Keith Bostic, Michael J. Karels, e John S. Quarterman. *The Design and Implementation of the 4.4 BSD Operating System.* Addison-Wesley, Reading, Massachusetts, 1996.

[69] Mounji. Languages and Tools for Rule-Based Distributed Intrusion Detection. Tese de Mestrado, Faculte's Universitaires Notre-Dame de la Paix, Namur, Belgium, Setembro de 1997.

[70] P. G. Neumann e Phillip Porras. Experience with EMERALD to Date. Em *First USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, CA, Abril de 1999.

[71] Understanding Autoimmune Diseases. NIH Publication No. 98-4273, Maio de 1998. U.S. Department of Health and Human Services, National Institute of Allergy and Infectious Diseases.

[72] Understanding Vaccines. NIH Publication No. 98-4219, Janeiro de 1998. U.S. Department of Health and Human Services, National Institute of Allergy and Infectious Diseases.

[73] Michael G. Noblett, Mark M. Pollitt, e Lawrence A. Presley. Recovering and Examining Computer Forensic Evidence. *Forensic Science Communications*, 2(4), Outubro de 2000. U.S. Department of Justice, FBI.

[74] Scientific Working Group on Digital Evidence (SWGDE) e International Organization on Digital Evidence (IOCE). Digital Evidence: Standards and Principles. *Forensic Science Communications*, 2(2), Abril de 2000. U.S. Department of Justice, FBI.

[75] Fabrício S. Paula, Marcelo A. Reis, Diego M. Fernandes, e Paulo L. Geus. ADenoIdS: A Hybrid IDS Based on the Immune System. Em *Proceedings of the ICONIP2002: 9th International Conference on Neural Information Processing, Special Session on Artificial Immune Systems and Their Applications*, Cingapura, Novembro de 2002.

[76] Phillip Porras. STAT, A State Transition Analysis Tool for Intrusion Detection. Tese de Mestrado, Computer Science Department, University of California, Santa Barbara, CA, Julho de 1992.

[77] Phillip Porras e P. G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. Em *Proceedings of the Twentieth National Information System Security Conference*, Baltimore, MD, 1997.

[78] Marcelo A. Reis, Diego M. Fernades, Fabrício S. Paula, e Paulo L. Geus. Modelagem de um Sistema de Segurança Imunológico. Em *Anais do SSI2001: 3º Simpósio Segurança em Informática*, pp. 91–100, Instituto Tecnológico de Aeronáutica-ITA, São José dos Campos, SP, Outubro de 2001.

[79] Marcelo A. Reis e Paulo L. Geus. Forense Computacional: Procedimentos e Padrões. Em *Anais do SSI2001: 3º Simpósio Segurança em Informática*, pp. 73–81, Instituto Tecnológico de Aeronáutica-ITA, São José dos Campos, SP, Outubro de 2001.

[80] Marcelo A. Reis e Paulo L. Geus. Análise Forense de Intrusões em Sistemas Computacionais: Técnicas, Procedimentos e Ferramentas. Em *Anais do I Seminário Nacional de Perícia em Crimes de Informática e IV Seminário Nacional de Fonética Forense*, Maceió, AL, Novembro de 2002. Associação Brasileira de Criminalística.

[81] Marcelo A. Reis e Paulo L. Geus. Modelagem de um Sistema Automatizado de Análise Forense: Arquitetura Extensível e Protótipo Inicial. Em *Anais do Wseg2002: Workshop em Segurança de Sistemas Computacionais*, Búzios, RJ, Maio de 2002. Workshop realizado durante o SBRC2002: Simpósio Brasileiro de Redes de Computadores.

[82] Marcelo A. Reis e Paulo L. Geus. Standardization of Computer Forensic Procedures and Protocols. Em *Proceedings of the 14th Annual Computer Security Incident Handling Conference*, Waikoloa, HI, USA, Junho de 2002. FIRST.Org, Inc.

[83] Marcelo A. Reis, Flávio S. Oliveira, Célio C. Guimarães, e Paulo L. Geus. Forense Computacional: Aspectos Legais e Padronização. Em *Anais do Wseg2001: Workshop em Segurança de Sistemas Computacionais*, pp. 80–85, Florianópolis, SC, Março de 2001. Workshop realizado durante o SCTF2001: IX Simpósio de Computação Tolerante a Falhas.

[84] Marcelo A. Reis, Fabrício S. Paula, Diego M. Fernandes, e Paulo L. Geus. A Hybrid IDS Architecture Based on the Immune System. Em *Anais do Wseg2002: Workshop em Segurança de Sistemas Computacionais*, Búzios, RJ, Maio de 2002. Workshop realizado durante o SBRC2002: Simpósio Brasileiro de Redes de Computadores.

[85] I. Roitt, J. Brostoff, e D. Male. *Immunology.* Mosby, 4ª edição, 1996.

[86] Deborah Russel e G. T. Gangemi Sr. *Computer Security Basics.* O'Reilly & Associates, Sebastopol, California, Dezembro de 1991.

[87] Tony Sammes e Brian Jenkinson. *Forensic Computing: A Practitioner's Guide.* Springer, London, UK, 2000.

[88] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, New York, 1996.

[89] M. M. Sebring, E. Shellhouse, M. E. Hanna, e R. A. Whitehurst. Expert Systems in Intrusion Detection: A Case Study. Em *Proceedings of the* 11^{th} *National Computer Security Conference*, pp. 74–81, Baltimore, Maryland, Outubro de 1988. NIST/NCSC.

[90] Glenn Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, Princeton, NJ, 1976.

[91] Glenn Shafer. Belief functions and possibility measures. *The Analysis of Fuzzy Information*, 1, 1986.

[92] A. Silberschatz e P. Galvin. *Operating System Concepts.* John Wiley & Sons, New York, 5ª edição, 1998.

[93] Stephen E. Smaha. Haystack: An Intrusion Detection System. Em *Proceedings of the Fourth IEEE Aerospace Computer Security Applications Conference*, Orlando, FL, Dezembro de 1988. IEEE Computer Society Press.

[94] Stephen E. Smaha. A Common Audit Trail Interchange Format for UNIX. Relatório técnico, Haystack Laboratories, Inc., Austin, TX, Outubro de 1994.

[95] Stephen E. Smaha e S. Snapp. Method and System for Detecting Intrusion into and Misuse of a Data Processing System. US555742, U.S. Patent Office, Setembro de 1996.

[96] B. Snow. Future of Security. Panel presentation at IEEE Security and Privacy, Maio de 1999.

[97] Anil Somayaji e Stephanie Forrest. Automated Response Using System-Call Delays. Em *Proceedings of the Ninth USENIX Security Symposium*, Denver, CO, Agosto de 2000.

[98] Anil Somayaji, Steven A. Hofmeyr, e Stephanie Forrest. Principles of a Computer Immune System. Em *Proceedings of the 1997 New Security Paradigms Workshop*, pp. 75–82, Langdale, Cumbria UK, 1997. ACM Press.

[99] W. Stallings. *Computer Organization and Architecture: Designing for Performance.* Prentice Hall, Upper Saddle River, New Jersey, 5ª edição, 2000.

[100] Peter Stephenson. *Investigating Computer-Related Crime.* CRC Press, Boca Raton, Florida, 2000.

[101] W. Richard Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, Reading, Massachusetts, 2ª edição, 1994.

[102] John Tan. Forensic Readiness. @state, Inc., Cambridge, MA, Julho de 2001.

[103] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, Upper Saddle River, New Jersey, 3ª edição, 1996.

[104] H. S. Teng, K. Chen, e S. Lu. Adaptive Real-Time Anomaly Detection Using Inductively Generated Sequential Patterns. Em *Proceedings of the IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, Maio de 1990.

[105] Computer Immune Systems. Disponível *online* em agosto de 2002 na URL http://www.cs.unm.edu/ immsec/research.htm. Computer Science Department, University of New Mexico.

[106] Nelson Uto e Ricardo Dahab. Segurança de sistemas de agentes móveis. Em *Anais do SSI2001: 3º Simpósio Segurança em Informática*, pp. 211–220, Instituto Tecnológico de Aeronáutica-ITA, São José dos Campos, SP, Outubro de 2001.

[107] Wietse Venema. Murphy's law and computer security. Em *Proceedings of the Sixth USENIX Security Symposium*, San Jose, Julho de 1996.

[108] Christina Warrender, Stephanie Forrest, e Barak Pearlmutter. Detecting Intrusions Using System Calls: Alternative Data Models. Em *Proceedings of the 1999 IEEE Sysmposium on Security and Privacy.* IEEE Computer Society Press, Maio de 1999. verificar se foi peblicado.

[109] David E. Wilkins, Karen L. Myers, John D. Lowrance, e Leonard P. Wesley. Planning and Reacting in Uncertain and Dynamic Environments. *Journal of Experimental and Theoretical AI*, 7(1):197–227, 1995.

Apêndice A

Detalhes particulares sobre a extração e análise das informações de um sistema computacional

Este apêndice apresenta maiores detalhes sobre a extração e análise dos dados contidos nas diversas fontes de informação descritas no Capítulo 5.

As técnicas apresentadas neste apêndice apenas ilustram como determinadas informações podem ser obtidas na máquina analisada, não havendo preocupação com o destino da saída dos comandos apresentados. Maiores detalhes sobre o processo de coleta de informações são discutidos na Capítulo 5.

Grande parte das ferramentas utilizadas nas diversas explanações e apresentadas nos exemplos são utilitários presentes na maioria das distribuições do sistema Linux. Aquelas ferramentas não distribuídas juntamente com o sistema Linux apresentam referências aos locais de obtenção.

A.1 Extração e reprodução dos dados da memória de vídeo

O comando `xwd`, do sistema X Window, pode ser usado para capturar uma janela particular ou toda a tela. O comando `xwd` necessita de acesso de *root* e deve ser executado a partir de outro terminal virtual ou remotamente, para não alterar a tela que se deseja capturar:

```
# xwd -display localhost:0 -root > screen.xwd
```

A opção `-display` serve para identificar a máquina e o número do terminal gráfico de onde se deseja coletar a imagem gráfica e a opção `-root` especifica que toda a tela deve ser capturada. Maiores detalhes sobre as opções do comando `xwd` podem ser encontrados na respectiva *man page*. O comando `xwd` gera sua saída em um formato especial (XWD), que pode ser salvo em um arquivo e visualizado através do utilitário `xwud`:

```
# xwud -in screen.xwd
```

A opção `-in` serve para especificar o arquivo contendo a saída do comando `xwd`. Vários utilitários, como `fbm`, `pbmplus` e `ImageMagick`, podem ser utilizados para converter o formato XWD para outros mais comuns, como TIFF e GIF [47].

A.2 Acesso às informações da memória principal do sistema

A.2.1 Processo de *dump* da memória e análise dos dados extraídos

O processo de *dump* da memória pode ser realizado através do comando `dd`, como segue:

```
# dd if=/dev/mem of=mem.dump bs=1024
# dd if=/dev/kmem of=kmem.dump bs=1024
```

O programa `dd` é um utilitário de cópia bastante útil para o processo de análise forense, pois permite copiar qualquer fluxo de bits. Maiores detalhes sobre suas opções podem ser obtidos na *man page* do comando. É importante lembrar que a memória principal do computador e a memória do *kernel* são acessíveis, no sistema Linux, através dos arquivos de dispositivo (*device files*) `/dev/mem` e `/dev/kmem`, respectivamente.

A execução do procedimento de *dump* causa a alteração de uma parte da memória, justamente onde o utilitário usado é alocado para execução [67]. Desse modo, não é possível verificar se as informações capturadas são exatamente iguais às originais [47].

A reconstrução completa das informações capturadas da memória requer um conhecimento altamente especializado [47], entretanto, a busca de palavras-chave pode ser uma alternativa mais simples e viável, podendo ser realizada através dos comandos `grep` e `strings` :

```
# grep palavra-chave mem.dump
# strings -a mem.dump | grep palavra-chave
# strings -a mem.dump | more
```

Executados sobre um conjunto de bits qualquer, o comando `grep` permite a busca de padrões nesse conjunto e o utilitário `strings` extrai as cadeias de caracteres ASCII nele contidas (é importante utilizar a opção `-a` do comando `strings`, para permitir que todo o conjunto de bits seja analisado). Detalhes sobre as opções dos comandos podem ser encontrados nas respectivas *man pages*.

A.2.2 Geração e análise de *core files*

O *core file* de um processo em execução pode ser gerado através do comando `kill`, indicando-se o sinal adequado a ser enviado (SIGQUIT, SIGILL, SIGTRAP, SIGFPE, SIGBUS, SIGSEGV ou SIGSYS) e o número de identificação do processo (PID), como ilustrado a seguir:

```
# kill -s SIGQUIT pid
```

Essa abordagem de envio de sinal só terá êxito se o comando `kill` for executado pelo mesmo usuário dono do processo (ou algum usuário com acesso de *root*), se o usuário dono do processo tiver permissão de escrita no diretório onde se encontra o respectivo arquivo executável, se o processo não redefiniu a ação a ser tomada ao receber o sinal e se o tamanho do *core file* não exceder o limite máximo imposto para o usuário dono do processo [34].

O comando `file` pode ser usado, como no exemplo a seguir, para determinar o programa relacionado ao *core file*, bem como o sinal que causou sua geração [47]:

```
# file core
core: ELF 32-bit LSB core file of 'teste' (signal 11), Intel 80386
```

No exemplo anterior, o *core dump* originou-se do programa `teste`, que teve sua execução interrompida pelo sinal 11 (SIGSEGV), indicando uma possível falha de segmentação. Pode-se ainda usar o comando `strings`, sobre um *core file*, para facilitar a visualização do conteúdo do espaço de memória do processo relacionado, permitindo identificar, por exemplo, os arquivos que o programa estava referenciando. Além disso, uma análise mais profunda pode ser executada com a ajuda de programas de depuração[1], como o `gdb` [34, 47]:

```
# gdb -c core
```

[1]Caso o programa relacionado ao *core file* tenha sido compilado com informações de depuração.

A.3 Captura e análise do tráfego de rede

Existem vários programas que podem ser usados para capturar o tráfego de rede, comumente denominados de *sniffers*. Além de capturar os pacotes que trafegam na rede, os *sniffers* podem decodificá-los e exibi-los em um formato mais legível, ou ainda executar operações mais complexas como reconstrução de sessão e recuperação de arquivos transferidos pela rede [10].

Talvez o exemplo mais comum desses programas seja o `tcpdump`, que pode ser usado para capturar todo tipo de tráfego de rede, decodificando e exibindo os pacotes à medida que eles são coletados (no caso de uma análise em tempo real) ou armazenando-os em um arquivo binário para uma análise posterior (através do próprio `tcpdump` ou de outros aplicativos). A segunda abordagem permite fazer uma cópia exata das informações que trafegam na rede, sendo a mais indicada no caso de uma análise em tempo real não ser necessária [10]. Alguns exemplos de utilização do `tcpdump` são apresentados na sequência:

```
# tcpdump -l -n -e -x -vv -s 1500
# tcpdump -w net.dump
# tcpdump -n -e -x -vv -s 1500 -r net.dump
```

No primeiro exemplo, o `tcpdump` é usado para capturar e exibir os pacotes à medida que eles são coletados. A opção `-l` permite a visualização imediata da saída à medida que ela é produzida, a opção `-n` impede a conversão de endereços em nomes (uma consulta DNS reversa pode alertar um atacante experiente [47]), a opção `-e` imprime o cabeçalho da camada de enlace, a opção `-x` imprime o conteúdo dos pacotes no formato hexadecimal, a opção `-vv` permite a visualização de informações extras e a opção `-s` determina o número de bytes de dados que devem ser capturados de cada pacote (o padrão é 68 bytes). No segundo exemplo, o `tcpdump` é usado para armazenar o tráfego de rede em um arquivo binário (`net.dump`), através da opção `-w`. Tal arquivo pode ser processado posteriormente, como no terceiro exemplo, através da opção `-r`.

O `tcpdump` possui, ainda, um conjunto de filtros que permitem selecionar os pacotes que devem ser capturados:

```
# tcpdump -l -n -e -x -vv -s 1500 host 192.168.1.3
# tcpdump -l -n -e -x -vv -s 1500 dst port 22
```

No primeiro exemplo, somente serão capturados os pacotes provenientes do endereço IP 192.168.1.3 ou que tenham o mesmo como destino. Já no segundo exemplo, será capturado todo tráfego destinado à porta 22. Maiores detalhes sobre as opções e regras de filtragem do `tcpdump` podem ser encontrados na *man page* do mesmo.

Além dos *sniffers*, alguns sistemas de detecção de intrusão podem ser usados para capturar e analisar o tráfego de rede, permitindo, por exemplo, inspecionar o tráfego e armazenar apenas os dados que pareçam suspeitos [10].

O Snort[2] é um programa que pode ser usado tanto como *sniffer* quanto como sistema de detecção de intrusão. Ele é capaz de inspecionar a área de dados de um pacote, decodificar suas diversas camadas e compará-lo com uma lista de regras. Desse modo, o Snort pode ser configurado com regras para detectar certos tipos de pacotes, inclusive aqueles relacionados com atividades hostis como, por exemplo, varredura de portas e ataques de *buffer overflows* [10]. Além disso, o Snort possui a capacidade de remontar os pacotes fragmentados, antes de compará-los com assinaturas de ataques conhecidos. Alguns exemplos de utilização do Snort são apresentados como segue:

```
# snort -v -d -e
# snort -d -l log_dir -c snort.conf
# snort -l log_dir -b
# snort -v -d -e -r net.dump
# snort -d -l log_dir -c snort.conf -r net.dump
```

No primeiro exemplo, o Snort é executado no modo *sniffer*, através da opção `-v`. As opções `-d` e `-e` instruem o Snort a exibir, respectivamente, a área de dados do datagrama e o cabeçalho da camada de enlace. No segundo exemplo, o Snort é executado como sistema de detecção de intrusão, através da opção `-c` que indica o arquivo de configuração do programa. No terceiro exemplo, o Snort é utilizado para armazenar o tráfego de rede em um arquivo binário (no formato do `tcpdump`), localizado no diretório `log_dir`. Tal arquivo pode ser processado posteriormente, através da opção `-r`, tanto no modo *sniffer* (quarto exemplo), quanto no modo de detecção de intrusão (quinto exemplo). Maiores informações sobre as opções e regras de detecção do Snort podem ser encontradas na *man page* do programa.

Antes de se efetuar a coleta do tráfego de rede, algumas questões fundamentais devem ser consideradas, incluindo, por exemplo:

- a localização do *sniffer*;
- o tipo de tráfego que deve ser coletado — se todo ele ou apenas um conjunto específico;
- e o momento da análise — se os pacotes devem ser decodificados e analisados à medida que são coletados ou se devem ser armazenados para análise posterior;

[2]Maiores informações sobre o Snort podem ser encontradas na URL http://www.snort.org (disponível em janeiro de 2003).

De maneira geral, o *sniffer* deve ser colocado em um ponto da rede que tenha acesso ao tráfego relacionado à máquina invadida. Entretanto, o atacante pode ter comprometido diversas máquinas da rede e a investigação busca determinar a extensão desse comprometimento. Nesse caso, o *sniffer* deve ser colocado em um ponto onde possa monitorar todo tráfego da rede, considerando sua topologia e as configurações dos elementos comutadores, como *hubs* e *switches*.

Geralmente, quanto maiores a quantidade de dados coletados e o número de operações realizadas sobre os pacotes (decodificação, exibição, remontagem ou checagem de assinaturas de ataques, por exemplo), maiores serão a carga sobre o sistema de coleta e as chances de que alguns pacotes sejam descartados [10]. Uma filtragem específica do tráfego de rede e a armazenagem dos pacotes, sem decodificação, para análise futura, podem reduzir a perda de informações. Em alguns casos, a análise do tráfego de rede necessita ser feita em tempo real, exigindo a decodificação e exibição dos pacotes à medida que eles são capturados. Entretanto, a prática mais recomendada, no caso de uma análise em tempo real não ser necessária, é a captura dos pacotes em seu estado original, no formato binário [10].

A análise do tráfego de rede capturado geralmente é feita com o auxílio de ferramentas que reconstroem e exibem as informações em um formato mais adequado. Algumas ferramentas como, por exemplo, o Ethereal[3] e o Review [10], permitem a reprodução da sessão capturada, além da visualização das informações de forma mais simples que a oferecida pelo `tcpdump`. Outra funcionalidade presente em algumas ferramentas (no Review por exemplo) é a reconstrução de arquivos que foram transferidos durante a sessão capturada, permitindo a recuperação de todo tipo de informação transferida pelo atacante (incluindo, por exemplo, suas ferramentas e possíveis dados roubados) [10]. Os sistemas de detecção de intrusão também são ferramentas úteis na análise do tráfego de rede, pois permitem automatizar o processo de reconhecimento de evidências da intrusão.

A.4 Acesso às informações dos processos em execução

A.4.1 Atividade do sistema

O comando `uptime` pode ser usado para determinar a atividade atual e recente do sistema, fornecendo informações sobre a quantidade de tempo em que o sistema encontra-se em execução, além de dados sobre as médias de carga de processamento para os últimos 1, 5 e 15 minutos. Um exemplo de execução do comando `uptime` é apresentado como segue:

[3]Maiores informações sobre o Ethereal podem ser encontradas na URL http://www.ethereal.com (disponível em janeiro de 2003).

```
# uptime
9:58am  up  1:55,  3 users,  load average: 0.14, 0.03, 0.01
```

No exemplo anterior, o comando `uptime` foi executado às 9:58 da manhã, informando que o sistema estava em execução há uma hora e cinquenta e cinco minutos e que havia três usuários "logados", além das médias de carga de processamento referentes aos últimos 1, 5 e 15 minutos (nessa ordem).

A.4.2 Listagem e detalhes dos processos

Uma lista de todos os processos em execução, com detalhes sobre o contexto e estado de cada um, pode ser obtida através do comando `ps`:

```
# ps -auxeww
```

O comando `ps` possui uma grande quantidade de opções que variam bastante entre as diferentes plataformas UNIX. No ambiente Linux, o exemplo anterior provê uma lista de todos os processos (inclusive aqueles sem terminal de controle), contendo, entre outras informações, o nome do programa relacionado, o número de identificação do processo, seu usuário dono e o tempo de início da execução. Maiores detalhes sobre a saída do comando `ps` e suas opções podem ser encontrados na *man page* do mesmo ou em [34]. Uma lista, com atualizações em tempo real, dos processos que mais consomem CPU pode ser obtida através do comando `top`.

A.4.3 Arquivos acessados

Outra informação importante relacionada aos processos refere-se os arquivos acessados por cada um. O comando `lsof` provê uma lista de todos os arquivos acessados por cada processo em execução, como ilustrado a seguir (a opção `-Dr` pode ser necessária em algumas plataformas UNIX, para evitar que o comando `lsof` escreva no disco da máquina analisada [67]):

```
# lsof
COMMAND  PID USER   FD   TYPE DEVICE    SIZE    NODE NAME
syslogd  550 root  txt    REG    3,2   26876  519522 /sbin/syslogd
syslogd  550 root   1w    REG    3,5 6505198   12259 /var/log/messages
syslogd  550 root   2w    REG    3,5  127747   12245 /var/log/secure
sshd     611 root  txt    REG    3,2  232412   33315 /usr/sbin/sshd
sshd     611 root  mem    REG    3,2  441668  243478 /lib/ld-2.2.2.so
sshd     611 root  mem    REG    3,2   35352  243560 /lib/libpam.so.0.74
```

```
sshd     611 root  mem   REG    3,2 5578134  243487 /lib/libc-2.2.2.so
sshd     611 root   0u   CHR    1,3          129949 /dev/null
sshd     611 root   3u  IPv4    822             TCP *:ssh (LISTEN)
```

O exemplo anterior contém apenas alguns fragmentos da listagem original, permitindo observar alguns dos arquivos regulares, *device files*, bibliotecas dinâmicas e *sockets* acessados pelos processos `syslogd` e `sshd`.

A.4.4 Interface `/proc`

Outra forma de acessar as informações relacionadas aos processos em execução é através da interface provida pelo diretório `/proc`. O diretório `/proc` é um pseudo sistema de arquivos usado como uma interface para as estuturas de dados do *kernel* do sistema operacional, permitindo uma visão do ambiente de cada processo em execução. Cada processo na memória possui um sub-diretório correspondente em `/proc`, cujo nome é o número de identificação do processo. O exemplo seguinte ilustra o conteúdo do sub-diretório correspondente ao processo `/root/teste`, cujo PID é 944:

```
# /root/teste -d

# ps -aux | grep /root/teste
USER  PID  %CPU  %MEM  VSZ   RSS  TTY    STAT  START  TIME  COMMAND
root  944  98.8  0.4   1300  292  pts/1  R     09:29  0:29  /root/teste -d

# ls -la /proc/944
total 0
dr-xr-xr-x    3 root   root    0 May 14 09:39 .
dr-xr-xr-x   55 root   root    0 May 14 04:28 ..
-r--r--r--    1 root   root    0 May 14 09:39 cmdline
lrwxrwxrwx    1 root   root    0 May 14 09:39 cwd -> /
-r--------    1 root   root    0 May 14 09:39 environ
lrwxrwxrwx    1 root   root    0 May 14 09:39 exe -> /root/teste
dr-x------    2 root   root    0 May 14 09:39 fd
-r--r--r--    1 root   root    0 May 14 09:39 maps
-rw-------    1 root   root    0 May 14 09:39 mem
lrwxrwxrwx    1 root   root    0 May 14 09:39 root -> /
-r--r--r--    1 root   root    0 May 14 09:39 stat
-r--r--r--    1 root   root    0 May 14 09:39 statm
-r--r--r--    1 root   root    0 May 14 09:39 status
```

Dentro do sub-diretório relacionado a cada processo, em `/proc`, existem informações bastante úteis para o processo de análise forense, incluindo o arquivo `cmdline`, o *link* simbólico `exe` e o diretório `fd`. O arquivo `cmdline` contém a linha de comando completa usada para executar o programa:

```
# cat /proc/944/cmdline
/root/teste-d
```

O conteúdo do arquivo `cmdline` é normalmente utilizado pelo comando `ps` para exibir o nome do programa relacionado ao processo [67]. Um atacante pode facilmente alterar a linha de comando através do código do programa[4], com o intuito de "camuflar" o processo na saída do comando `ps` [67].

O arquivo `exe` é um *link* simbólico para o programa que foi executado. Um atacante pode iniciar a execução de um programa e "deletá-lo" em seguida, com o intuito de esconder seus rastros no sistema. Entretanto, o atacante não está verdadeiramente "deletando" o programa. O sistema Linux mantém para cada arquivo um contador, chamado de *link count*, que representa o número de processos usando o arquivo. Quando o *link count* iguala-se a zero, nenhum processo está usando o arquivo e o mesmo será "deletado". Quando um atacante executa e "deleta" seu programa, este será removido da cadeia de diretórios (não será mais visível através do comando `ls`), o *link count* do programa será decrementado em uma unidade e a marca de tempo de "deleção" receberá o horário corrente. Contudo, o *link count* não se igualará a zero até que o processo do atacante termine. Os arquivos com *link count* igual a zero são denominados de *unlinked files* e estão marcados para "deleção" quando o sistema for desligado [67].

É de grande importância a recuperação desses arquivos marcados para "deleção", pois após o desligamento do sistema, a recuperação dos arquivos "deletados" torna-se um processo mais complexo [67]. O exemplo seguinte mostra o procedimento usado por muitos atacantes para limpar seus rastros:

```
# /root/teste -d &
[1] 1094

# rm -f /root/teste

# ls -la /proc/1094
total 0
```

[4]Em um programa escrito em código C, basta copiar o nome desejado na primeira posição do vetor `argv`, passado como parâmetro para a função principal `main` (através da função `strcpy(argv[0], ''nome'')`, por exemplo).

```
dr-xr-xr-x    3 root   root   0 May  14 11:43 .
dr-xr-xr-x   61 root   root   0 May  14 04:28 ..
-r--r--r--    1 root   root   0 May  14 11:43 cmdline
lrwxrwxrwx    1 root   root   0 May  14 11:43 cwd -> /
-r--------    1 root   root   0 May  14 11:43 environ
lrwxrwxrwx    1 root   root   0 May  14 11:43 exe -> /root/teste (deleted)
dr-x------    2 root   root   0 May  14 11:43 fd
-r--r--r--    1 root   root   0 May  14 11:43 maps
-rw-------    1 root   root   0 May  14 11:43 mem
lrwxrwxrwx    1 root   root   0 May  14 11:43 root -> /
-r--r--r--    1 root   root   0 May  14 11:43 stat
-r--r--r--    1 root   root   0 May  14 11:43 statm
-r--r--r--    1 root   root   0 May  14 11:43 status
```

No exemplo anterior, o programa `/root/teste` é executado e "deletado" em seguida. Na listagem do diretório `/proc/1094`, o *link* simbólico `/proc/1094/exe` aponta para o programa executado, apresentando a indicação `(deleted)`, pois o programa será marcado para "deleção" quando o processo 1094 terminar. Nesse caso, o programa `/root/teste` pode ser facilmente recuperado, enquanto o processo 1094 estiver em execução, através de um utilitário de cópia qualquer, como o `cp`:

```
# cp /proc/1094/exe /root/teste_recuperado
```

Outra fonte de informação útil em `/proc` é o sub-diretório `fd`, que possui um *link* simbólico para cada arquivo[5] que o processo tem aberto. Cada *link* recebe como nome um inteiro positivo, referente ao *file descriptor* do arquivo aberto:

```
# ls -la /proc/547/fd
total 0
dr-x------   2 root   root    0 May  15 10:16 .
dr-xr-xr-x   3 root   root    0 May  15 10:16 ..
lrwx------   1 root   root   64 May  15 10:16 0 -> socket:[732]
l-wx------   1 root   root   64 May  15 10:16 1 -> /var/log/messages
l-wx------   1 root   root   64 May  15 10:16 2 -> /var/log/secure
l-wx------   1 root   root   64 May  15 10:16 3 -> /var/log/maillog
l-wx------   1 root   root   64 May  15 10:16 4 -> /var/log/cron
l-wx------   1 root   root   64 May  15 10:16 5 -> /var/log/spooler
l-wx------   1 root   root   64 May  15 10:16 6 -> /var/log/boot.log
```

[5]Por arquivo entenda-se arquivos comuns, *devices files*, *sockets* e *pipes* [34, 68].

O exemplo anterior mostra o conteúdo do diretório `fd` relacionado ao processo `syslogd` (PID 547). Os *links* 0, 1 e 2 referem-se, respectivamente, à entrada padrão, saída padrão e saída de erro padrão.

A.5 Acesso às informações sobre o estado das conexões de rede

O comando `netstat` pode ser usado para capturar informações sobre o estado das conexões e portas abertas no sistema:

```
# netstat -anpe
```

A opção `-a` permite a visualização dos servidores aguardando uma conexão; a conversão de endereços em nomes é desativada com a opção `-n`; a opção `-p` mostra o número de identificação do processo e o nome do programa associado a cada conexão; e informações extras, como o usuário dono do processo, são habilitadas através da opção `-e`. Maiores detalhes sobre a saída e opções do comando `netstat` podem ser encontrados na respectiva *man page*.

A.6 Acesso às informações sobre o estado das interfaces de rede

O comando `ifconfig` pode ser usado para obter informações sobre as interfaces de rede, como ilustrado a seguir:

```
# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:30:21:08:8C:CE
          inet addr:10.1.1.1  Bcast:10.1.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:10 Base address:0xde00

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
```

```
RX packets:68 errors:0 dropped:0 overruns:0 frame:0
TX packets:68 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
```

No exemplo anterior, o comando `ifconfig` é executado com a opção `-a`, para fornecer informações sobre todas as interfaces de rede do sistema. A indicação `PROMISC`, na terceira linha das informações sobre a interface `eth0`, indica que a interface encontra-se em modo promíscuo, caracterizando a existência de um *sniffer* [67].

A.7 Acesso às informações sobre os usuários logados

O comando `w` pode ser usado para obter informações sobre os usuários logados[6], como apresentado no exemplo seguinte:

```
# w
  3:47pm  up  1:46,  5 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM   LOGIN@   IDLE   JCPU   PCPU   WHAT
root     tty1     -      3:23pm   23:39  0.21s  0.15s  -bash
root     pts/0    -      3:11pm   35:46  0.04s  0.04s  /bin/cat
root     pts/1    -      3:12pm   0.00s  11.56s 0.03s  w
root     pts/2    -      3:24pm   22:45  0.09s  0.09s  /bin/bash
```

A.8 Acesso às tabelas e *caches* mantidas pelo sistema operacional

A.8.1 Tabela e *cache* de roteamento

O comando `route` pode ser usado para determinar a tabela de rotas corrente do sistema analisado, como ilustrado no exemplo seguinte:

```
# route -een
Kernel IP routing table
Destination Gateway Genmask       Flags Metric Ref Use Iface MSS Window irtt
10.1.1.0    0.0.0.0 255.255.255.0 U     0      0     0 eth0  40  0      0
127.0.0.0   0.0.0.0 255.0.0.0     U     0      0     0 lo    40  0      0
```

[6] A listagem de usuários logados fornecida pelo comando `w` pode não ser completa, uma vez que alguns serviços como, por exemplo, o programa `su`, não registram os *logins* no arquivo de *log* `wtmp` (consultado pelo comando `w`).

No exemplo anterior, o comando `route` é executado com a opção `-ee`, que permite visualizar todos os parâmetros da tabela de rotas, além da opção `-n`, que impede a resolução de nomes. Maiores detalhes sobre a saída do comando `route` e suas opções podem ser encontrados na *man page* do mesmo. O comando `route` pode ser usado, ainda, para visualizar o conteúdo da *cache* de roteamento do sistema, através da opção `-C`:

```
# route -nC
Kernel IP routing cache
Source          Destination     Gateway         Flags Metric Ref Use Iface
143.106.23.1    200.154.152.241 200.154.152.241 1     0      0     6 lo
143.106.23.1    200.154.152.241 200.154.152.241 1     0      0     0 lo
200.176.2.10    200.154.152.241 200.154.152.241 1     0      0     0 lo
200.154.152.241 143.106.23.1    200.175.132.2         0      0     0 ppp0
200.154.152.241 200.176.2.10    200.175.132.2         0      0     0 ppp0
```

A.8.2 *Cache* de resolução de endereços MAC

O comando `arp` pode ser usado para visualizar o conteúdo da *cache* ARP do sistema analisado, como ilustrado a seguir:

```
# arp -nv
Address      HWtype  HWaddress           Flags Mask   Iface
10.1.1.44    ether   00:E0:4C:39:01:FB   C            eth0
10.1.1.45    ether   00:E0:4C:39:01:F3   C            eth0
10.1.1.40    ether   00:50:BF:7D:53:FF   C            eth0
10.1.1.1     ether   00:00:21:27:40:C0   C            eth0
Entries: 4      Skipped: 0      Found: 4
```

No exemplo anterior, a opção `-n` impede a resolução de nomes e a opção `-v` fornece informações extras. Maiores detalhes sobre a saída do comando `arp` e suas opções podem ser encontrados na *man page* do comando.

A.8.3 *Cache* de resolução de nomes

É possível extrair o conteúdo da *cache* do servidor DNS, para procurar entradas indevidas, através do comando `rndc` do BIND 9:

```
# rndc dumpdb
```

A.9 Investigação dos módulos de *kernel* carregados

A investigação dos LKMs do sistema analisado pode ser feita através da comparação das informações fornecidas pelos comandos `lsmod`, `kstat`[7] e pela interface `/proc/modules`, como ilustrado a seguir:

```
# lsmod
Module                  Size  Used by
serial_cs               5040   0  (unused)
pcnet_cs               10052   1
8390                    5860   0  [pcnet_cs]
ds                      6088   2  [serial_cs pcnet_cs]
i82365                 21276   2
pcmcia_core            43424   0  [serial_cs pcnet_cs ds i82365]
bsd_comp                3620   0

# cat /proc/modules
serial_cs               5040   0 (unused)
pcnet_cs               10052   1
8390                    5860   0 [pcnet_cs]
ds                      6088   2 [serial_cs pcnet_cs]
i82365                 21276   2
pcmcia_core            43424   0  [serial_cs pcnet_cs ds i82365]
bsd_comp                3620   0 (unused)

# ./kstat -M
Module                  Address
knull                   0xc283a000
oMBRa                   0xc2838000
serial_cs               0xc2835000
pcnet_cs                0xc282f000
8390                    0xc282c000
ds                      0xc2827000
i82365                  0xc281c000
pcmcia_core             0xc2810000
bsd_comp                0xc280e000
```

[7] A ferramenta `kstat` pode ser encontrada na URL http://www.s0ftpj.org/en/site.html (disponível em janeiro de 2003), juntamente com um HOWTO para detecção de LKMs.

O exemplo anterior ilustra uma situação onde um módulo malicioso foi instalado no sistema, modificando a chamada de sistema `sys_query_module`, para "camuflá-lo" na saída do comando `lsmod`, e a chamada de sistema `sys_read`, para impedir que o módulo seja visível em `/proc/modules`. O comando `kstat` é executado com a opção `-M`, que permite listar os LKMs instalados. A opção `-M` permite, ainda, varrer uma porção da memória do *kernel* em busca de LKMs invisíveis (*stealth*) e, então, tentar torná-los removíveis novamente. No exemplo anterior, é possível notar uma inconsistência na saída do comando `kstat`, onde aparece o módulo `oMBRa` (terceira linha), possivelmente suspeito (o módulo `knull` é instalado pelo próprio comando `kstat`). Outra indicação comum da existência de módulos maliciosos é a existência de erros ou ausência de dados na saída do comando `lsmod`.

Através da ferramenta `kstat` é possível, ainda, verificar se a tabela de chamadas de sistema foi possivelmente alterada pelo atacante, como ilustrado a seguir:

```
# kstat -s 0
SysCall                 Address
sys_exit                0xc011608c
sys_fork                0xc0107668
sys_read                0xc5801230 WARNING! should be at 0xc011356c
sys_query_module        0xd5809060 WARNING! should be at 0xc01169e0

# kstat -s 1
Restoring system calls addresses...
```

Na primeira execução do comando `kstat`, no exemplo anterior, é utilizada a opção `-s` com o argumento `0`, permitindo checar todos os endereços das chamadas de sistema. É possível observar que as chamadas de sistema `sys_read` e `sys_query_module` tiveram seus endereços modificados, conforme a situação descrita anteriormente. Já na segunda execução do comando `kstat`, a opção `-s`, com o argumento `1`, é usada para restaurar os endereços originais das chamadas de sistema modificadas.

Segundo o autor, a ferramenta `kstat` permite a visualização de informações extraídas diretamente das estruturas do *kernel*, através da interface provida pelo arquivo `/dev/kmem`. Sua utilização é especialmente útil quando não se pode confiar no sistema analisado, embora seja possível modificar o conteúdo da memória do *kernel*.

A.10 Acesso e recuperação de informações escondidas ou "deletadas"

Para facilitar a compreensão de como o disco pode ser utilizado sem a abstração de arquivos, uma breve introdução acerca da estrutura e organização desse dispositivo de armazenagem secundária é apresentada na sequência. Em seguida, são discutidas algumas formas de se esconder informações em um disco. Por fim, é apresentada a recuperação de informações escondidas ou "deletadas", contidas nas áreas do disco não acessíveis através de um sistema de arquivos.

A.10.1 Estrutura e organização do disco

Dentro de seu invólucro metálico, um disco é composto por uma pilha de pratos magnéticos, contendo, acima e abaixo de cada prato, uma cabeça de leitura e gravação. Cada prato é dividido em círculos concêntricos, chamados de trilhas, divididos em setores (menores unidades de alocação do disco). O conjunto de trilhas paralelas em cada prato é denominado de cilindro.

Cada setor do disco pode ser unicamente identificado por seu endereço CHS: C para o número do cilindro (começando em 0), H para o número da cabeça de leitura e gravação (começando em 0) e S para o número do setor (começando em 1) [87]. Normalmente o cilindro mais externo e a cabeça mais ao topo recebem o índice 0. No entanto, o primeiro cilindro físico do disco é reservado para o uso do controlador do disco, recebendo o índice -1 [87]. O cilindro -1 é acessível somente ao controlador, através de comandos internos específicos, e contém dados sobre o mapeamento de setores defeituosos e sobre a geometria física do disco (usados pela BIOS para auto-configuração, por exemplo) [87]. Um setor pode ser identificado também por seu endereço absoluto, que se inicia em zero e é incrementado até o final do disco [67].

Um disco pode ser logicamente dividido em partições, cada qual referenciada separadamente pelo sistema operacional. Desse modo, através do particionamento, um único disco pode ser transformado em múltiplos discos individuais. As informações sobre o particionamento são mantidas em uma área especial do disco chamada de tabela de partições. Todo disco possui um setor especial, denominado *master boot record* (MBR), localizado no endereço CHS 0,0,1. O MBR contém o programa de análise de partições (*partition analysis program*), executado durante a inicialização do sistema, e a tabela de partições [87]. O programa de análise de partições investiga a tabela de partições, contida nos últimos bytes do MBR, e determina qual é a partição ativa do disco (aquela que contém o sistema operacional que deve ser inicializado), desviando sua execução para o programa, denominado *bootstrap loader*, contido no primeiro setor da partição (setor de *boot*) [87].

O *bootstrap loader* inicializa, então, o sistema operacional contido na partição.

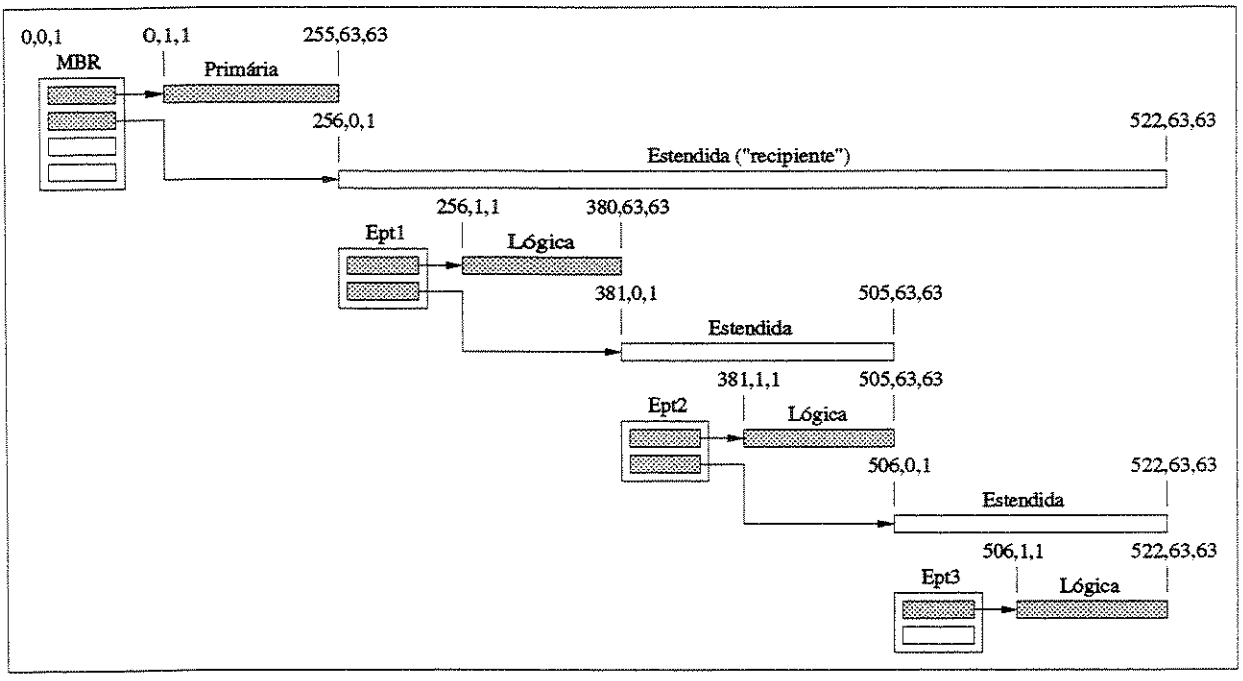


Figura A.1: Partições estendidas e lógicas.

A tabela de partições sempre começa no *offset* $1be_h$[8] do MBR e pode conter até quatro entradas de 16 bytes [87]. Cada uma dessas entradas pode referenciar uma partição primária, ou seja, uma partição que pode conter um *bootstrap loader* no primeiro setor, juntamente com um sistema operacional [87]. Para superar essa limitação de se ter apenas quatro partições no disco, uma das entradas da tabela de partições do MBR pode referenciar uma partição estendida. Essa entrada, referente à primeira partição estendida, é um "recipiente" que engloba uma ou mais partições lógicas (também chamadas de partições secundárias) e novas partições estendidas. O primeiro setor de uma partição estendida contém uma nova tabela de partições e o restante desse setor é usualmente preenchido com zeros. Essa tabela de partições estendida também começa no *offset* $1be_h$ do setor e, embora tenha quatro entradas de 16 bytes, é permitida a utilização de apenas duas delas. Somente a primeira entrada da tabela de partições estendida pode referenciar uma partição lógica, que pode abrigar um sistema de arquivos e, excepcionalmente, um sistema operacional (capaz de ser inicializado a partir de uma partição lógica, como o Windows NT, OS/2 e o Linux, por exemplo) [87]. A segunda entrada pode apenas especificar uma

[8] O caracter h subscrito indica um número no formato hexadecimal.

nova partição estendida, que, por sua vez, contém uma nova tabela de partições estendida [87]. A Figura A.1 ilustra a organização das partições estendidas e lógicas.

As informações referentes à geometria e particionamento do disco podem ser obtidas através dos comandos `fdisk`[9] ou `sfdisk`, como ilustrado a seguir:

```
# fdisk -lu

Disk /dev/hda: 255 heads, 63 sectors, 1240 cylinders
Units = sectors of 1 * 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *          63     9221309     4610623+   c  Win95 FAT32 (LBA)
/dev/hda2        9221310    18040994     4409842+  83  Linux
/dev/hda3       18040995    18458684      208845   82  Linux swap
/dev/hda4       18458685    19920599      730957+   5  Extended
/dev/hda5       18458748    19486844      514048+  83  Linux
/dev/hda6       19486908    19695689      104391   83  Linux
/dev/hda7       19695753    19920599      112423+  83  Linux
```

No exemplo anterior, o comando `fdisk` é executado com a opção `-l`, que permite visualizar informações sobre as partições sem alterar nenhum dado do disco, além da opção `-u`, que fornece as informações em termos de setores (ao invés de cilindros). O comando `fdisk`, com a opção `-u`, utiliza o endereçamento absoluto de setores (começando em zero e incrementando até o final do disco), facilitando a localização exata das partições.

É possível, ainda, obter tais informações de geometria e particionamento a partir da imagem do disco armazenada em um arquivo[10], utilizando-se o comando `sfdisk` como segue (no exemplo seguinte, a imagem do disco está armazenada no arquivo `hda.dd`):

```
# sfdisk -luS hda.dd

Disk hda.dd: 1240 cylinders, 255 heads, 63 sectors/track
Units = sectors of 512 bytes, counting from 0

 Device Boot    Start       End   #sectors  Id  System
hda.dd1   *        63   9221309    9221247   c  Win95 FAT32 (LBA)
hda.dd2       9221310  18040994    8819685  83  Linux
```

[9]Segundo a própria *man page* do utilitário `fdisk`, o programa `sfdisk` é uma versão mais correta e poderosa.

[10]Os diferentes tipos de imagem do disco são discutidos em detalhes na Seção 5.5.4.

```
hda.dd3      18040995  18458684    417690  82  Linux swap
hda.dd4      18458685  19920599   1461915   5  Extended
hda.dd5      18458748  19486844   1028097  83  Linux
hda.dd6      19486908  19695689    208782  83  Linux
hda.dd7      19695753  19920599    224847  83  Linux
```

A.10.2 Possíveis locais para se esconder informações no disco

A explanação apresentada anteriormente, no nível de detalhes usado, é necessária para identificar as diversas oportunidades que existem para esconder informações em um disco, incluindo, por exemplo [87]:

- com controladores antigos, setores e trilhas perfeitos podem ser marcados como defeituosos, através de comandos internos do controlador, de modo que as informações neles contidas tornam-se inacessíveis;
- algumas partições podem ser deliberadamente marcadas como "escondidas", usando por exemplo o programa PartitionMagic[11];
- as partições acessíveis podem não ocupar todo o disco, existindo espaços suspeitos entre as partições ou no final do disco. Além disso, a tabela de partições pode ter sido modificada, usando algum editor de disco, de modo que uma ou mais partições válidas não são mais registradas;
- a BIOS pode não suportar a geometria do disco, de modo que não é permitido acesso a toda porção física do disco;
- as partições lógicas podem não ocupar totalmente a primeira partição estendida (partição "recipiente"), sobrando um espaço suspeito ao seu final;
- como indicado na Figura A.1, é uma prática normal as tabelas de partições (do MBR ou estendidas) começarem na cabeça 0, setor 1 de um cilindro, e o primeiro setor da primeira partição começar na cabeça 1, setor 1 do cilindro. A consequência dessa prática é a existência de setores não utilizados entre o setor da tabela de partições e início da primeira partição. Esses setores podem ser utilizados para esconder informações, sem qualquer risco de serem detectadas pelo uso normal do sistema de arquivos;

[11]Informações sobre o programa PartitionMagic podem ser encontradas na URL http://www.powerquest.com (disponível em janeiro de 2003).

- apenas o MBR normalmente contém um programa de análise de partições. Uma tabela de partições estendida (Ept1, Ept2 e Ept3 na Figura A.1) apenas contém uma ou duas entradas de partições, começando no *offset* $1be_h$ do setor, de modo que a maior parte do setor não é utilizada. Os bytes do *offset* 00_h ao $1bd_h$ do setor que contém uma tabela de partições estendida podem ser usados para esconder informações;

- um sistema de arquivos pode não ocupar totalmente sua partição, devido ao fato do tamanho da partição não ser múltiplo do tamanho de bloco utilizado pelo sistema de arquivos. Os últimos setores desperdiçados devem ser checados;

- podem existir partições que não contêm um sistema de arquivos. Isso pode ser proposital, com o intuito de desviar a atenção da partição e esconder informações;

- um determinado sistema de arquivos pode estar sendo utilizado apenas para desviar a atenção, de modo que os espaços não alocados aos arquivos podem estar sendo usados diretamente;

- os *file slacks* também podem ser usados para esconder pequenas porções de dados;

- o programa de análise de partições do MBR e o *bootstrap loader* do setor de *boot* das partições podem ser deliberadamente alterados, permitindo a execução de código malicioso durante a inicialização do sistema;

- áreas de *swap*, que não contêm um sistema de arquivos, podem armazenar diversos dados temporários como, por exemplo, dados do *kernel*, de processos, *buffers* de arquivos temporários e da impressora. Entre essas informações podem estar dados que nunca foram salvos no disco ou, ainda, dados deliberadamente escondidos;

A.10.3 Extração e recuperação de informações escondidas ou "deletadas"

As informações armazenadas em áreas não acessíveis através de um sistema de arquivos podem ser extraídas por meio de *raw I/O*, utilizando-se, por exemplo, o comando `dd` e o *device file* (ou a imagem em arquivo) correspondente ao disco analisado, como ilustrado a seguir:

```
# dd if=hda.dd of=unusedSectors.dd bs=512 skip=1 count=62
62+0 records in
62+0 records out
```

No exemplo anterior, o comando `dd` é usado para extrair o espaço não utilizado entre o MBR (setor 0) e o setor de *boot* da primeira partição do disco (setor 63). A informação é extraída da imagem do disco, armazenada no arquivo `hda.dd`, e é preservada no arquivo binário `unusedSectors.dd`. As opções `bs`, `skip` e `count` são utilizadas para instruir o comando `dd` a copiar 62 setores de 512 bytes, a partir do setor de número 1. Maiores detalhes sobre as opções do comando `dd` podem ser encontrados na respectiva *man page*.

Uma maneira mais eficiente de extrair as informações contidas nas áreas não acessíveis através de um sistema de arquivos é através de ferramentas específicas. O utilitário `unrm`, que compõe o conjunto de ferramentas chamado *The Coroner's Toolkit* (TCT)[12], é capaz de extrair toda informação presente nos espaços não alocados de um sistema de arquivos, como exemplificado a seguir:

```
# unrm hda2.dd > unallocated.unrm
```

No exemplo anterior, o comando `unrm` é usado para copiar, no arquivo binário `unallocated.unrm`, toda informação contida nos espaços não alocados do sistema de arquivos da imagem `hda2.dd` (imagem da partição Linux `/dev/hda2` do disco suspeito). É importante mencionar que o arquivo destino deve estar em outro sistema de arquivos que o utilizado como fonte para o comando `unrm` e o seu tamanho pode compreender vários gigabytes de dados. Além disso, o comando `unrm` acessa apenas as informações dos espaços não alocados, não extraindo, por exemplo, os dados contidos nos *file slacks* (que são áreas alocadas).

As informações presentes nos *file slacks* podem ser extraídas, por exemplo, através da ferramenta `bmap`[13], como segue:

```
# md5sum -b /etc/passwd
e437472d827159584ddb2f7e0f038d88 */etc/passwd

# echo "Hello World" | bmap -mode putslack /etc/passwd
stuffing block 378669
file size was: 897
slack size: 3199
block size: 4096

# md5sum -b /etc/passwd
e437472d827159584ddb2f7e0f038d88 */etc/passwd
```

[12] O TCT é apresentado em maiores detalhes no Apêndice B.

[13] A ferramenta `bmap` pode ser obtida na URL ftp://ftp.scyld.com/pub/forensic_computing/bmap/ (disponível em janeiro de 2003).

```
# bmap -mode slack /etc/passwd
getting from block 378669
file size was: 897
slack size: 3199
block size: 4096
Hello World
```

Inicialmente, no exemplo anterior, é computado o *hash* criptográfico MD5 do arquivo `/etc/passwd`, através do comando `md5sum`[14]. Em seguida, na primeira execução da ferramenta `bmap`, a frase "`Hello World`" é introduzida no *file slack* do arquivo. É importante notar que o *hash* criptográfico do arquivo não é alterado. Na segunda execução, o `bmap` é utilizado para recuperar as informações contidas no *file slack*.

A análise das informações extraídas das áreas não acessíveis através de um sistema de arquivos é, na maioria das vezes, um processo tedioso e demorado, uma vez que esses dados geralmente constituem um fluxo de bits sem estrutura alguma aparente. Um editor hexadecimal pode ser usado para visualizar e examinar essas informações. Nesse sentido, o comando `xxd` é bastante útil, permitindo a conversão de um fluxo de bits qualquer para o formato hexadecimal:

```
# dd if=/dev/hda bs=512 count=1 skip=0 | xxd
1+0 records in
1+0 records out
0000000: faeb 7c6c 6261 4c49 4c4f 0100 1504 5a00  ..|lbaLILO....Z.
0000010: 0000 0000 b789 9f3c a742 b0ca 00a8 42b0  .......<.B....B.
0000020: ca00 a642 b0ca 0001 445a aa42 b0ca 00ab  ...B....DZ.B....
0000030: 42b0 ca00 6f2b b0c9 0070 2bb0 c900 712b  B...o+...p+...q+
...
00001b0: 595f eb02 b440 0000 0000 0000 0000 8001  Y_...@..........
00001c0: 0100 0cfe bf3d 3f00 0000 7fb4 8c00 0000  .....=?.........
00001d0: 813e 83fe ffff beb4 8c00 e593 8600 00fe  .>..............
00001e0: ffff 82fe ffff a348 1301 9a5f 0600 00fe  .......H..._....
00001f0: ffff 05fe ffff 3da8 1901 9b4e 1600 55aa  ......=....N..U.
```

O exemplo anterior mostra as informações do MBR do disco `/dev/hda`, processadas pelo comando `xxd`. É possível visualizar as informações nos formatos hexadecimal e

[14] O *hash* criptográfico é de grande utilidade no processo de análise forense, permitindo estabelecer uma assinatura confiável para um fluxo de bits qualquer. Essa assinatura é utilizada para checagem de integridade e autenticidade, que pode ser totalmente automatizada, através de ferramentas como o Tripwire, ou "manual", por meio do comando `md5sum` (no caso de *hashes* MD5).

ASCII. No exemplo anterior, pode-se observar parte do código do programa de análise de partições, no início do setor, e a tabela de partições, estendendo-se do *offset* $1be_h$ ao $1fd_h$.

Outra abordagem que pode diminuir a complexidade da análise dessas informações não-estruturadas é a busca direta de palavras-chave, utilizando, por exemplo, os comandos `strings` e `grep`. O exemplo seguinte ilustra essa abordagem:

```
# unrm /dev/hda5 | strings -a | grep Jul
Jul 19 14:38:05 host pppd[866]: Exit.
Jul 19 13:52:22 host kde[690]: session opened for user root by (uid=0)
```

No exemplo anterior, é utilizada uma combinação dos comandos `unrm`, `strings` e `grep` para buscar entradas de *log* "deletadas", referentes ao mês de julho, na partição que abriga o diretório `/var/log` (repositório dos principais arquivos de *log* do Linux).

Além da análise dos dados com editores hexadecimal e da busca de palavras-chave, pode-se ainda tentar organizar as informações de maneira coerente e estruturada, formando arquivos de dados (binários, de texto, figuras, entre outros tipos de arquivos). O conjunto de ferramentas TCT contém um programa, chamado `lazarus`, que recebe um fluxo de bits como entrada e sistematicamente analisa cada bloco de dados, tentando reconstruir arquivos com informações coerentes[15]. O `lazarus` pode ser usado com qualquer tipo de fluxo de bits, incluindo, por exemplo, *dumps* da memória, áreas de *swap* e *core files*.

A.11 Preparação do sistema de arquivos para o processo de análise forense

Como é visto em detalhes na Seção 5.5.4, a imagem do disco suspeito pode ser armazenada em um único arquivo (contendo todo o disco), em vários arquivos (contendo separadamente cada sistema de arquivos ou partição do disco) ou, ainda, espelhada em outro disco de tamanho igual ou superior (chamado de disco espelho). Desse modo, o sistema de arquivos a ser analisado pode apresentar-se idealmente de três formas:

- espelhado em um disco instalado no sistema de análise (o disco espelho);
- contido na imagem em arquivo de todo o disco suspeito;
- ou contido em uma imagem em arquivo individual;

[15]Maiores detalhes sobre o funcionamento do `lazarus` são apresentados no Apêndice B.

Na primeira forma, o sistema de arquivos analisado encontra-se em uma das partições do disco espelho, podendo ser acessado diretamente através de um *device file* do sistema de análise. Na terceira forma, o arquivo contendo a imagem individual do sistema de arquivos pode ser usado diretamente em muitas ferramentas, substituindo o *device file*. Já na segunda forma, é preciso identificar na imagem em arquivo de todo o disco, onde começa o sistema de arquivos em questão. Isso pode ser feito através de um *device file* especial, chamado *loopback device*, e do comando `losetup`:

```
# sfdisk -luS hda.dd

Disk hda.dd: 1240 cylinders, 255 heads, 63 sectors/track
Units = sectors of 512 bytes, counting from 0

   Device Boot  Start     End  #sectors  Id  System
hda.dd1   *        63   9221309   9221247   c  Win95 FAT32 (LBA)
hda.dd2       9221310  18040994   8819685  83  Linux
hda.dd3      18040995  18458684    417690  82  Linux swap
hda.dd4      18458685  19920599   1461915   5  Extended
hda.dd5      18458748  19486844   1028097  83  Linux
hda.dd6      19486908  19695689    208782  83  Linux
hda.dd7      19695753  19920599    224847  83  Linux

# losetup -o 4721310720 /dev/loop0 hda.dd
```

No exemplo anterior, o *loopback device* `/dev/loop0` é associado ao sistema de arquivos presente na partição `hda.dd2` (referente à partição `/dev/hda2` do disco original). Isso é feito indicando-se o *offset*, em bytes, a partir do início do arquivo `hda.dd` (arquivo contendo a imagem de todo o disco), onde se encontra o sistema de arquivos desejado. No caso do exemplo, o *offset* indicado é 4721310720 bytes, referente ao endereço absoluto do setor de início da partição (9221310), multiplicado por 512 bytes (que é o tamanho de cada setor). Dessa forma, o *loopback device* `/dev/loop0` pode ser utilizado para acessar as informações do sistema de arquivos analisado, do mesmo modo que o *device file* `/dev/hda2` seria utilizado no disco original. Maiores detalhes sobre o comando `losetup` podem ser obtidos na *man page* do mesmo.

Uma vez identificada a forma de acesso ao sistema de arquivos a ser analisado, ele pode ser montado no sistema de análise, permitindo o exame de seus arquivos e diretórios. Se o sistema de arquivos é acessível através de um disco espelho, sua montagem pode ser feita normalmente, utilizando o *device file* correspondente, como ilustrado a seguir:

```
# mount -t ext2 -o ro,noexec,nodev /dev/hda2 /mnt/mountpoint
```

É importante observar que o sistema de arquivos analisado deve ser montado apenas para leitura (através da opção `ro`), impedindo qualquer tipo de alteração. A execução de programas e a interpretação de *device files* contidos no sistema de arquivos montado também podem ser desabilitadas (através das opções `noexec` e `nodev`). É conveniente criar um diretório específico, referente ao caso investigado, como ponto de montagem do sistema de arquivos analisado. Maiores informações sobre o comando `mount` e suas opções podem ser encontradas na *man page* do comando.

Nos casos em que o sistema de arquivos analisado está contido em uma imagem em arquivo, a montagem é feita utilizando-se um *loopback device* (através das opções `loop` e `offset`), como segue:

```
# mount -t ext2 -o ro,noexec,nodev,loop=/dev/loop0 hda2.dd \
  /mnt/mountpoint
# mount -t ext2 -o ro,noexec,nodev,loop=/dev/loop1,offset=4721310720 \
  hda.dd /mnt/mountpoint
```

No exemplo anterior, o arquivo `hda2.dd` contém a imagem individual do sistema de arquivos analisado, que é montado através do *loopback device* `/dev/loop0`. Já o arquivo `hda.dd` contém a imagem de todo o disco, precisando ser indicado o *offset* (em bytes) onde se encontra o sistema de arquivos analisado (de maneira idêntica ao apresentado anteriormente para o comando `losetup`). Se o comando `losetup` foi utilizado anteriormente para associar um *loopback device* ao sistema de arquivos analisado, esse *loopback device* pode ser utilizado diretamente, no comando `mount`, para montar o sistema de arquivos, como ilustrado a seguir:

```
# losetup -o 4721310720 /dev/loop0 hda.dd
# mount -t ext2 -o ro,noexec,nodev /dev/loop0 /mnt/mountpoint
```

Devidamente preparado e montado, o sistema de arquivos pode ser examinado em busca de evidências da intrusão, quer seja através da extração de informações de suas estruturas internas ou da análise de seus arquivos e diretórios, como apresentado na sequência.

A.12 Extração de informações das estruturas internas do sistema de arquivos

Cada arquivo ou diretório é representado no sistema Linux por uma estrutura de dados chamada *inode*, que contém a localização em disco dos blocos de dados do arquivo

Elemento	Descrição
USERID e GROUPID	Números de identificação do usuário proprietário do arquivo e do grupo proprietário. Os valores são indexados aos nomes contidos em `/etc/passwd` e `/etc/group`, respectivamente
Tipo	Tipo do *inode* : regular (arquivo comum), diretório, dispositivo de caracter (*character device*), dispositivo de bloco (*block device*) ou *named pipe*
Permissões de acesso	Permissões para escrita, leitura e execução. As permissões são dadas para três diferentes abstrações: o proprietário, o grupo e todos que têm acesso ao sistema
Marcas de tempos: *mtime, atime* e *ctime*	Tempos de (respectivamente): última modificação de conteúdo, último acesso e última mudança nas propriedades do arquivo
Número de *links*	Número de entradas de diretório referenciando o *inode*
Ponteiros para os blocos de dados	Localização dos blocos de dados no disco
Tamanho	Tamanho do arquivo

Tabela A.1: Conteúdo de um *inode*.

ou diretório e informações sobre propriedade, permissões de acesso e marcas de tempo (*timestamps*). A Tabela A.1 resume o conteúdo de um *inode*.

No caso particular dos diretórios, os blocos de dados contêm estruturas especiais denominadas entradas de diretório. Cada uma dessas estruturas referencia um arquivo ou diretório, armazenando, entre outras informações, o nome e o número do *inode* do arquivo ou diretório referenciado (essa referência é chamada de *link*). Nota-se, portanto, que o *inode* não contém o nome do respectivo arquivo ou diretório, e sim a entrada de diretório que referencia o *inode*. Essa organização permite que um *inode* seja referenciado por várias entradas de diretório diferentes, possuindo diversos *links* e possivelmente diferentes nomes. Logo, durante a análise forense, é importante lembrar que um mesmo arquivo pode aparecer um múltiplos diretórios simultaneamente e que esse arquivo pode ter nomes diferentes em cada diretório [47]. O acesso às diversas estruturas do sistema de arquivos é apresentado como segue.

A.12.1 Acesso às informações do *inode*

O conteúdo do *inode* relacionado a um arquivo ou diretório pode ser visualizado através do comando `stat`, como ilustrado a seguir:

```
# stat /var/log/messages
  File: "/var/log/messages"
  Size: 2984638    Blocks: 5858       Regular File
Access: (0600/-rw-------)         Uid: ( 0/ root )  Gid: ( 0/ root )
Device: 305        Inode: 12259      Links: 1
Access: Fri Jul 19 16:24:38 2002
Modify: Tue Jul 23 14:12:25 2002
Change: Tue Jul 23 14:12:25 2002
```

No exemplo anterior, é visualizado o *inode* de número 12259, referente ao arquivo `/var/log/messages`. É possível identificar, entre outras informações, as permissões de acesso[16] ao arquivo (`Access`), os USERID e GROUPID do proprietário (`Uid` e `Gid`[17], respectivamente) e as marcas de tempo do arquivo (`Modify`, `Access` e `Change`). O programa `stat` pode, ainda, ser executado com a opção `-f`, permitindo obter informações sobre o sistema de arquivos onde reside o arquivo passado como parâmentro, incluindo, por exemplo, o tipo do sistema de arquivos e o número de *inodes* e blocos de dados, como ilustrado a seguir:

```
# stat -f .
  File: "."
    ID: 0        0      Namelen: 255  Type: EXT2
Blocks: Total: 1085138  Free: 668780  Available: 613657  Size: 4096
Inodes: Total: 551616   Free: 445510
```

Outra ferramenta que permite visualizar as informações de um determinado *inode* é o programa `istat`, pertencente ao conjunto de ferramentas TCTUTILs[18]. Diferente do comando `stat`, a ferramenta `istat` mostra o endereço dos blocos de dados do arquivo ou diretório associado ao *inode*, como no exemplo seguinte:

[16]Uma explanação detalhada sobre permissões de acesso e propriedade de arquivos pode ser encontrada em [34].

[17]Os nomes associados, na saída do comando `stat`, ao USERID e GROUPID do arquivo são indexados a partir dos arquivos `/etc/passwd` e `/etc/group` da máquina onde o sistema de arquivos está sendo examinado. Desse modo, durante a análise forense, é recomendado utilizar os valores numéricos USERID e GROUPID ou fazer manualmente a associação com os respectivos nomes, utilizando os arquivos `/etc/passwd` e `/etc/group` do sistema suspeito [47].

[18]O TCTUTILs é apresentado em maiores detalhes no Apêndice B.

```
# istat /dev/hda2 2
inode: 2
Allocated
uid / gid: 0 / 0
mode: rwxr-xr-x
size: 4096
num of links: 18
Modified:       07.25.2002 08:17:50     (AMT+0)
Accessed:       07.25.2002 09:26:33     (AMT+0)
Changed:        07.25.2002 08:17:50     (AMT+0)
Deleted:        12.31.1969 20:00:00     (AMT+0)
Direct Blocks:
  511
```

No exemplo anterior, o conteúdo do *inode* de número 2 (associado ao diretório /) é visualizado. O utilitário `bcat`, também pertencente ao TCTUTILs, pode ser usado para visualizar o conteúdo do bloco 511, alocado ao diretório / (como visto no exemplo anterior), da seguinte forma:

```
# bcat -h /dev/hda2 511
0       02000000 0c000102 2e000000 02000000     .... .... .... ....
16      0c000202 2e2e0000 0b000000 14000a02     .... .... .... ....
32      6c6f7374 2b666f75 6e640000 c17e0000     lost +fou nd.. .~..
48      0c000302 76617200 81fd0000 28000402     .... var. .... (...
64      70726f63 1d000000 1c000801 706f7765     proc .... .... powe
80      726f6666 6c2d7374 61676532 2e696d67     roff l-st age2 .img
96      417c0100 0c000302 746d7000 01fb0100     A|.. .... tmp. ....
112     0c000302 64657600 c1790200 0c000302     .... dev. .y.. ....
128     65746300 41770300 0c000302 75737200     etc. Aw.. .... usr.
```

A listagem do bloco 511 continua até completar o tamanho do bloco (4096 bytes nesse caso). É possível observar as entradas de diretório contidas no bloco como, por exemplo, a entrada que vai do *offset* 96 ao 107 da listagem, apresentada em detalhes na Tabela A.2 (de acordo com a estrutura `ext2_dir_entry_2` do sistema de arquivos EXT2FS).

A.12.2 Acesso às entradas de diretório

Embora a listagem dos blocos de dados alocados a um diretório permita a visualização de suas entradas de diretório, como visto anteriormente, é necessário identificá-las e interpretá-las "manualmente" (conforme as informações apresentadas na Tabela A.2). A

Campo	Valor hexadecimal	Significado
Número do *inode* (32 bits no formato *little endian*)	`417c0100`	A entrada referencia o *inode* número 97345
Tamanho da entrada (16 bits no formato *little endian*)	`0c00`	A entrada ocupa 12 bytes no total
Tamanho do nome contido na entrada (8 bits)	`03`	O nome ocupa 3 bytes (sem contar o caracter nulo)
Tipo da entrada (8 bits)	`02`	A entrada referencia um diretório
Nome do arquivo ou diretório (tamanho variável de no máximo 255 caracteres, terminado com um caracter nulo)	`746d7000`	O nome do diretório referenciado é `tmp`

Tabela A.2: Conteúdo de uma entrada de diretório.

automatização desse processo é possível através do programa `fls`, integrante do conjunto de ferramentas TCTUTILs, permitindo a listagem e interpretação das entradas de diretório contidas nos blocos de dados alocados a um diretório qualquer. É possível visualizar o nome dos arquivos ou diretórios referenciados, o número dos *inodes* e o tipo de cada entrada, como ilustrado a seguir:

```
# fls /dev/hda2 441130
r 441157:       teste
r 441159:       teste.c
d 457411:       frames
r * 441337:     files
```

No exemplo anterior, são visualizadas as entradas de diretório contidas nos blocos de dados alocados ao diretório associado ao *inode* 441130.

A.12.3 *Mactimes*

As marcas de tempo dos arquivos, chamadas de *mactimes*, representam recursos importantes para ajudar na reconstrução dos eventos associados a uma intrusão [10]. O sistema UNIX, em geral, possui pelo menos três marcas de tempo para cada arquivo: última modificação de conteúdo (*mtime*), último acesso (*atime*) e última mudança nas propriedades

do arquivo (*ctime*). O sistema Linux possui ainda uma quarta marca, com o tempo de "deleção" do arquivo (*dtime*).

Os *mactimes* são armazenados no formato correspondente ao número de segundos passados desde a "época do UNIX" (01 de janeiro de 1970) e indicam o tempo no horário de *Greenwich* (GMT). A interpretação dos *mactimes*, feita por utilitários como o `ls`, varia de acordo com a zona de tempo (*timezone*) configurada, através da variável de ambiente TZ, na máquina onde o utilitário é executado. Desse modo, a análise forense deve computar os tempos no horário de *Greenwich* ou adequar a variável de ambiente TZ, da máquina de análise, ao valor encontrado no sistema suspeito [47].

A visualização dos *mactimes* de um arquivo pode ser feita de maneira simples através do comando `ls`, indicando a opção correspondente ao *mactime* desejado (`-u` para *atime*, `-c` para *ctime* e nenhuma para *mtime*, que é a marca de tempo visualizada na execução padrão), ou por meio do programa `stat`, como visto anteriormente. Entretanto, o conjunto de ferramentas TCT possui um utilitário, chamado `mactime`, que permite criar uma linha de tempo, cronologicamente ordenada, com os arquivos e seus respectivos *mactimes*:

```
# mactime -d /root 07/24/2002
Jul 24 02 08:49:14   4096 mac drwxr-xr-x root root  /root/backups
Jul 24 02 08:49:48  19269 m.c -rw------- root root  /root/.bash_history
Jul 24 02 08:55:10     16 .a. -rw------- root root  /root/.esd_auth
Jul 24 02 13:14:06      7 .a. -rw-r--r-- root root  /root/.wmrc
Jul 24 02 13:14:10      7 m.c -rw-r--r-- root root  /root/.wmrc
                      101 m.c -rw------- root root  /root/.Xauthority
Jul 24 02 13:14:11    266 .a. -rw-r--r-- root root  /root/.bash_profile
                     1126 .a. -rw-r--r-- root root  /root/.Xresources
```

No exemplo anterior, o utilitário `mactime` é executado sobre o diretório `/root`, através da opção `-d`, requisitando uma listagem dos arquivos que tiveram algum *mactime* (indicado pela letra inicial — m, a ou c) alterado a partir da data 24 de julho de 2002 (no formato americano mês/dia/ano). É possível observar, por exemplo, que o arquivo `/root/.wmrc` foi acessado pela última vez às 13:14:06 do dia 24 de julho de 2002 (quarta linha da listagem) e teve seu conteúdo modificado às 13:14:10 do dia 24 de julho de 2002 (quinta linha da listagem).

Infelizmente os *mactimes* são facilmente modificados e extremamente efêmeros — a próxima vez que alguém acessar ou modificar um arquivo de qualquer forma, os valores anteriores dos *mactimes* serão perdidos. Entretanto, se houver razoável certeza de que os *mactimes* não foram deliberadamente distorcidos, eles podem fornecer pistas valiosas sobre os arquivos e diretórios acessados e/ou modificados pelo invasor.

A.12.4 Recuperação de dados "deletados" do sistema de arquivos

A reconstrução de informações "deletadas", através da análise direta dos setores disponíveis do disco, sem a utilização de qualquer estrutura de dados que indique alguma organização dessas informações, é uma tarefa bastante complexa e na maioria das vezes ineficiente, como visto na Seção A.10. Por outro lado, a utilizando das pistas deixadas nas estruturas de dados do sistema de arquivos torna o processo de recuperação de informações "deletadas" mais simples e imediato.

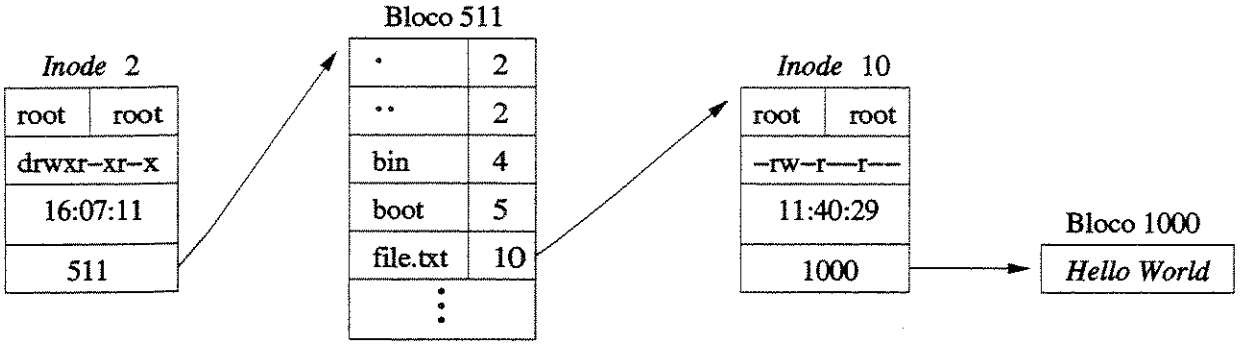


Figura A.2: Representação das estruturas internas de um sistema de arquivos, referentes ao arquivo `/file.txt` (o *inode* 2 é alocado ao diretório `/`, cujas entradas de diretório estão armazenadas no bloco de dados 511, e o *inode* 10 é alocado ao arquivo `file.txt`, cujas informações estão no bloco de dados 1000).

A quantidade e qualidade das pistas deixadas nas estruturas de dados do sistema de arquivos determina o número de informações que podem ser recuperadas sobre um arquivo ou diretório "deletado". Por exemplo, seguindo a representação ilustrada na Figura A.2, quando o arquivo `/file.txt` é "deletado", o sistema não apaga as informações contidas no bloco de dados e *inode* alocados ao arquivo, nem na respectiva entrada de diretório. Na verdade, essas estruturas são apenas disponibilizadas para reutilização por um novo arquivo ou diretório. Utilizando-se algumas ferramentas, pertencentes tanto ao TCT quanto ao TCTUTILs, e dependendo do grau de utilização do sistema de arquivos, é possível recuperar as informações da entrada de diretório referente ao arquivo "deletado", do *inode* que ele ocupava e, ainda, dos blocos de dados utilizados (caso essas estruturas não tenham sido reutilizadas).

Através da ferramenta `fls`, apresentada anteriormente, é possível visualizar as entradas de diretório referentes a arquivos ou diretórios "deletados", como ilustrado a seguir (conforme, ainda, a representação da Figura A.2):

```
# fls -d /dev/hda2 2
r * 10:          file.txt
r * 343:         poweroff
```

No exemplo anterior, a ferramenta `fls` é executada com a opção `-d`, permitindo listar as entradas de diretório referentes aos arquivos e diretórios "deletados", contidas nos blocos de dados do diretório / (*inode* 2). É possível observar a entrada relacionada ao arquivo "deletado" `/file.txt`, permitindo recuperar o nome do arquivo (caso não fosse conhecido) e o número do *inode* anteriormente ocupado (*inode* 10). De posse dessas informações, pode-se verificar se o *inode* associado ao arquivo "deletado" encontra-se disponível (através, por exemplo, do programa `istat`) e, então, utilizar outras ferramentas para tentar recuperar maiores informações sobre o arquivo e até mesmo seu conteúdo.

O conjunto de ferramentas TCT possui um utilitário, chamado `ils`, que permite visualizar as informações contidas em *inodes* não alocados, referentes ao último "ocupante" do *inode* (arquivos ou diretórios "deletados"). Essas informações incluem, por exemplo, os USERID e GROUPID do proprietário, os *mactimes* e os ponteiros para os blocos de dados do arquivo ou diretório. Outro utilitário do TCT, chamado `icat`, permite visualizar o conteúdo de um arquivo ou diretório a partir do número do *inode* correspondente, podendo ser utilizado inclusive com *inodes* não alocados, para a recuperação do conteúdo de arquivos ou diretórios "deletados". Tais ferramentas podem ser utilizadas em conjunto para obter informações e recuperar arquivos ou diretórios "deletados", como ilustrado a seguir (considerando novamente a representação da Figura A.2):

```
# ils /dev/hda2 10 | ils2mac > inode_info
# mactime -b inode_info 7/25/2002
Jul 25 02 11:40:29  6 ma. -rw-r--r-- root root  <hda2-dead-10>
Jul 25 02 11:40:30  6 ..c -rw-r--r-- root root  <hda2-dead-10>
# icat /dev/hda2 10
Hello World
```

No exemplo anterior, a ferramenta `ils` é utilizada para recuperar as informações contidas no *inode* 10 (disponibilizado pela "deleção" do arquivo `/file.txt`). Juntamente com a ferramenta `ils`, é executado o *script* `ils2mac` (também pertencente ao TCT), que permite converter a saída do comando `ils` para o formato processado pela ferramenta `mactime`. As informações recuperadas e convertidas são armazenadas no arquivo `inode_info`. Em seguida, o comando `mactime` é executado sobre os dados contidos no arquivo `inode_info`, permitindo a visualização das informações relacionadas ao último "ocupante" do *inode* 10 (possivelmente o arquivo "deletado" `/file.txt`). Por fim, a ferramenta `icat` é utilizada para recuperar o conteúdo dos blocos de dados referenciados pelo *inode* 10 (no caso do exemplo, o conteúdo do arquivo `/file.txt` é recuperado).

A.13 Análise das configurações do sistema

Algumas ferramentas podem facilitar a análise das configurações do sistema, incluindo, por exemplo, o comando `diff`, que pode ser usado para identificar as diferenças entre os arquivos de configuração e suas cópias de segurança, além de ferramentas de análise de vulnerabilidades [3], como o COPS [28], que varre o sistema em busca de falhas de segurança conhecidas em suas configurações. Outra alternativa é o comando `rpm`, que pode ser usado para checar a consistência dos binários instalados e de alguns arquivos de configuração do sistema[19], como ilustrado a seguir:

```
# rpm -Va
S.5....T c /etc/printcap
S.5....T c /etc/man.config
.M....G.   /dev/tty1
S.5....T   /usr/share/umb-scheme/slibcat
.......T c /etc/pam.d/system-auth
S.5....T c /etc/xinetd.d/telnet
```

Todo objeto que tem alterada alguma das características apresentadas na sequência é listado pelo comando `rpm`, juntamente com a letra associada à mudança (maiores detalhes sobre o comando `rpm` podem ser encontrados na *man page* do mesmo):

- 5: alteração na assinatura MD5;
- S: alteração no tamanho do arquivo;
- L: alteração no *link* simbólico;
- T: alteração no tempo de modificação;
- D: alteração no *device*;
- U: alteração no usuário proprietário;
- G: alteração no grupo proprietário;
- M: alteração nas permissões;

[19] Existe uma discussão na SANS sobre a utilização do `rpm` como ferramenta de recuperação. Maiores detalhes podem ser encontrados na URL http://www.sans.org/newlook/resources/IDFAQ/RPM.htm (disponível em janeiro de 2003).

A.14 Busca e análise dos executáveis e bibliotecas suspeitas

A.14.1 Localização dos executáveis

Os programas SUID e SGID costumam ser bastante explorados pelos atacantes [67]. Através do comando `find`, é possível encontrar todos os programas SUID e SGID do sistema:

```
# find / -perm -4000 -print0 | xargs -0 ls -l
# find / -perm -2000 -print0 | xargs -0 ls -l
```

Além dos programas SUID e SGID, é necessário localizar todos os executáveis presentes nos diretórios dos usuários e nos diretórios mais comumente utilizados pelos atacantes (`/tmp`, `/usr/tmp`, `/var/tmp` e `/dev`, por exemplo). É importante observar que os atacantes podem "camuflar" seus executáveis, desabilitando a permissão de execução dos arquivos. Desse modo, uma combinação dos comandos `find`, `file` e `grep` pode ser usada como segue[20]:

```
# find /home /tmp /usr/tmp /var/tmp /root /dev -type f -print0 | xargs \
  -0 file -zL | grep -E 'executable|script|perl'
```

A.14.2 Identificação dos executáveis e bibliotecas suspeitas

Obviamente é necessário mais do que simplesmente listar os executáveis do sistema. Uma das primeiras coisas a se fazer é identificar a presença de *trojan horses* instalados, checando a integridade dos executáveis mais comumente visados pelos atacantes como, por exemplo, `ps`, `netstat`, `ls`, `login`, `in.telnetd` e `ssh` (em geral todos os executáveis localizados em `/bin`, `/sbin`, `/usr/bin` e `/usr/sbin` devem ser verificados) [47]. Essa checagem pode ser feita através dos *mactimes* dos arquivos (*ctime* ou *mtime* próximos do período da invasão) ou de *hashes* criptográficos, utilizando-se ferramentas automatizadas como o Tripwire ou "manualmente" com o comando `md5sum` (comparando-se com cópias confiáveis). Outra forma de checar a consistência dos executáveis instalados é através do comando `rpm`, como apresentado anteriormente.

Além de verificar os executáveis encontrados, é importante também notar a ausência ou adição de novos programas no sistema. Em conjunto com os *trojan horses* identificados, alguns executáveis certamente levantam suspeitas e necessitam de uma análise detalhada, incluindo, por exemplo:

[20] A combinação dos comandos `find`, `file` e `grep` é uma abordagem lenta, porém confiável, para identificar os executáveis do sistema.

- aqueles localizados nos diretórios mais comumente utilizados pelos atacantes;
- aqueles com nomes suspeitos (ocultos, pouco usuais, que fazem alusão a atividades maliciosas ou que representam ferramentas conhecidamente utilizadas pelos invasores);
- executáveis com a permissão de execução desativada;
- e programas SUID ou SGID *root*, em especial aqueles localizados fora dos diretórios padrão de executáveis;

A.14.3 Análise dos executáveis e bibliotecas

Os executáveis e bibliotecas identificados como suspeitos devem ser analisados para determinar se são hostis e quais funcionalidades são implementadas. Algumas vezes, os próprios nome e diretório do arquivo podem fornecer pistas úteis. Muitos atacantes nem mesmo se preocupam em modificar os nomes de suas ferramentas, de modo que se o executável suspeito tiver nomes do tipo *sniffer* ou *cracker*, existem boas chances do mesmo implemetar a funcionalidade sugerida por seu nome (uma busca na Internet pelo nome do arquivo pode revelar pistas valiosas) [47]. Entretanto, existem outras formas de analisar os executáveis e bibliotecas suspeitos, conforme descrito a seguir.

Tentar ler o arquivo deve ser a primeira abordagem. Se o código suspeito é um *script*, é possível analisar os comandos e determinar sua funcionalidade. Entretanto, se o código é um binário, pouco se pode fazer nesse sentido, a menos que o código-fonte esteja disponível. Geralmente os atacantes compilam suas ferramentas na máquina invadida, de modo que o código-fonte pode ainda estar presente no sistema (geralmente nas proximidades do executável ou nos diretórios mais comumente utilizados pelos atacantes, como `/tmp`) ou pode ter sido "deletado" e ainda estar disponível para recuperação (segundo as técnicas descritas anteriormente).

Caso o código-fonte não esteja disponível, é possível analisar as cadeias de caracteres contidas nos binários, através do comando `strings`, permitindo revelar informações sobre os arquivos que ele acessa, suas mensagens de erro e ajuda, nomes de funções e bibliotecas utilizadas, entre outros dados que podem fornecer pistas sobre a funcionalidade do executável. Em alguns casos é possível encontrar palavras comumente utilizadas pelos atacantes, na forma de senhas (a palavra "sartori", por exemplo, é usada como senha em alguns *rootkits*), gírias (a palavra "warez", por exemplo) ou nomes de ferramentas hostis (uma busca na Internet[21] pode revelar muitas dessas palavras). Desse modo, é possível utilizar o comando `strings` em conjunto com o utilitário `grep` para procurar palavras-chave,

[21] Em *sites* como, por exemplo, o localizado na URL http://www.phrack.com (disponível em janeiro de 2003).

como as mencionadas anteriormente ou ainda nomes de arquivos específicos e mensagens produzidas no terminal.

Além do comando `strings`, a ferramenta `nm` pode ser usada para listar a tabela de símbolos do programa suspeito. Através das opções `-Du`, do comando `nm`, é possível visualizar as bibliotecas utilizadas pelo binário suspeito, o que pode fornecer pistas como, por exemplo, se o código desconhecido pode abrir um *socket* ou utiliza algum tipo de cifragem. Técnicas de engenharia reversa também podem ser aplicadas, embora não seja uma tarefa trivial, para tentar recontruir o código-fonte do binário suspeito [47].

Uma última abordagem para a análise do código suspeito é sua execução em um ambiente devidamente preparado e controlado, de modo que seja possível rastrear a execução do programa e monitorar as alterações causadas no sistema, limitando ao máximo a possibilidade de estragos fora do ambiente controlado. Algumas etapas são necessárias para a execução do código suspeito, como listadas a seguir:

- preparar o ambiente de execução com o mesmo sistema operacional e plataforma da máquina invadida. Não utilizar o sistema de análise, muito menos a própria máquina invadida, para conduzir o teste do código suspeito. É possível utilizar o VMWare[22] com a opção *nonpersistent writes* habilitada, para impedir que o programa suspeito escreva no disco [67]. Criar um segmento de rede isolado para o ambiente de execução, instalando um *sniffer* em um sistema separado, para monitorar o tráfego de rede nesse segmento;

- produzir uma imagem da condição inicial do ambiente de execução, com *hashes* criptográficos e *mactimes* dos principais arquivos do sistema;

- executar o código suspeito e interceptar as chamadas de sistema e de biblioteca feitas pelo processo (através das ferramentas `strace` e `ltrace`) [67]. Além de executar o programa, é possível depurá-lo através de ferramentas como o `gdb`, permitindo observar passo a passo cada ação executada pelo código suspeito;

- analisar as chamadas de sistema executadas, principalmente as funções `open`, `read`, `write`, `unlink`, `lstat`, `socket` e `close` [67], bem como as bibliotecas acessadas. Observar o estado das conexões de rede e das portas abertas no ambiente de execução (através do comando `netstat`), além dos processos em execução (por meio do comando `ps`) e os arquivos abertos por eles (através do utilitário `lsof`). Determinar as alterações ocorridas no ambiente de execução, utilizando a imagem produzida anteriormente, e avaliar os dados coletados pelo *sniffer* (se for o caso);

[22] Informações sobre o VMWare podem ser encontradas na URL http://www.vmware.com (disponível em janeiro de 2003).

A.15 Transmissão das informações coletadas através da rede

A transmissão das informações pela rede pode ser feita através do comando `nc` (Netcat[23]), em um procedimento que envolve dois passos. No primeiro passo, o sistema de análise deve ser preparado para receber as informações, através da execução do comando `nc` com as opções `-l` e `-p`, como ilustrado a seguir:

```
# nc -l -p 2222 | tee ps.out | md5sum -b > ps.out.md5
```

No exemplo anterior, o sistema de análise é configurado para receber as informações pela porta 2222, armazená-las no arquivo `ps.out` e gerar o *hash* criptográfico das informações recebidas, armazenando-o no arquivo `ps.out.md5`. É utilizada uma combinação dos comandos `tee` e `md5sum`, que permite armazenar as informações recebidas e gerar seu *hash* criptográfico no mesmo instante. O comando `tee` simplesmente recebe um fluxo de dados pela entrada padrão e o armazena em um arquivo, replicando de maneira exata, para a saída padrão, o fluxo de dados recebido. Maiores detalhes sobre os comandos envolvidos no exemplo anterior podem ser encontrados nas respectivas *man pages*.

No segundo passo do procedimento de transmissão das informações pela rede, o comando desejado é executado no sistema suspeito e sua saída é redirecionada para o programa `nc`, informando o endereço IP do sistema de análise e o número da porta configurada para receber os dados, como ilustrado a seguir (no exemplo seguinte, os programas confiáveis são acessados, no sistema suspeito, a partir de um CDROM, montado em `/mnt/cdrom`):

```
# /mnt/cdrom/toolkit/ps -aux | /mnt/cdrom/toolkit/nc 10.1.1.1 2222
```

É importante observar que o *hash* criptográfico gerado no sistema de análise (pela combinação dos comandos `tee` e `md5sum`), ao receber as informações provenientes do comando executado na máquina analisada, permite apenas checar futuramente a integridade dos dados armazenados na máquina de análise. A autenticidade desses dados só pode ser garantida se tal *hash* criptográfico for comparado com o *hash* dos dados originais, produzidos pela execução do comando no sistema suspeito. Devido à alta volatilidade das informações originais, uma nova execução do comando provavelmente irá gerar dados diferentes, resultando em um *hash* criptográfico completamente distinto. Por esse motivo, a geração do *hash* criptográfico dos dados enviados para o sistema de análise deve ser feita no momento em que as informações são coletadas pelo comando executado. Isso pode ser feito através do mesmo princípio utilizado pelo comando `tee`, de modo que o

[23] O programa Netcat e maiores detalhes sobre o mesmo podem ser encontrados na URL http://www.atstake.com/research/tools (disponível em janeiro de 2003).

fluxo de dados gerado pelo comando de coleta é transmitido para o sistema de análise e replicado exatamente para o programa responsável por gerar o *hash* criptográfico. Foi implementado, como parte deste trabalho, um pequeno *script* na linguagem Perl, chamado `tee_command.pl`[24], que executa a funcionalidade descrita anteriormente, podendo ser usado na máquina suspeita da seguinte forma:

```
# /mnt/cdrom/toolkit/ps -aux | /mnt/cdrom/toolkit/perl \
  /mnt/cdrom/toolkit/tee_command.pl "/mnt/cdrom/toolkit/nc 10.1.1.1 \
  2222" | /mnt/cdrom/toolkit/md5sum -b
```

Essa abordagem permite ainda provar que a trasmissão dos dados pela rede não introduziu qualquer tipo de alteração nas informações coletadas. Para a geração dos *hashes* criptográficos, é importante que o fluxo de dados seja sempre o mesmo, desde a coleta na máquina suspeita até seu armazenamento no sistema de análise, como ilustrado pela Figura A.3.

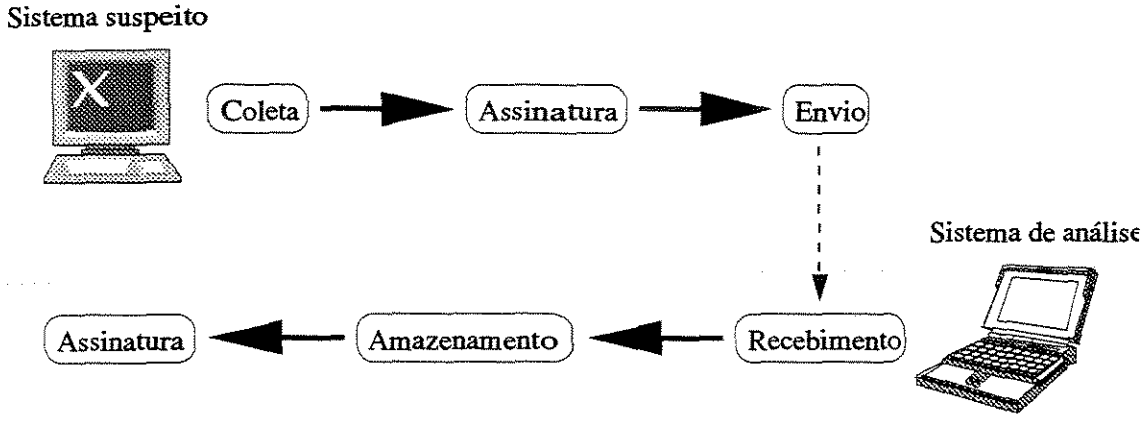


Figura A.3: Fluxo das informações voláteis desde a coleta no sistema suspeito até seu armazenamento no sistema de análise.

A.16 Procedimentos de imagem dos discos suspeitos

As diversas abordagens para o procedimento de imagem são detalhadas na sequência, utilizando-se o comando `dd` e o disco suspeito instalado na interface IDE secundária mestre do sistema de análise (acessível através do *device file* `/dev/hdc`).

[24] O *script* `tee_command.pl` pode ser obtido na URL http://www.ic.unicamp.br/~ra000504 (disponível em janeiro de 2003).

A.16.1 Imagem em arquivo de todo o disco

Esta abordagem é bastante simples e pode ser executada da seguinte forma:

```
# dd if=/dev/hdc of=suspect_img.dd
19920600+0 records in
19920600+0 records out
```

No exemplo anterior, a imagem do disco suspeito é armazenada no arquivo `suspect_img.dd`. Pode-se utilizar as opções `noerror` e `sync`, do comando `dd`, para o caso de existirem setores defeituosos no disco suspeito. Uma prática recomendada é a utilização de nomes, para as imagens em arquivo, que identifiquem precisamente o disco copiado. Após a geração da imagem é necessário verificar se os dados foram copiados com exatidão, o que pode ser feito através de *hashes* criptográficos, como segue:

```
# dd if=/dev/hdc | md5sum -b
19920600+0 records in
19920600+0 records out
54d3ca2fe9b2e61b6d4b35580b42b6ba *-
# dd if=suspect_img.dd | md5sum -b
54d3ca2fe9b2e61b6d4b35580b42b6ba *-
```

No exemplo anterior, os *hashes* criptográficos MD5 do disco suspeito e de sua imagem são idênticos, garantindo a autenticidade e integridade da cópia produzida. Devido ao grande volume de dados contido nos discos, a geração do *hash* criptográfico do disco suspeito pode ser feita em conjunto com a produção de sua imagem, através do comando `tee`:

```
# dd if=/dev/hdc | tee suspect_img.dd | md5sum -b
19920600+0 records in
19920600+0 records out
54d3ca2fe9b2e61b6d4b35580b42b6ba *-
# md5sum -b suspect_img.dd
54d3ca2fe9b2e61b6d4b35580b42b6ba *-
```

O procedimento do exemplo anterior adiciona um atraso na geração da imagem do disco suspeito, entretanto, reduz consideravelmente o tempo gasto para checar os *hashes* criptográficos, pois o disco suspeito é lido apenas uma vez.

A.16.2 Imagem em arquivo das partições separadas

Caso o investigador deseje produzir uma imagem em arquivo para cada partição do disco suspeito, é possível utilizar as opções `skip` e `count` do comando `dd` para indicar o início e fim de cada partição, como ilustrado a seguir:

```
# fdisk -lu /dev/hdc

Disk /dev/hdc: 255 heads, 63 sectors, 1240 cylinders
Units = sectors of 1 * 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hdc1   *          63     9221309     4610623+   c  Win95 FAT32 (LBA)
/dev/hdc2         9221310    18040994     4409842+  83  Linux
/dev/hdc3        18040995    18458684      208845   82  Linux swap


# dd if=/dev/hdc of=suspect_imgINI.dd bs=512 count=63
63+0 records in
63+0 records out
# dd if=/dev/hdc of=suspect_img1.dd bs=512 skip=63 count=9221247
9221247+0 records in
9221247+0 records out
# dd if=/dev/hdc of=suspect_img2.dd bs=512 skip=9221310 count=8819685
8819685+0 records in
8819685+0 records out
# dd if=/dev/hdc of=suspect_img3.dd bs=512 skip=18040995 count=417690
417690+0 records in
417690+0 records out
# dd if=/dev/hdc of=suspect_imgFIM.dd bs=512 skip=18458685
1461915+0 records in
1461915+0 records out
```

No exemplo anterior, o disco suspeito possui três partições, que não ocupam todo o disco. A imagem de cada partição é armazenada respectivamente nos arquivos `suspect_img1.dd`, `suspect_img2.dd` e `suspect_img3.dd`. Além disso, os espaços não utilizados do disco suspeito também são coletados, como os espaços anterior à primeira partição e posterior à terceira, armazenados respectivamente em `suspect_imgINI.dd` e `suspect_imgFIM.dd`.

A.16.3 Espelhamento

Para o procedimento de espelhamento do disco suspeito, o comando `dd` pode ser usado para "esterilizar" o disco espelho, eliminando qualquer informação que possa ser confundida com os dados do disco suspeito. Isso pode ser feito da seguinte forma:

```
# dd if=/dev/zero of=/dev/hdb
```

No exemplo anterior, o disco espelho, acessado através do *device file* `/dev/hdb`, é inteiramente preenchido com zeros. A limpeza do disco espelho pode ser verificada através da combinação dos comandos `xxd` e `grep`, como segue:

```
dd if=/dev/hdb | xxd | grep -v "0000 0000 0000 0000 0000 0000 0000 0000"
```

Caso a combinação de comandos anterior produza alguma saída (com exceção da saída normal do comando `dd`), o disco espelho não foi devidamente "esterilizado". Depois de preparado o disco destino, o espelhamento pode ser executado de duas maneiras, como ilustrado no exemplo a seguir:

```
# dd if=/dev/hdc of=/dev/hdb
# dd if=suspect_img.dd of=/dev/hdb
```

Na primeira execução do comando `dd`, no exemplo anterior, o espelhamento é feito diretamente do disco suspeito (`/dev/hdc`) para o disco espelho (`/dev/hdb`). É preciso extrema atenção nesse caso, pois uma inversão nas opções do comando `dd` pode ser desastrosa. Na segunda execução, a imagem em arquivo do disco suspeito, produzida em um momento anterior, é utilizada para gerar o disco espelho, de maneira mais segura.

Apêndice B

Conjunto de ferramentas para análise forense em sistemas computacionais

Além de experiência e sólidos conhecimentos, o investigador depende totalmente do conjunto de ferramentas que ele utiliza para coletar, documentar, preservar e processar as informações provenientes do sistema investigado. No dia-a-dia, o investigador deve estar preparado para lidar com as mais diversas arquiteturas e sistemas operacionais, logo, seu conjunto de ferramentas deve ser o mais amplo possível.

Independente do tipo de ferramenta à disposição do investigador, é preciso estar familiarizado com seu funcionamento e verificar se elas executam exatamente aquilo que se espera. O investigador deve ser capaz de interpretar com confiança a saída dos programas e deve estar ciente de todas as implicações que eles podem causar no sistema ou informação analisada. Praticar e testar as ferramentas é tão importante quanto adquiri-las.

Na sequência são apresentados alguns dos principais itens que devem compor o conjunto de ferramentas do investigador.

B.1 Sistema de análise

O conjunto de ferramentas de investigação deve conter um sistema de análise, que é a máquina onde idealmente a análise das informações coletadas é realizada. Esse sistema representa um ambiente controlado onde o investigador dispõe de todos os componentes de *hardware* e utilitários de *software* necessários para coletar, receber, preservar e analisar as informações digitais provenientes do sistema suspeito. O ideal é que seja uma máquina portátil, com alta capacidade de processamento e armazenagem de dados, com interface de rede e que permita a conexão dos mais variados tipos de periféricos e dispositivos como, por exemplo, unidades de CDROM, *zip drives* e discos rígidos. Existem computadores especialmente desenvolvidos para a prática forense, com configurações que

permitem acessar informações através de diferentes tipos de tecnologias. Um exemplo é a unidade portátil fornecida pela Forensic-Computers.com[1], ilustrada na Figura B.1.

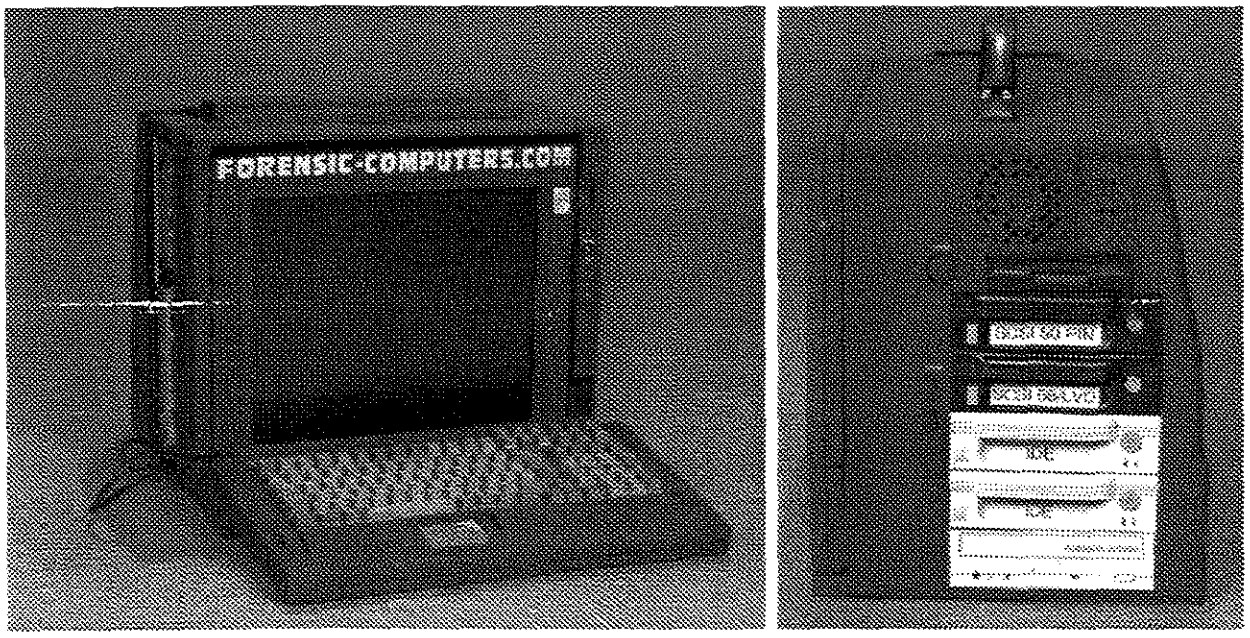


Figura B.1: Unidade portátil de investigação forense da Forensic-Computers.com.

Quanto ao sistema operacional da máquina de análise, cada investigador possui uma preferência, orientada principalmente pela familiaridade, facilidade de uso, disponibilidade de utilitários e capacidade de acesso a informações provenientes de outras plataformas. O ambiente Linux, por exemplo, oferece suporte a uma grande variedade de sistemas de arquivos, de modo que é possível a utilização de um único conjunto de utilitários para análise de diferentes plataformas. No entanto, algumas situações requerem a utilização da mesma plataforma encontrada no sistema suspeito. É importante ter sempre disponíveis as mídias de instalação dos principais sistemas operacionais, em suas mais variadas versões, permitindo recriar as mesmas condições encontradas no sistema suspeito. Pode-se, ainda, instalar múltiplos sistemas operacionais na máquina de análise, possibilitando a escolha durante a inicialização do sistema ou através de programas como o VMWare.

[1]Maiores informações sobre dispositivos de *hardware* para a prática forense podem ser encontradas na URL http://www.forensic-computers.com (disponível em janeiro de 2003).

B.2 Ferramentas auxiliares

No conjunto de ferramentas do investigador não pode faltar, obviamente, itens auxiliares para desmontagem das máquinas; identificação, transporte e armazenamento de materiais coletados; e documentação das atividades. Como exemplos, podem ser citados os seguintes itens: chaves para diversos tipos de parafusos, alicates, extensões e filtros de linha, cabos e conectores de rede, etiquetas e formulários, plástico-bolha, embalagem contra eletricidade estática, máquina fotográfica, mídias removíveis e discos rígidos.

B.3 Ferramentas para utilização no sistema suspeito

Muitas vezes a coleta de informações, e até mesmo a análise, precisa ser feita diretamente no sistema comprometido. Os atacantes frequentemente alteram os programas contidos na máquina invadida e, portanto, o investigador necessita de um conjunto de utilitários confiáveis para conduzir a investigação diretamente no sistema suspeito. Uma solução é a utilização de um CDROM ISO 9660, que virtualmente qualquer plataforma pode acessar, contendo todos os utilitários (binários e *scripts*), destinados a diversas plataformas, que o investigador possa precisar. Além disso, é preciso que os binários sejam compilados estaticamente ou que o CDROM contenha também cópias confiáveis das bibliotecas dinâmicas, para evitar problemas com bibliotecas comprometidas no sistema suspeito. Existem iniciativas voluntárias no sentido de se criar um conjunto de utilitários[2], compilados estaticamente para diferentes plataformas, apropriados para a prática forense. Como sugestão de ferramentas que podem compor esse CDROM, podem ser citados os seguintes programas:

ls	strings	rpm	ps	ltrace	route	what	telnet
cp	grep	gzip	kill	uptime	rndc	lastcomm	ssh
cd	find	fdisk	ftp	netstat	lsmod	lastlog	TCT
cat	file	dd	gdb	nfsstat	kstat	perl	TCTUTILs
more	diff	xxd	top	arp	md5sum	bash	
less	stat	losetup	lsof	tcpdump	pwck	tee	
vi	tar	mount	strace	ifconfig	grpck	nc	

Outra ferramenta muitas vezes necessária é uma distribuição de algum sistema operacional que caiba em uma mídia removível (em alguns disquetes ou em um CDROM, por exemplo). É importante que tal distribuição não execute qualquer operação sobre os dispositivos de armazenagem secundária (como a criação de áreas de *swap*, por exemplo),

[2]Tal conjunto de ferramentas pode ser encontrado na URL http://www.incident-response.org (disponível em janeiro de 2003).

tenha suporte à comunicação por rede e permita a criação de imagens dos discos [67]. Existem algumas distribuições compactas do sistema Linux que podem ser usadas, como, por exemplo, o *Pocket-Linux*, *Tomsrtbt*, *Trinux* e *Linuxcare Bootable Toolbox*[3].

B.4 Utilitários de forense

Ao longo do Apêndice A são apresentados diversos utilitários, conhecidos dos usuários e administradores de sistemas Linux, que originalmente não foram desenvolvidos para o propósito único da análise forense. Um aspecto importante da prática forense é o uso de ferramentas não necessariamente destinadas a essa finalidade [47]. Entretanto, algumas necessidades singulares resultaram no desenvolvimento de ferramentas próprias para a investigação forense. No restante desta seção, algumas dessas ferramentas de forense são apresentadas. Entre elas, o TCT e o TCTUTILs já foram mencionadas anteriormente no Apêndice A. Apesar do escopo deste trabalho restringir-se ao ambiente Linux, algumas ferramentas destinadas a outras plataformas também são apresentadas, devido a sua ampla utilização e relevência na área da forense computacional. É importante mencionar, ainda, que existem algumas ferramentas disponíveis apenas para mantenedores da lei e agências governamentais e, portanto, não são discutidas nesta seção.

B.4.1 *The Coroner's Toolkit* (TCT)

O TCT[4] é um conjunto de ferramentas de código aberto e gratuito, desenvolvido por Dan Farmer e Wietse Venema, que permite investigar um sistema UNIX comprometido. É importante mencionar que o TCT não foi inicialmene desenvolvido para apresentar evidências úteis em um processo criminal, e sim para ajudar a determinar o que aconteceu em um sistema invadido [47]. O TCT apresenta quatro partes principais:

- `grave-robber`;
- `mactime`;
- utilitários (`icat`, `ils`, `pcat`, `md5`, `timeout`);
- `unrm` e `lazarus`;

[3]As distribuições compactas do sistema Linux — *Pocket-Linux*, *Tomsrtbt*, *Trinux* e *Linuxcare Bootable Toolbox* — podem ser encontradas, respectivamente, nas URLs http://www.tux.org/pub/distributions/tinylinux/pocket-linux/, http://www.toms.net/rb/, http://trinux.sourceforge.net, http://lbt.linuxcare.com/ (todas disponíveis em janeiro de 2003).

[4]O TCT pode ser encontrado na URL http://www.porcupine.org/forensics/tct.html (disponível em janeiro de 2003).

Os principais componentes do TCT são detalhados na sequência. No entanto, a maioria das ferramentas possui uma série de opções de execução, que não são discutidas neste trabalho. Além disso, alguns componentes do TCT possuem restrições quanto ao sistema de arquivos e plataforma suportados, que também não são abordadas. Maiores detalhes sobre as ferramentas do TCT, suas opções e compatibilidades podem ser encontradas nas respectivas *man pages*.

Grave-robber

A ferramenta `grave-robber` pode ser considerada a parte central de todo o TCT. O `grave-robber` é basicamente uma ferramenta de coleta de dados que executa uma série de comandos em uma tentativa de reunir informações básicas sobre o sistema e salvar alguns arquivos necessários para a análise. É importante mencionar que o `grave-robber` apenas coleta informações, não fazendo qualquer esforço no sentido de interpretá-las.

A ferramenta captura os dados de acordo com a ordem de volatilidade (conceito introduzido por Farmer e Venema) e, como padrão de execução, ela coleta informações efêmeras (processos e estado da rede), descobre o que puder sobre a configuração de *hardware* do sistema (especialmente sobre os discos e partições) e busca por arquivos críticos (como, por exemplo, arquivos de *log* e de configuração).

Antes de iniciar a coleta dos dados, o `grave-robber` examina alguns dos utilitários e arquivos mais importantes do sistema, permitindo que o investigador possa utilizá-los com segurança. Usualmente são verificados os arquivos contidos no diretório `/etc` e os utilitários comumente localizados em diretórios como `/bin` e `/usr/bin`. Essa atividade é controlada pelo arquivo de configuração `look@first` do TCT.

Embora seja um processo bastante demorado, recomenda-se executar o `grave-robber` sobre o sistema todo (passando, como argumento, o diretório raiz do sistema). Como saída, o `grave-robber` produz os seguintes arquivos e diretórios (toda saída produzida contém uma marca de tempo e é gerado o *hash* criptográfico MD5 do arquivo produzido):

- `body`: banco de dados para o programa `mactime`, contendo os atributos dos arquivos analisados (incluindo os *mactimes*);
- `body.S`: contém os atributos de todos os arquivos SUID encontrados;
- `command_out`: sub-diretório contendo a saída dos diversos comandos executados pelo `grave-robber` como, por exemplo, o `ps` e `netstat`;
- `removed_but_running`: sub-diretório contendo arquivos que foram "deletados", mas ainda estão em execução;
- `icat`: sub-diretório contendo arquivos em execução que ainda estão no disco;

- `proc`: sub-diretório contendo arquivos copiados do diretório `/proc`;
- `conf_vault`: sub-diretório contendo os arquivos e diretórios críticos salvos pelo `grave-robber`;
- `user_vault`: sub-diretório contendo os arquivos e diretórios de usuário salvos pelo `grave-robber` (como os *history files*, por exemplo);
- `trust`: sub-diretório contendo relações de confiança do sistema (como os arquivos `.rhosts` e `.forward` e as tarefas agendadas, por exemplo);
- `coroner.log`: *log* de execução do `grave-robber`, contendo a data e horário de execução de todos os programas;
- `error.log`: *log* de erro do `grave-robber`;
- `MD5_all`: *hash* MD5 de tudo que é produzido como saída do programa `grave-robber`;

Mactime

A ferramenta `mactime` pode ser usada para processar o arquivo `body`, produzido pelo `grave-robber`, ou pode ser executada sobre um determinado diretório. Essa ferramenta produz uma listagem de todos os arquivos e diretórios que tiveram algum de seus *mactimes* alterados a partir de uma determinada data (passada como parâmetro). Tal listagem é ordenada segundo uma linha de tempo e cada entrada corresponde à alteração de um ou mais *mactimes* (identificados pelas letras m, a ou c), como, por exemplo:

```
(data      horário  tamanho  MAC  permissões  UID   GID   arquivo)
Apr 05 99  04:05:00  64092   m..  -r-xr-xr-x  root  root  /bin/ps
Apr 10 99  04:05:00  45724   m..  -rwxr-xr-x  root  root  /bin/ls
Apr 12 99  01:04:39  45724   .a.  -rwxr-xr-x  root  root  /bin/ls
Apr 12 99  14:10:15  14716   m.c  -rwxr-xr-x  root  root  /bin/cat
```

Utilitários

O TCT possui uma série de utilitários que são executados pela ferramenta `grave-robber`. Entretanto, esses programas auxiliares podem ser bastante úteis em outras situações, como apresentado no Apêndice A. Alguns dos principais utilitários presentes no TCT são descritos como segue:

- `icat`: permite visualizar o conteúdo de um arquivo ou diretório a partir do número de seu *inode*. Pode ser usado para recuperar o conteúdo de *inodes* não alocados;

- `ils`: lista as informações sobre os *inodes* de um sistema de arquivos. Em sua execução padrão o `ils` lista apenas informações sobre *inodes* de arquivos "deletados";
- `ils2mac`: *script* usado para converter a saída do comando `ils` para o formato do arquivo `body`, permitindo sua utilização com a ferramenta `mactime`;
- `pcat`: permite visualizar a memória de um processo;
- `md5`: computa o *hash* criptográfico MD5 de um fluxo de bits qualquer (um arquivo, por exemplo);
- `timeout`: permite a execução de um comando qualquer com uma restrição de tempo. Caso o comando executado não termine dentro do limite de tempo estipulado, ele é finalizado;

Unrm e lazarus

A ferramenta `unrm` é uma variante do programa `dd`, sendo capaz de copiar os blocos de dados de um sistema de arquivos. Em sua execução padrão, o `unrm` extrai apenas os blocos de dados não alocados do sistema de arquivos. Essa ferramenta é utilizada na tentativa de recuperar informações "deletadas". É importante redirecionar a saída do programa `unrm` para outro sistema de arquivos diferente do analisado. Caso contrário, o fluxo de bits gerado irá ocupar os blocos não alocados que deveriam ser extraídos, possivelmente sobrescrevendo as informações procuradas.

O resultado da execução do programa `unrm` é apenas um enorme fluxo de bits sem qualquer sentido aparente. Para tentar recontruir arquivos "deletados" ou outros tipos de dados, a partir de um fluxo de bits qualquer, o TCT possui uma ferramenta chamada `lazarus`. Esse programa toma os dados produzidos pela ferramenta `unrm` (ou qualquer outra fonte como, por exemplo, a memória e áreas de *swap*) e tenta criar alguma estrutura, organizando os blocos de dados.

Basicamente o processo de análise realizado pelo programa `lazarus` executa os seguintes passos:

1. Leitura de um bloco de dados da entrada (tipicamente são lidos blocos de 1024 bytes);
2. Determinação do formato dos dados contidos no bloco lido — texto ou binário. Isso é feito pela checagem dos dados contidos no primeiro décimo do bloco. Se eles são todos caracteres que podem ser impressos, o conteúdo é classificado como texto, caso contrário o programa assume que se trata de um bloco com dados binários;

3. Se o bloco contém texto, o programa testa os dados contra um conjunto de expressões regulares para tentar determinar do que se trata;

4. Se o bloco contém dados binários, o comando `file` é executado sobre o bloco. Se não há sucesso, os primeiros bytes do bloco são examinados para determinar se os dados parecem estar no formato ELF (representando um binário executável);

5. Se o bloco (contendo dados binários ou texto) é reconhecido nos passos anteriores como sendo de um determinado tipo, ele é marcado como tal. Se esse bloco é de um tipo diferente do bloco processado anteriormente, então ele é salvo como um novo arquivo. Caso contrário, ele é concatenado ao bloco anterior. Se o bloco não é reconhecido nos passos anteriores, mas é posterior a um bloco reconhecido, o programa assume que o bloco em questão é uma continuação dos dados contidos no bloco reconhecido e, portanto, o concatena ao bloco anterior (considerando o Princípio da Localidade);

A saída produzida pelo programa `lazarus` corresponde aos blocos de dados e um mapa dos blocos reconhecidos e seus respectivos tipos. É possível gerar uma saída no formato HTML, contendo um mapa com *links* que permitem navegar pelos dados interpretados, como ilustrado nas Figuras B.2 e B.3.

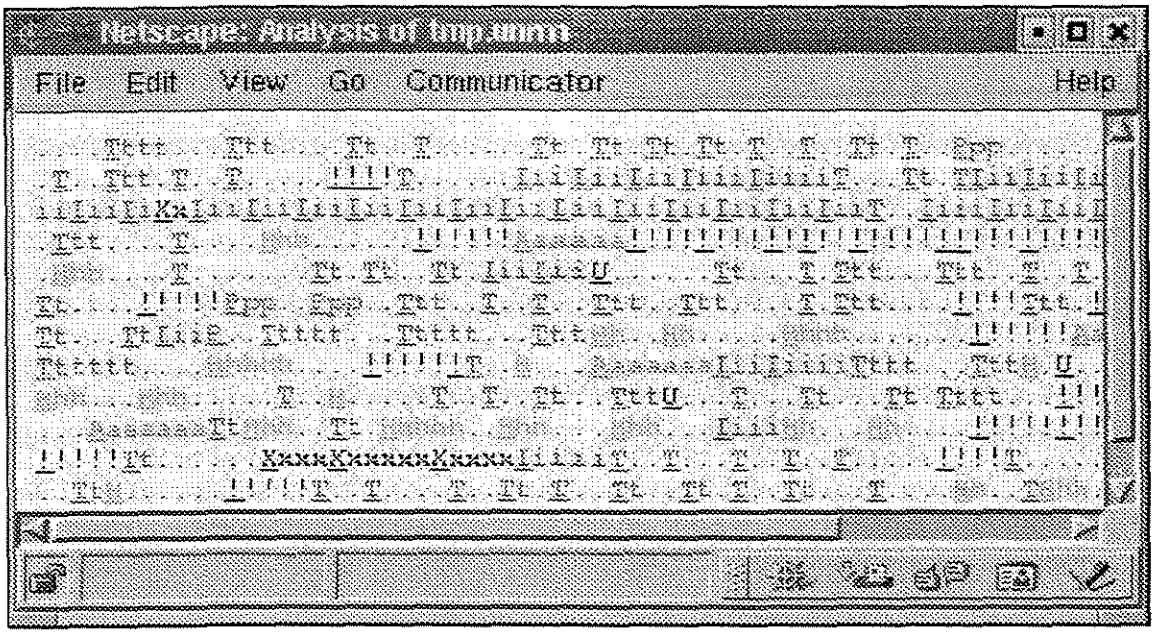


Figura B.2: Mapa dos blocos reconhecidos pelo programa `lazarus`.

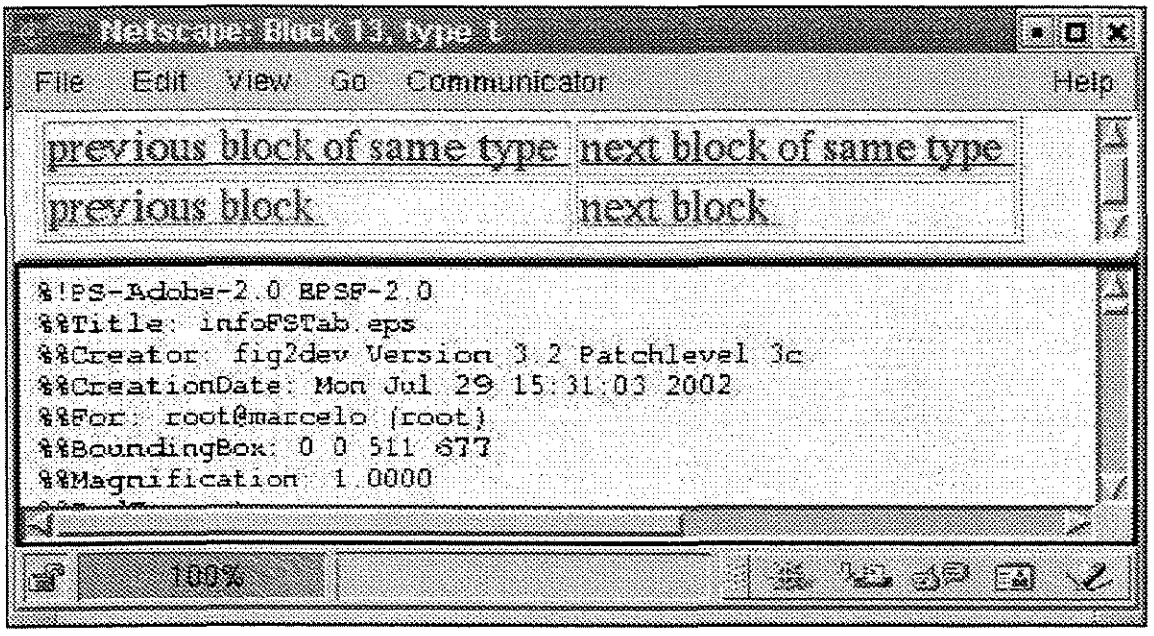


Figura B.3: Navegação pelos blocos interpretados pelo programa `lazarus`.

B.4.2 TCTUTILs

O TCTUTILs[5] é um conjunto de ferramentas de código aberto e gratuito, desenvolvido por Brian Carrier, que utiliza funções e estruturas do TCT para prover funcionalidades extras. Muitas dessas funcionalidades são discutidas no Apêndice A, entretanto, um resumo dos componentes do TCTUTILs é apresentado como segue:

- `bcat`: permite visualizar o conteúdo de um determinado bloco de dados do sistema de arquivos;
- `blockcalc`: cria um mapeamento, para um determinado bloco de dados, entre a imagem original do sistema de arquivos e a imagem contendo apenas os blocos não alocados (produzida pelo programam `unrm` do TCT);
- `fls`: lista as entradas de diretório contidas nos blocos de dados de um determinado diretório. Permite a visualização de entradas referentes a arquivos "deletados";

[5] O TCTUTILs pode ser encontrado na URL http://www.cerias.purdue.edu/homes/carrier/forensics/ (disponível em janeiro de 2003).

- `find_file`: encontra o arquivo correspondente a um determinado *inode*, mesmo que o *inode* não esteja alocado. Permite recuperar, por exemplo, o nome de arquivos "deletados";
- `find_inode`: encontra o *inode* que tem alocado um determinado bloco de dados do sistema de arquivos;
- `istat`: permite visualizar informações sobre um determinado *inode*;

Maiores detalhes sobre as ferramentas do TCTUTILs podem sem encontrados nas respectivas *man pages* ou em [8]. O TCTUTILs, em sua versão 1.01, suporta apenas os sistemas de arquivos FFS e EXT2FS, sendo testado apenas nas plataformas Linux, OpenBSD e Solaris. O TCTUTILs não se encontra mais em desenvolvimento e suas funcionalidades foram todas transferidas para o conjunto de ferramentas chamado *The @stake Sleuth Kit* (TASK), apresentado na sequência.

B.4.3 *The @stake Sleuth Kit* (TASK)

O TASK[6] é um conjunto de ferramentas de código aberto e gratuito, desenvolvido por Brian Carrier, para análise de sistemas de arquivos UNIX (BSDi FFS, FreeBSD FFS, OpenBSD FFS, Solaris FFS, Linux EXT2FS e Linux EXT3FS) e Microsoft (FAT, FAT12, FAT16, FAT32 e NTFS). O TASK integra as ferramentas de análise de sistema de arquivos do TCT com os utilitários do TCTUTILs, além de adicionar novas funcionalidades. Entre suas principais características, segundo o autor, estão a independência de plataforma e o suporte aos sistemas de arquivos NTFS e FAT.

As ferramentas do TASK são organizadas em uma abordagem de camadas e o nome de cada programa, em uma mesma camada, inicia-se com a mesma letra, facilitando a identificação de sua função. Essas camadas incluem o sistema de arquivos como um todo, os blocos de dados (conteúdo dos arquivos), as estruturas de dados do sistema de arquivos (*inodes*, por exemplo) e a interface de interação humana (nomes dos arquivos). Uma breve descrição dos componentes do TASK é apresentada como segue (algumas ferramentas são pertencentes ao TCT ou TCTUTILs, embora apresentem nomes diferentes):

- `dcalc`: corresponde ao `blockcalc` do TCTUTILs;
- `dcat`: corresponde ao `bcat` do TCTUTILs;
- `dls`: corresponde ao `unrm` do TCT;

[6] O TASK pode ser encontrado na URL http://www.atstake.com/research/tools/task (disponível em janeiro de 2003).

- `dstat`: permite visualizar informações sobre um determinado bloco de dados, incluindo, por exemplo, o número do grupo e se o bloco encontra-se alocado;
- `ffind`: corresponde ao `find_file` do TCTUTILs;
- `fls`: corresponde ao `fls` do TCTUTILs;
- `fsstat`: permite visualizar informações detalhadas sobre um sistema de arquivos, incluindo, por exemplo, o nome do volume, a marca de tempo da última montagem e detalhes sobre cada grupo;
- `icat`: corresponde ao `icat` do TCT;
- `ifind`: corresponde ao `find_inode` do TCTUTILs;
- `ils`: corresponde ao `ils` do TCT;
- `istat`: corresponde ao `istat` do TCTUTILs;
- `mactime`: corresponde ao `mactime` do TCT;
- `md5`: corresponde ao `md5` do TCT;
- `sha1`: computa o *hash* criptográfico de um fluxo de bits qualquer, usando o algoritmo SHA-1 [34, 88];

Pequenas alterações, além do suporte a outros sistemas de arquivos, estão presentes em algumas das ferramentas que possuem correspondentes no TCT ou TCTUTILs (apenas o programa `mactime` apresenta modificações substanciais). Maiores detalhes sobre os programas do TASK podem ser encontrados nas respectivas *man pages*.

B.4.4 *Autopsy Forensic Browser* (AFB)

O AFB[7] é uma ferramenta de código aberto e gratuito, desenvolvida por Brian Carrier, que provê uma interface gráfica (ilustrada na Figura B.4) para as ferramentas do TASK, permitindo a análise dos arquivos, diretórios, blocos de dados e *inodes* (alocados ou "deletados") presentes na imagem de um sistema de arquivos (ou no arquivo gerado pelo programa `dls`). Através do AFB, as imagens dos sistemas de arquivos da máquina invadida podem ser examinadas no nível de abstração de arquivos, *inodes* ou blocos de dados. É possível, também, buscar por palavras-chave e expressões regulares, bem como criar uma linha de tempo contendo os *mactimes* dos arquivos e diretórios.

[7] O AFB pode ser encontrado na URL http://www.atstake.com/research/tools/autopsy (disponível em janeiro de 2003).

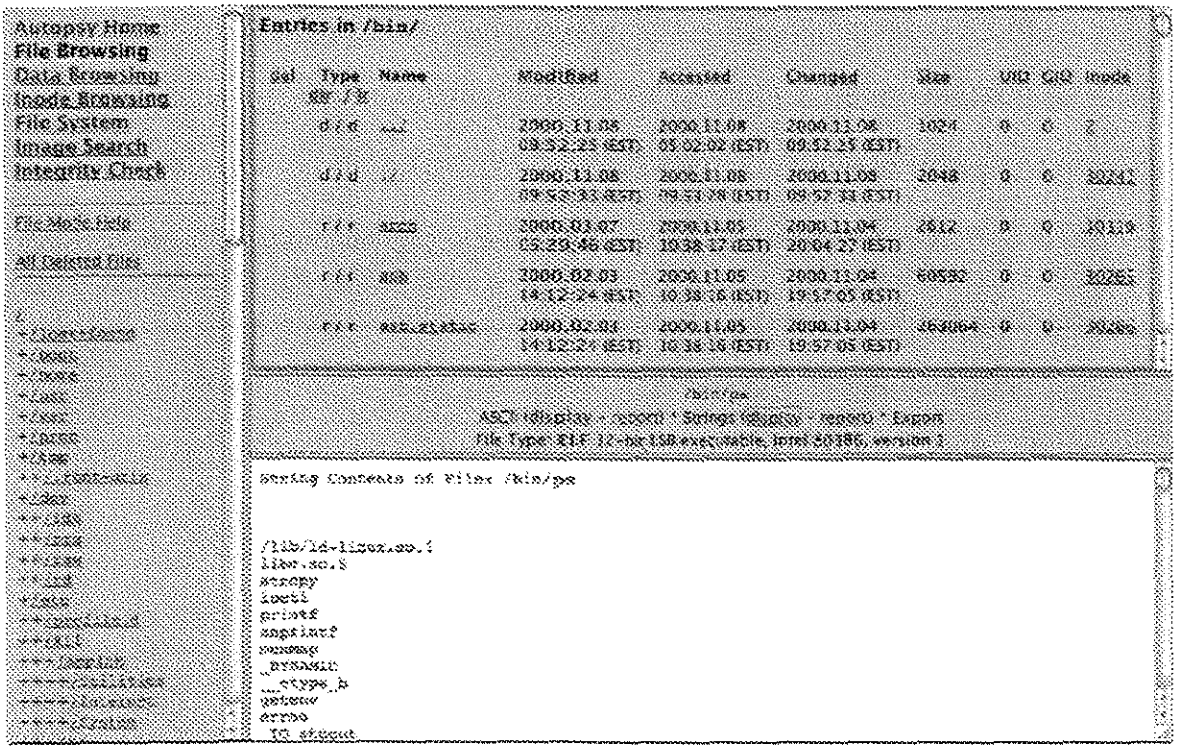


Figura B.4: Exemplo da interface provida pelo *Autopsy Forensic Browser*.

A interface provida pelo AFB é baseada em HTML, segundo um modelo cliente-servidor. O AFB corresponde à parte servidora e qualquer navegador HTML (com suporte a *frames* e *forms*) pode ser usado como cliente. Essa abordagem permite que o AFB seja executado diretamente no sistema comprometido (caso não seja possível extrair imagens dos sistemas de arquivos), por meio de uma mídia removível, fornecendo acesso remoto e protegido contra escritas ao investigador, que utiliza um navegador HTML no sistema de análise.

O autor dos conjuntos de ferramentas TCTUTILs e TASK recomenda que eles sejam utilizados através do AFB[8]. Maiores detalhes sobre o AFB podem ser encontrados na *man page* do mesmo.

[8]No caso do TCTUTILs, o AFB deve ser usado em sua versão 1.01.

B.4.5 ForensiX

A ferramenta ForensiX[9] é um sistema que integra uma coleção de programas de coleta e análise através de uma interface gráfica (ilustrada na Figura B.5). Desenvolvido por Fred Cohen, o ForensiX é baseado no ambiente Linux, explorando muitas de suas vantagens como sistema de análise forense, e foi desenvolvido com o intuito de facilitar, de maneira eficiente e apropriada, a documentação, imagem e exame de informações digitais.

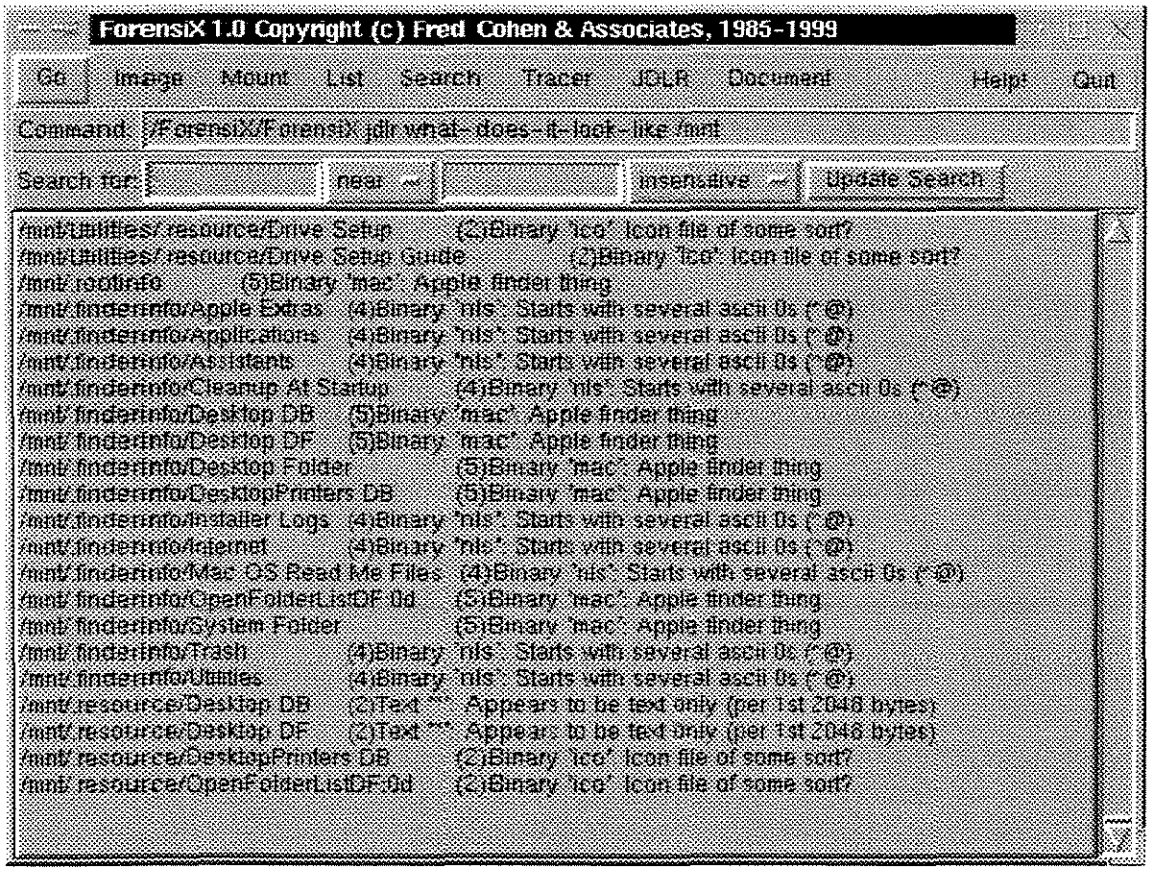


Figura B.5: Exemplo da interface provida pelo ForensiX.

Algumas das principais características do ForensiX são listadas como segue:

- capacidade de produzir imagens a partir de diferentes tipos de fontes de dados, incluindo tráfego IP, disquetes, discos rígidos (IDE e SCSI), CDROMs, cartões PCMCIA e portas paralela e serial;

[9]Maiores informações sobre o ForensiX podem ser encontradas na URL http://www.all.net/ForensiX/index.html (disponível em janeiro de 2003).

- pode armazenar uma imagem em arquivos, discos, fitas e CD-Ws;
- capacidade de montar (com proteção a escritas) imagens e mídias contendo diferentes tipos de sistemas de arquivos, incluindo os dos ambientes Windows, WindowsNT, DOS, UNIX, MacIntosh e PalmOS;
- provê documentação automática das ações e cadeia de custódia das informações;
- permite a reprodução do processo de análise forense;
- provê checagem de integridade das informações;
- permite a busca de palavras-chave e *hashes* criptográficos conhecidos;
- capacidade de identificar o tipo de um arquivo através de seu conteúdo, permitindo encontrar tentativas de esconder informações;
- permite a análise de arquivos "deletados", blocos não alocados, áreas de *swap*, blocos defeituosos e *file slacks*;
- possibilita a busca e visualização de arquivos de imagens gráficas;
- permite checar um sistema UNIX em busca de vulnerabilidades conhecidas;
- capacidade de produzir um *snapshot* de um sistema de arquivos e compará-lo com outros;
- permite a customização e programação de capacidades de análise e busca;

Devido a uma restrição imposta pelo *Digital Millennium Copyright Act* (DMCA)[10], em relação à distribuição de ferramentas de forense digital sem o financiamento explícito de agências mantenedoras da lei, o ForensiX foi forçado a ser retirado temporariamente do mercado.

B.4.6 *New Technologies Incorporated* (NTI)

A empresa NTI[11] estabeleceu-se como uma das maiores vendedoras de *software* para forense. A NTI oferece treinamentos e pacotes de ferramentas para os mais variados propósitos, incluindo resposta a incidentes, proteção de evidências, limpeza de discos e

[10]Maiores detalhes sobre o DMCA podem ser encontrados na URL http://thomas.loc.gov/cgi-bin/query/z?c105:H.R.2281.ENR: (disponível em janeiro de 2003).

[11]Maiores informações sobre a NTI podem ser encontradas na URL http://www.forensics-intl.com (disponível em janeiro de 2003).

produção de imagens. Seus programas são implementados na forma de ferramentas de linha de comando, baseadas no ambiente DOS. Por esse motivo são rápidas e pequenas o suficiente para caber em um disquete, de modo que o sistema suspeito pode ser inicializado com um disco de *boot* do DOS e analisado através da mídia contendo as ferramentas de forense. Algumas das principais ferramentas desenvolvidas pela NTI são descritas como segue:

- `CRCMD5`: valida o conteúdo de um ou mais arquivos;
- `DiskScrub`: realiza a esterilização de um disco, eliminando todos os dados;
- `DiskSig`: checa a integridade de uma imagem;
- `FileList`: cria uma lista dos arquivos de um sistema, ordenados pelo tempo de última utilização;
- `Filter_we`: filtro inteligente que utiliza lógica *fuzzy*;
- `GetFree`: coleta os blocos não alocados de um sistema de arquivos;
- `GetSlack`: coleta os *file slacks*;
- `GetTime`: captura a data e hora do sistema analisado;
- `Net Threat Analyzer`: usado para identificar abusos em contas pela Internet;
- `M-Sweep`: utilitário de limpeza de segurança;
- `NTI-Doc`: programa de documentação da análise;
- `PTable`: analisa e documenta as partições de um disco;
- `Seized`: usado para travar e proteger computadores apreendidos;
- `ShowFL`: usado para analisar uma listagem de arquivos;
- `TextSearch Plus`: busca palavras-chave e arquivos gráficos;
- `SafeBack`: utilitário para produção de imagens de discos;

B.4.7 EnCase

O EnCase[12] é um sistema integrado de análise forense baseado no ambiente Windows, desenvolvido pela *Guidance Software* (provedora de outras ferramentas e treinamento no campo da forense). Assim como as ferramentas da NTI, o EnCase é amplamente utilizado por mantenedores da lei e profissionais de segurança de computadores em todo o mundo [47].

O processo utilizado pelo EnCase começa com a criação das imagens dos discos (disquetes, Zips, Jaz, CDROMs e discos rígidos) relacionados ao caso investigado. Depois da criação das imagens, chamadas de EnCase *evidence files*, pode-se adicioná-las a um único caso (*case file*) e conduzir a análise em todas elas simultaneamente.

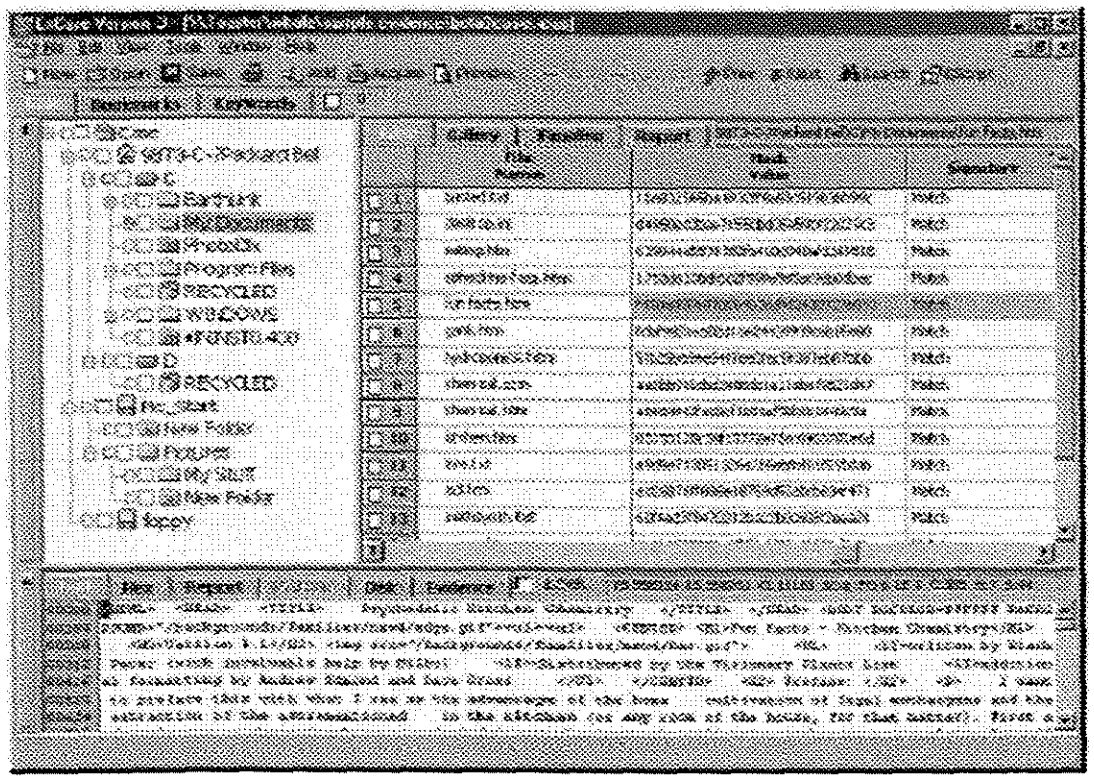


Figura B.6: Exemplo da interface provida pelo EnCase.

O ambiente Windows não é considerado apropriado, por muitos pioneiros da área, para a prática forense, uma vez que ele rotineiramente altera os dados e escreve no disco rígido sempre que é acessado [10]. Entretanto, o EnCase não opera na mídia original ou

[12]Maiores informações sobre o EnCase podem ser encontradas em [10] ou na URL http://www.encase.com (disponível em janeiro de 2003).

discos espelhados. Ao invés disso, o EnCase monta os *evidence files* como discos virtuais protegidos contra escritas. Então, o EnCase (não o sistema operacional) reconstroi o sistema de arquivos contido em cada *evidence file*, permitindo ao investigador visualizar, ordenar e analisar os dados, de maneira não invasiva, através de uma interface gráfica (ilustrada na Figura B.6).

Várias funções e ferramentas de análise são integradas pelo EnCase, podendo ser citadas as seguintes:

- visualização do caso através de uma interface do tipo Windows Explorer;
- suporte a múltiplos sistemas de arquivos, incluindo FAT (12, 16 e 32), NTFS, Macintosh, CDROM, EXT2FS, UFS, PDAs e RAID;
- visualização de todos os arquivos de um caso, incluindo os *file slacks* (no formato texto ou hexadecimal);
- visualização e análise remota dos dados contidos na máquina investigada, através da interface paralela ou da rede;
- busca e visualização de imagens gráficas (mesmo "deletadas");
- mecanismo de busca de palavras-chave e expressões;
- relatório do caso, contendo a descrição dos dados e sua cadeia de custódia;
- criação e checagem de *hashes* criptográficos dos arquivos;
- customização de filtros e funções através de uma linguagem de *scripts*;
- visualização de arquivos compostos, incluindo o registro do Windows, anexos de mensagens de correio eletrônico e arquivos compactados;
- visualização da linha de tempo de utilização dos arquivos;
- identificação do tipo de um arquivo;
- visualização de arquivos de *swap*, *file slacks*, filas e arquivos localizados em *Recycle Bin*;
- mapa gráfico de alocação do disco;

Apêndice C

Standardization of computer forensic protocols and procedures

Este apêndice apresenta o artigo científico *Standardization of Computer Forensic Protocols and Procedures*, de autoria de Marcelo A. Reis e Paulo L. Geus, publicado em *Proceedings of the* 14^{th} *Annual Computer Security Incident Handling Conference*, Waikoloa, Hawaii, USA. Junho, 2002. FIRST.Org, Inc.

C.1 Abstract

In the last few decades, the use of computers has become part of everyday's life. Unfortunately, those who commit crimes are not apart from this computer revolution. As a consequence, procedures which allow investigators to recover data from computers involved in illegal acts and use them as evidence in criminal investigations are becoming fundamental to law enforcement agencies. This work discusses the need to adopt procedures and standards, scientifically evaluated, to conduct forensic examinations in computer systems. To this effect, a standardization model is proposed to develop and to evaluate procedures for the computer forensic science. Some standards are proposed to populate the model and issues surrounding the standardization of forensic procedures are discussed.

C.2 Introduction

In the last few decades, the use of computers has become part of everyday's life. Financial transactions and commerce can be made through the Internet, all sorts of information (from simple personal mail to confidential data) are stored and transmitted electronically, digital devices are present in almost every kind of electronic equipment (from a simple

cd player to the most advanced passenger aircraft), in fact, a true revolution has taken place.

Unfortunately, those who commit crimes are not apart from this computer revolution. The use of computers in criminal activities is a fact and is increasing everyday. In some cases, computers provide the means to consummate the crime. For example, the Internet can be used to spread computer viruses and images of child pornography, or to launch attacks against a computer network. In other cases, computers become storage devices for evidence of crime. For instance, a drug dealer might keep a list of drug suppliers in his/her personal computer, or a money laundering operation might retain false financial records on a network server.

The dramatic increase in computer-related crimes requires law enforcement agencies to work on new techniques for addressing and fighting crime, through training courses and partnerships with scientific research institutions, in order to understand how to obtain and use electronic evidence stored in computers [15]. As a consequence, procedures which allow investigators to recover data from computers involved in illegal acts and use them as evidence in criminal investigations are becoming fundamental to law enforcement agencies. Such procedures must be technologically robust to ensure that all probative information is recovered. Moreover, they must be legally defensible to ensure that nothing in the original evidence was altered [3].

The importance of robust and defensible procedures to conduct a forensic examination can be illustrated by the trial of the american football player O. J. Simpson [20]. Simpson was charged of murdering his ex-wife and a friend of hers. Several blood samples where collected from the crime scene and a DNA analysis showed that they belonged to Simpson. The prosecutors had enough identification evidence to condemn him, however, the defense was able to nullify all DNA evidences and Simpson was declared innocent. The defense strategy was not to contest the probative value of DNA, but to raise suspicions against the procedures used to conclude that Simpson's DNA was in the blood samples from the crime scene, proving that such samples could be contaminated with Simpson's blood during their manipulation.

Simpson's trial motivated american forensic laboratories to revise their procedures in order to correct flaws that could lead to similar situations. The lack of good policies and reliable procedures is a serious problem when the scope comes to computer forensics, since it is a relatively recent forensic discipline and it deals with volatile and latent evidence that exists only in a metaphysical electronic form [3]. A survey conducted by the U.S. Secret Service, in 1995, reported that 70% of the law enforcement agencies that had computer forensic laboratories were doing the work without a written procedures manual [3].

The concern with computer-related crimes is not restricted to the most developed nations. It is a global sense once the use of computers in criminal activities increased

worldwide. The connectivity resulting from the advance of the Internet enables criminals to act transjurisdictionally with ease [4]. Consequently, a criminal may be brought to justice in one country while the digital evidence required to prosecute him may reside in others.

This situation requires that all nations have the ability to collect and preserve digital evidence, not only for their own needs, but also for the potential needs of other countries [4]. Each jurisdiction has its own system of government and administration of justice and once it is not reasonable to expect all nations to know about the laws and rules of other countries, there must be a commom sense that allows the exchange of digital evidence [4]. In this case, procedures used to collect, preserve and analyze digital evidence must be coherent with legal and technical standards of the nations, in order to guarantee that evidence is lawful, defensible and so acceptable.

There are ongoing international efforts to develop examination standards and to provide structure to computer forensics. The formation of the International Organization on Computer Evidence (IOCE) [6] was the first step in order to provide international law enforcement agencies a forum for the exchange of information concerning computer crime. The Scientific Working Group on Digital Evidence (SWGDE) [7] was established as the U.S.-based component of IOCE and was charged with the development of guidelines and standards for digital evidence examination. The work towards standardization has originated a list of principles, discussed in [4], that were approved by IOCE delegates and has been adopted as the draft standard for U.S. law enforcement agencies.

Although research on computer forensic standards and procedures is starting to gain significance there is a lot to work out. Questions regarding, for example, privacy, authorship of digital information and computer forensic tools must be addressed.

This paper proposes an extension to the model for developing guidelines for computer forensic evidence presented in [3]. The new standardization model is detailed in all its aspects and standards are proposed to populate the model. This work is an evolution of the authors' research effort presented in [1,2].

This paper is organized as follows. Section C.3 contains a brief explanation of the computer forensic science. Section C.4 describes, justifies and populates the proposed standardization model. Some issues surrounding the standardization of forensic procedures are discussed in Section C.5. And finally, Section C.6 composes some conclusions about the work presented in this paper.

C.3 Computer Forensic Science

Computer forensic science is the science of acquiring, preserving, retrieving, examining and presenting computer evidence, might it be physical components or data that has been

processed electronically and stored in computer media, in a suitable way for the courts of law [3].

The purpose of the computer examination is to find information related to the case, might it be data stored in files or memory, deleted or not, encrypted or possibly damaged. The search for evidence can be viewed as a detailed scanning within the sources of information of the computer system. Among these sources of information are the file system, log and audit records, free space on the storage device (file slacks and unallocated space, for example), temporary files, swap area, boot sector, memory, registers, peripherals and executing processes (a more detailed analysis of these sources of information can be found in [2]).

Forensic science has produced, all over the years, results that have been judged to be valid and reliable, affecting criminal investigations dramatically and providing compelling testimony in trials [3]. Forensic science relies on the ability of the scientists to produce a report based on the objective results of a scientific examination. As a consequence, forensic results are supported by scientific foundation, what increases their importance as probative information. A mistaken result can condemn an innocent or release a criminal.

In this sense, to support the results of a forensic examination, procedures and protocols are needed to ensure that the resulting information exists on the evidence analyzed and is unaltered by the examination process [3]. Such procedures and protocols must be detailed, documented and peer-reviewed, and acceptable to the relevant scientific community [3].

Computer forensic science is a research area relatively recent, but it is increasing the needs of development in this field, once the use of computers in criminal activities is becoming a common practise. Computers have reached tradicional crimes, such as extortion, robbery and drug dealing, and also have originated a new sort of crime, sometimes called "Internet crimes", such as denial of service attacks, network intrusions and dissemination of computer worms and viruses.

C.4 Standardization Model

The challenge to computer forensic science is to develop and evaluate protocols and procedures that provide valid and defensible results, while protecting the evidence from change. To attain this goal it is important to consider some features that differentiate computer forensic science from most traditional forensic disciplines, such as [3]:

- Computer forensic analysis attempts to recover only probative information from a large volume of data, while some traditional forensic analyses try to gather as much information as possible from an evidence sample;

- Computer forensic science is used most effectively when only the most probative information and details of the investigation are provided to the forensic examiner, in order to reduce the huge search scope. On the other hand, for example, a DNA examination can be conducted without knowledge of specific circumstances of the related crime;

- There commonly is a requirement to perform computer examinations at virtually any physical location, not only in a controlled laboratory setting;

- In the general case, computer forensic science does not need to make interpretive statements as to the accuracy or reliability of the information obtained and normally renders only the information recovered;

- Computer evidence is primarily market-driven and almost never exists in isolation. It is a product of the data stored, the application used to create and store it, and the computer system that directed these activities. As a result, computer forensic science cannot rely on receiving similar evidence in every submission;

- Computer forensic science issues must be adressed in the context of an emerging and rapidly changing environment. In this sense, the science must adapt quickly to new products and inovations with valid and reliable examination techniques;

As a consequence of these features, computer forensic protocols should be written in a hierarchical manner so that common and essential principles remain constant, while examination techniques can adapt quickly to the computer system analyzed [3]. Moreover, computer forensic protocols and procedures should be coherent with legal and technical standards. This is in order to allow the exchange of digital evidence among different jurisdictions in a way that evidence is guaranteed to be reliable and defensible. A forensic examination is a multi-disciplinary activity that attempts to produce suitable results for courts or public forum, and so it must be concerned not only with technical aspects but also with legal issues.

Technical standards refer to the practical issues of the computer forensic examination and must guarantee minimum requirements for the evidence, such as reliability, integrity, accuracy and durability. On the other hand, legal standards are related to the laws and rules of evidence that are essential to the acceptance of results and conclusions by courts and other agencies. As part of legal standards are the legal constraints applied to the seizure of evidence, to the examination process and to the forensic examiner.

To this effect, the authors propose a standardization model, illustrated on Figure C.1, based on the model presented in [3]. The proposed model is an extension of the latter, distinguishing itself by concerns with the legal aspects of forensic science (represented by

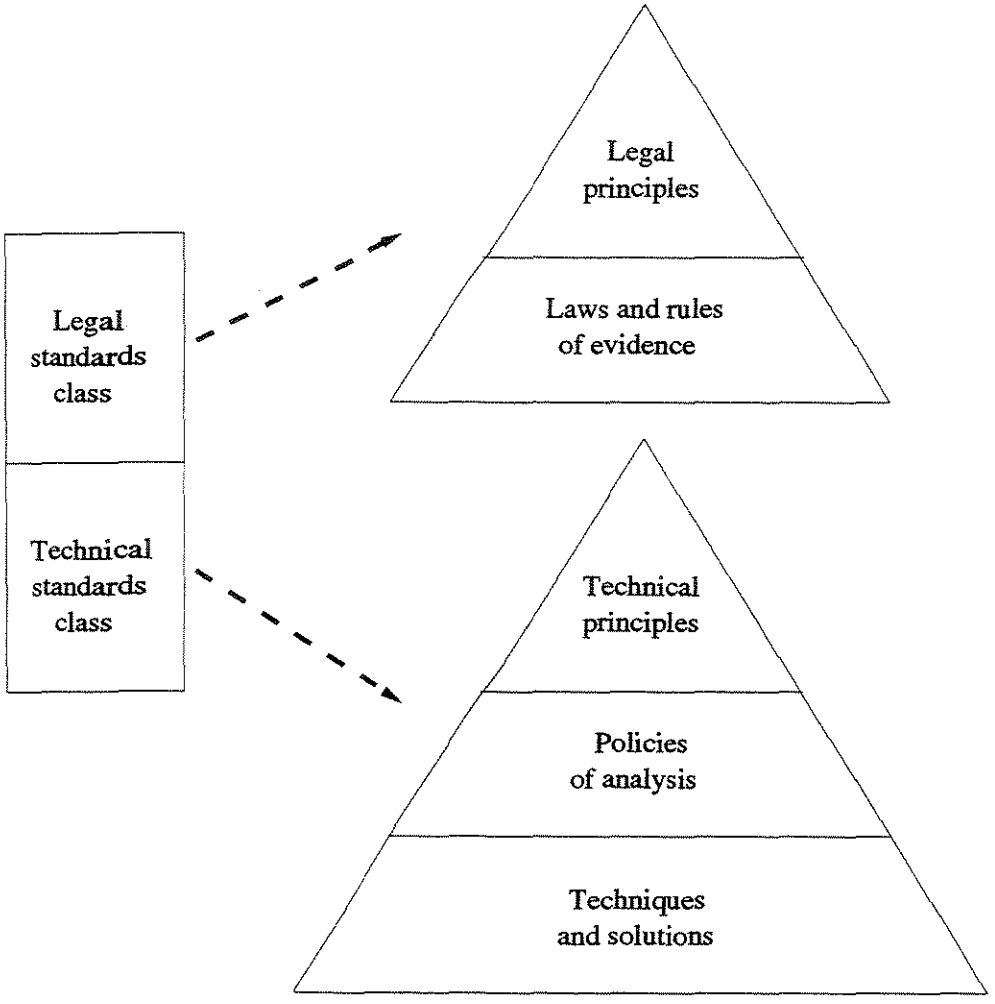


Figure C.1: Standardization model.

the inclusion of the legal standards class). This extension better represents the concerns of the forensic science and allows modelling the legal acceptance of examination results and procedures. The proposed model is a two-class hierarchical structure detailed as follows.

C.4.1 Legal Standards Class

Forensic is defined by [21] as meaning "appropriate for courts of law or for public discussion or argumentation". Forensic analysis is a scientific specialty whose purpose is to provide information suitable for presentation in the courts considering laws and rules of evidence for federal and state jurisdictions [5].

In this sense, legal standards are the laws and rules that dictate the admissibility of digital evidence in the courts of law. They consist of the formalities and judicial constraints that are applied to the experts, the forensic science and the examination.

Since digital crimes can transcend country boundaries, it is not reasonable to expect all nations to have the same laws and rules of evidence. In order to allow the exchange of digital evidence, such legal constraints should be coherent with principles consistent with all legal systems. Based on Brazilian legislation and references [15,16,17], the authors propose some general features that might represent such principles:

- forensic analysis, as well as the development and evaluation of examination procedures and protocols, should be conducted by a graduate person, that is forensically competent and has technical and scientific skills related to the nature of the analysis. Besides that, this person should not have records that could raise suspicions against his/her moral strength and ethics;
- whenever a new forensic examination procedure is conceived, it should be evaluated and testified before use;
- minimum requirements for the evidence (such as reliability, integrity, accuracy and durability) should be predisposed and accurately defined;
- technical standards should guarantee the minimum requirements for the evidence;
- forensic procedures should be coherent with established standards;
- forensic procedures and standards should be generally accepted by the relevant scientific community and should be reviewed on a regular basis;
- forensic tools should be licensed or authorized for use in the examination. Such tools should be generally accepted in the field or supported by tests conducted in a scientific manner;
- the access rights over private information should be established (for instance, the expert could be granted the right to access all kinds of private information during forensic examination). Examination procedures should respect the established rights;
- forensic examination should produce a report according to some standardized model;
- the expert should be legally responsible for the examination results and evidence while it is in his/her possession;

- the situations in which one expert is legally prohibited to conduct a particular examination should be predisposed (for instance, when the perpetrator has some level of kinship with the examiner);
- forensic examination should be conducted by at least two experts, in order to reduce personal influence on the results;
- good working conditions should be guaranteed to the examiner, especially regarding the availability of equipament and material needed for the examination, resulting in an accurate and efficient work;
- the expert should weight his/her assertions and conclusions with scientific foundations, in a way that there is, from the scientific point of view, only one possibility to get to such assertions;
- forensic examination should be an indispensable part of investigation whenever there are evidences to be analyzed;
- forensic examination should preserve evidence in order to allow a second analysis;
- the original state of evidence should be protected until the arrival of the experts. Any alteration should be reported;
- the examination results should be as much elucidating as possible; to this effect, the use of photographs, drawings and diagrams should be allowed;
- warrant requirements for searching and seizing computers should be predisposed;
- situations in which searching and seizing computers without a warrant is allowed should be established.

C.4.2 Technical Standards Class

Technical standards are related to the practical requirements and conducts needed to carry out the examination. They are hierarchically arranged as follows.

C.4.2.1 Technical Principles

Technical principles are basic concepts that always apply to all sorts of examination. They are consensus directives that should guarantee minimum requirements to the evidence, such as reliability, integrity, accuracy and durability. Among the following principles, some are quoted [9,4] and some are proposed by the authors:

- actions taken along the examination should not change the evidence;
- a chain of custody documentation should be maintained for all digital evidence. It should report the name of the examiner that is handling the evidence, date, time and activities performed;
- copies of the original digital evidence should be made and whenever possible the examination should be applied over the copies;
- the copies of the original digital evidence should be authenticated by means of cryptographic signatures [19];
- the examiner should not trust the analyzed system, because it might be corrupted;
- all assumptions that come up during examination should be reported for later scientific foundation;
- all media utilized during the examination process should be freshly prepared, completely wiped of non-essential data, scanned for viruses and forensically verified before use;
- all information (activities relating to the seizure, storage, examination or transfer of digital evidence, case notes and examiner's observations, and data extracted and analyzed) should be recorded in a permanent manner and should be available for review and testimony;
- precautions should be taken to prevent the transfer of viruses, destructive programs, or other inadvertent writes to/from the examined media and other media used for the examination;
- analysis techniques and tools should be known in each detail, in order to avoid unexpected implications and side effects over the analyzed system;
- care and attention should be taken in regard to metric units and mathematical approximations, in order to avoid missing any information in the storage devices;
- all resources that forensic science has available should be attempted to access information that is deleted, locked, hidden, password protected or encrypted. Resources used should be reported;
- evidence should be searched in every possible and relevant source of information, according to their volatility order;

- a complete examination may not be authorized, possible or necessary. In such instances, the examiner should report the reason for not conducting a complete examination.

C.4.2.2 Policies of Analysis

A Policy of analysis is a practical guidance that applies to forensic examinations, defining how they are planned, performed, monitored, recorded and reported to ensure that technical principles are observed. There is, in the policy, the specification of everything that is necessary for the accomplishment of the analysis and step-by-step guidelines applied to most examinations (guidelines represent only the framework of the examination process). A sketch of a guideline is proposed as follows, based on [9,10,11,12,13].

- *Step 0*: Define the best approach for the examination, identifing all activities that will be required;
- *Step 1*: Prepare the examination system, providing the most suitable, properly tested and forensically sterilized hardware and software setup for the examination (in some cases it may be necessary to reproduce a specific setup, such as a network arrangement);
- *Step 2*: Stabilize the initial condition of the computer system subject of the examination, in order to preserve as much evidence as possible and protect systems and data out of the examination scope. The computer system should be described in its initial condition (hardware and software setups, executing processes, network connections, date and time accuracy, for example). Some issues should be evaluated at this point, such as the system shutting off (when and how it should be done), the maintenance of the system online or offline, the need to capture network traffic and detailed information of executing processes. Issues like these should be adressed in the policy of analysis;
- *Step 3*: Copy the information stored in the computer system. The copy should contain all the information in its original state and should be authenticated;
- *Step 4*: Extract as much relevant information as possible, regarding their order of volatility;
- *Step 5*: Examine each information separately, then in a correlative manner;
- *Step 6*: Correlate each evidence found (computer evidence or not). The examiner should establish a timeline, order of events, related activities and contradictory evidence;

- *Step 7*: Elaborate the final report based on the objective results of the examination.

C.4.2.3 Techniques and Solutions

Techniques and solutions are hardware and software solutions to specific activities performed during examination. They are detailed instructions that describe the use of techniques and tools necessary for the examination process. An example of solution is presented as follows.

Solution for imaging a hard disk with IDE interface, using an *Intel Pentium III* hardware running *Red Hat Linux 7.1* operating system:

- *Step 1*: Describe the hard disk, reporting all relevant information that may be printed on its external surface, as well as the jumpers original setup;
- *Step 2*: Install the disk on the analysis system on an open second IDE port and boot the system. To avoid damaging the disk by possible master/slave conflicts on the IDE controller, install it as the only drive connected to the IDE cable on the second IDE interface. Consider in the next steps that the disk to be imaged is accessible by the block device */dev/hdc*;
- *Step 3*: Have the BIOS detect the disk on the analysis system. Verify and make notes of the disk geometry detected;
- *Step 4*: Identify the partitions on the drive using the command *fdisk -l /dev/hdc*. Make notes;
- *Step 5*: Generate the MD5 checksum of the information stored on the disk using the command *dd if=/dev/hdc | md5sum -b*;
- *Step 6*: Copy each and every bit from the disk to the analysis sytem using the command *dd if=/dev/hdc of=filename*, where *filename* is the name of the file that will hold the disk image;
- *Step 7*: Generate the MD5 checksum of the file holding the disk image using the command *dd if=filename | md5sum -b*;
- *Step 8*: Compare the MD5 checksums generated on *Steps 5* and *7*. They should be exactly the same.

The previous example is meant to be an illustrative explanation and so does not take into account a series of adverse situations, such as the size of the file generated, the presence of bad sectors on the disk and the unmatching of the MD5 checksums.

C.5 Further Debate

Computer forensic science is an evolving forensic discipline that claims for standards and discussion. To this effect, some issues surrounding the standardization of computer forensic procedures are recommended for further debate.

C.5.1 Scientific Community Acceptance

The development and discussion of computer forensic science foundations should be carried in compliance with the relevant scientific community. It should not be carried in a commercial manner, in other words, all information (such as events, tools and techniques) should be shared and tested in a scientific manner.

C.5.2 General Purpose Discipline

The goal of the forensic analysis of computer systems is basically *persuasion based on factual evidence* [5]. As a consequence, forensic techniques might have a wider use than law enforcement purposes. The benefit would be a much higher confidence level in the information presented to decision makers in the business, industrial, government or military domains [5].

C.5.3 *Proven Correct* Tools

In the courts, admission and presentation of scientific evidence is guided by the established judicial rule and legal precedence. So, even though computer forensic scientists can do, for instance, a bit image of digital evidence and authenticate it with hashing algorithms, it will be the accuracy and reliability of the copy tool and hash employed that may be called into question [5]. It is importante to use *proven correct* tools to conduct a computer forensic examination. However, this assertion relies on a lacking definition of *proven correct.*

Proving that a tool is correct is definitely not an easy task that can fall into several levels of recursion. Consider the following:

One could use software engineering to build the algorithm and data flow of the tool. Then the correctness of the algorithm should be proved, as well as the accuracy of the implementation. The next step would be proving that the implementation compiles to the right machine instructions, and so one should prove that the compiler is *proven correct.* Besides that, one should prove that the operating system is *proven correct* and so the hardware, because the execution of the tool depends directly on them.

This approach would be impracticable (what to say, then, if only the binary code is available). The most practical and wise approach would be to test and evaluate the tools [8], in a way that if the scientific community agreed that the tools are accurate and reliable, they could be used as forensic tools.

C.5.4 Legal and Technical Standards Relationship

The relationship between legal aspects and technical practices can be better understood through the following example. Consider that the legal access right over private information granted to the examiner does not allow him/her to examine some documents, such as communications between the suspect and their spouse, attorney or priest. Now consider an examination procedure that is strictly compliant with all technical principles and policies, but whose actions are based on the inspection of all kinds of documents stored in the computer system. This examination procedure would not be considered legally defensible.

C.5.5 Standardization Level

The standardization of computer forensic procedures and protocols may not be applicable to all levels of the model proposed in Section C.4. This is as a consequence of the great variety of exams that may be requested and diversity of emerging and rapidly changing technologies. In this sense, standardization might be restricted to the levels of principles (legal and technical) and policies of analysis, allowing the use of non-standardized techniques and solutions, still coherent with the standardized levels. Another approach is the standardization only on the level of principles.

The hierarchical feature of the proposed model allows addressing the problem of diversity of examinations and evolving technologies. In the top of the hierarchy there are general and constant aspects. As the hierarchy is traversed, aspects become more specific and less constant, working as a set of primitive techniques. A new procedure might be developed by derivation of these primitive techniques. In this way, this new procedure, after being evaluated and testified according to the general and constant aspects, might then be included in the set of primitive techniques.

C.6 Conclusions

Criminal activity has also converted from a physical dimension, in which evidence and investigations are described in tangible terms, to a cyber dimension, in which evidence exists only electronically and investigations are conducted online [3]. Forensic science, as

a discipline integrated by the different fields of the scientific knowledge, needs to remain up-to-date with regard to the scientific progress.

The role played by forensic science in a criminal investigation is the extraction and presentation of evidence in a suitable way for the courts of law. To this effect, procedures and protocols are needed to support the results of a forensic examination. As an evolving forensic discipline, computer forensic science claims for standards and scientific research.

In comparison with traditional forensic disciplines, computer forensic science is almost entirely technology and market-driven and generally located outside the laboratory setting. Also, the examinations present unique variations in almost every situation [3]. As a consequence, computer forensic procedures and protocols should be written in a hierarchical manner so that good principles remain constant. However, examination techniques must be allowed to quickly adapt to the computer system to be examined and to recognize the fast-changing and diverse world of computer science [3].

Moreover, computer forensic protocols and procedures should be coherent with legal and technical standards, in order to allow the exchange of digital evidence among different jurisdictions in a way that evidence is guaranteed to be reliable and defensible. To this effect, a standardization model (see Section C.4) is proposed by the authors to evaluate and develop computer forensic procedures.

The work presented in this paper represents an effort to fulfill the lack of scientific research on computer forensic science. However, there is yet a lot of work to be done, with some issues still open for further research, such as:

- the difficulty in documenting every possible action in the form of operational procedures;
- the access rights over private information during the examination process. Examiners have no ability to identify possible privileged information until after they read it. Also, probative information might be hidden inside privileged information;
- the lack of tools for analysis and correlation of evidence;
- the amount of information to be analyzed increases with the storage capacity of the devices. A complete and efficient examination becomes a challenge;
- the difficulty in accessing encrypted or password locked information;
- the development of security policies that maximize the usefulness of incident evidence data, and minimize the cost of forensic examination [14];

C.7 References

[1] M. A. Reis, F. Oliveira, P. L. Geus, and C. Guimarães. Forense Computacional: Aspectos Legais e Padronização. (Computer Forensics: Legal Aspects and Standardization). In *Proceedings of WSeg'2001: Workshop on Computer Security*, Florianópolis, SC, Brazil, March 2001, pp 80-85. (in Portuguese).

[2] M. A. Reis, and P. L. Geus. Forense Computacional: Procedimentos e Padrões. (Computer Forensics: Procedures and Standards). In *Proceedings of SSI'2001: 3rd Symposium on Informatics Security*, São José dos Campos, SP, Brazil, October 2001, pp 73-81. (in Portuguese, awarded with honorable mention).

[3] M. Noblett, M. Pollitt, and L. Presley. Recovering and Examining Computer Forensic Evidence. In *Forensic Science Communications*, Volume 2, Number 4, October 2000. U.S. Department of Justice, FBI.

[4] Scientific Working Group on Digital Evidence and International Organization on Digital Evidence. Digital Evidence: Standards and Principles. In *Forensic Science Communications*, Volume 2, Number 2, April 2000. U.S. Department of Justice, FBI.

[5] G. L. Palmer. CyberForensic Analysis.

[6] International Organization on Computer Evidence (IOCE) Web Site. In URL <http://www.ioce.org>. (last visited on November 2001).

[7] Scientific Working Group on Digital Evidence (SWGDE) Wb Site. In URL <http://www.for-swg.org/swgdein.htm>. (last visited on November 2001).

[8] Computer Forensics Tool Testing (CFTT) Project Web Site. In URL <http://www.cftt.nist.gov>, September 2001. (last visited on November 2001).

[9] C. Hosmer, J. Feldman, and J. Giordano. Advancing Crime Scene Computer Forensic Techniques. In URL <http://wetstonetech.com/crime.htm>, 2000. WetStone Technologies Inc. (last visited on November 2001).

[10] D. Farmer, and W. Venema. Computer Forensics Analysis Class Handouts. In URL <http://www.fish.com/forensics/ class.html>, 1999. (last visited on August 2001).

[11] R. Firth, G. Ford, B. Fraser, et al. Detecting Signs of Intrusion. In URL <http://www.cert.org/security-improvement/modules/m09.html>, 1997. Security Improvement Module, CERT Coordination Center. (last visited on August 2001).

[12] Evidence Examinations - Computer Examinations. In *Handbook of Forensic Services*, 1999. U.S. Department of Justice, FBI.

[13] D. Dittrich. Basic Steps in Forensic Analysis of Unix Systems. In URL <http://www.ic.unicamp.br/ra000504/ftp/forensics/papers/dittrich/basicsteps.html>. (last visited on November 2001).

[14] J. Tan. Forensic Readiness. In *The CanSecWest Computer Security Conference*, April 2001.

[15] O. S. Kerr. Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations. In URL http://www.cybercrime.gov/searchmanual.htm, 2001. Computer Crime and Intellectual Property Section (CCIPS), Criminal Division, U. S. Department of Justice. (last visited on August 2001).

[16] D. Tochetto, H. Galante, J. Zarzuela, et al. *Tratado de Perícias Criminalísticas*. (Treatise of Criminalistic Examinations). Sragra-DC Luzzatto, 1995. First edition. (in Portuguese).

[17] A. Espindula. A Função Pericial do Estado. (The Forensic Duty of the State). In URL <http://www.espindula.com.br/artigo1.htm>. (in Portuguese, last visited on August 2001).

[18] A. Espindula. Técnicas Criminalísticas para Conclusão de Laudo Pericial. (Criminalistic Techniques for the Examination Report Conclusion). In URL <http://www.ic.unicamp.br/ra000504/ftp/forensics/papers/espindula/laudo.doc>. (in Portuguese, last visited on August 2001).

[19] S. Garfinkel, and G. Spafford. (1996). *Practical Unix and Internet Security*. O'Reilly and Associates, 1996. Second edition.

[20] CNN - O. J. Simpson Trial. In URL <http://www.cnn.com/US/OJ/index.html>, 1995. Cable News Network, Inc. (last visited on August 2001).

[21] The American Heritage Dictionary of the English Language. Houghton Mifflin Company. Fourth edition.

Apêndice D

Aspectos de utilização e configuração do AFA

Como apresentado no Capítulo 6, o AFA é composto de um servidor (o *script* `afa_server`), executado no sistema de análise, e de um *script* cliente (o `gather_target`), executado na máquina invadida para a coleta de informações voláteis e imagens dos discos (quando a máquina não pode ser desligada). A utilização e configuração dos componentes do AFA são descritas como segue.

D.1 Manual de utilização do `afa_server`

```
NAME
      afa_server - server side of AFA (Automated Forensic Analyser)

SYNOPSIS
      afa_server  [ -lsvMV ] [ -c case_name ] [ -d data_dir ] [ -e execution_dir
      ] [ -i local_device or image_dir ] [ -p port ] [ -t target_device ]

DESCRIPTION
      afa_server is the main part of AFA (Automated Forensic  Analyser).  It  is
      executed  at  the  forensic  station and is responsible for receiving data
      from the target machine (volatile information and disk images), organizing
      the data received (authenticating and timestamping), performing local disk
      imaging (if target machine can be turned off and disassembled),  searching
      for evidences on gathered data, analysing evidences  found  and  producing
      output based on analysis.

      afa_server implements all steps of a computer forensic examination: infor
      mation gathering, evidence searching and analysis. The information gather
      ing process is controled by the  configuration  file  gather.rules.  This
```

process intends to collect volatile information from the target machine and depends on the execution of the client part of AFA, gather_target (8), on the target machine. All data is transfered over the network. Evidence searching is performed at the forensic station, through the execution of a series of plugins (as stated in the configuration file plugins.cf). Evi dence analysis is not yet implemented in this version.

Every time afa_server is executed, it produces an execution directory, which is saved in the directory $DATA (the value of which is found in the afa.cf file or set with -d option). The execution directory is named under the case name (the value of which is found in the afa.cf file, under variable $CASE, or can be set with -c option) concatenated with afa_server start execution time. If option -e is used, no execution directory is created, in fact afa_server is executed to analyse a previ- ously created execution directory.

The AFA project provides an architecture to automate the computer forensic examination process. Although the last step of this process is not yet implemented, the work done until now is a great effort in the direction of that automation. The AFA architecture is extensible and adaptable, in the way that is can be easily configurated and extended with new analysis plu gins. Moreover, its project is very simple and uses a great variety of external programs, which are commonly used on forensic examinations, such as netcat (1), ps (1), netstat (1), and so on (it is important to use reliable and statically linked programs, whose path is set in the configu ration file paths.cf).

OPTIONS

-c case_name
: Sets the name of the investigation case. It overrides variable $CASE in afa.cf

-d data_dir
: Sets the data directory where execution directories are located. It overrides variable $DATA in afa.cf

-e execution_dir
: Use existing execution directory from previous run of afa_server. The execution directory is located on the data directory.

-i local_device or image_dir
: Tells afa_server to produce an image of the disk located on local_device or to use a previously created image located on image_dir (image directory must have a valid image_description file).

```
       -l      Log execution on log file indicated by variable $logfile in
               afa.cf

       -M      Do not mount images on forensic station.

       -p port Sets the port number where afa_server should listen for imaging
               informations. It overrides variable $PORT in afa.cf

       -s      Make separate images for each partition.

       -t target_device
               Device used by disk image on target machine. If imaging is per
               formed on the forensic station, but informations like fdisk output
               are gathered from target machine, the device may be different.

       -v      Be verbose.

       -V      Prints version.

ENVIRONMENT
       AFA_HOME, location of AFA software and configuration files.

FILES
       afa.cf: The main configuration file (is Perl executable code).
       paths.cf: Paths to the tools used by AFA software (is Perl executable
       code).
       gather.rules: Configuration file for information gathering.
       plugins.cf: Configuration file for evidence searching.
       image_description: Contains the description of the image files on image
       directory.

HISTORY
       afa_server is under prototype phase and has been freely available since
       2002.

AUTHOR
       Marcelo Abdalla dos Reis <marcelo.reis@ic.unicamp.br>

SEE ALSO
       gather_target(8) ps_analyser(8) packet_analyser(8)
```

D.2 Manual de utilização do gather_target

```
NAME
       gather_target - client side of AFA (Automated Forensic Analyser)
```

SYNOPSIS

gather_target [-IvV] [-i devide_list] [-s device_list]

DESCRIPTION

gather_target is executed at the target machine, in order to collect volatile information. It runs many sub-programs in an attempt to capture forensic information about a Unix system. Which sub-programs and the way they must be executed are indicated in the configuration file gather.rules.

All data is transfered through the network to the afa_server , at the forensic station. Besides volatile information, gather_target can transfer images of the disks on the target machine (in the case it can not be turned off).

As gather_target is executed at the target machine, it must be contained in a removable media, along with all sub-programs run by it (it is impor tant to use reliable and statically linked programs, whose path is set in the configuration file paths.cf).

OPTIONS

-i device_list
Images the disks on device list. If this option is not provided, the default is to produce images of all disks on target machine.

-I Do not image disks.

-s device_list
Make separate partition images for each disk on device list.

-v Be verbose.

-V Prints version.

ENVIRONMENT

AFA_HOME, location of AFA software and configuration files.

FILES

afa.cf: The main configuration file (is Perl executable code).
paths.cf: Paths to the tools used by AFA software (is Perl executable code).
gather.rules: Configuration file for information gathering.

HISTORY

gather_target is under prototype phase and has been freely available since

```
       2002.

AUTHOR
       Marcelo Abdalla dos Reis <marcelo.reis@ic.unicamp.br>

SEE ALSO
       afa_server(8)
```

D.3 Arquivos de configuração do AFA

O AFA possui quatro arquivos de configuração principais, descritos como segue:

- `afa.cf`: é o arquivo de configuração principal, onde se encontram variáveis importantes para o funcionamento dos componentes do AFA;
- `paths.cf`: contém o caminho para os sub-programas utilizados pelos componentes do AFA;
- `gather.rules`: configura a etapa de coleta de informações, indicando os comandos que devem ser executados, onde as informações coletadas devem ser armazenadas e as portas TCP utilizadas para a transferência das informações;
- `plugins.cf`: configura a etapa de busca de evidências, indicando os *plugins* que devem ser executados;

Os arquivos de configuração utilizados nos testes preliminares do AFA são apresentados na sequência. Todos os arquivos possuem comentários (indicados pelo caracter "#" no início da linha) explicando a sintaxe utilizada e a finalidade de cada declaração. O caracter "\" contido no final de uma linha é utilizado, neste texto, para indicar a sua continuidade na linha seguinte. Cada declaração nos arquivos de configuração do AFA deve estar contida em uma única linha.

D.3.1 Arquivo `afa.cf`

```
#
#  Configuration file for AFA software
#
#  Each statement is a Perl executable code.
#  It contains important variables used by AFA components
#

#
```

```
#  Initial directories
#

# path to the AFA home directory
$AFA_HOME = "/root/tools/projeto/afa";

# path to the aux scripts used by the main ones
$LIB    = "$AFA_HOME/lib"    unless $LIB;

# path to the analysis plugins
$PLUGINS    = "$AFA_HOME/plugins"    unless $PLUGINS;

# path to the plugins configuration directory
$PLUGINSCFG    = "$PLUGINS/conf"    unless $PLUGINSCFG;

# path to the main AFA components
$BIN    = "$AFA_HOME/bin"    unless $BIN;

# path to the AFA configuration directory
$CONFIG = "$AFA_HOME/conf"    unless $CONFIG;

# path to data directory where the execution directories are located
$DATA    = "$AFA_HOME/data"    unless $DATA;

# set INC variable
@INC = ("$AFA_HOME/lib", "$AFA_HOME/conf", @INC);

#
#  Special variables
#

require "paths.cf";

# log file name (program name plus extension)
$logfile = "$AFA_HOME/log/$0".".log"  unless $logfile;

# date format (day_mon_year or year_mon_day)
$dateformat = "day_mon_year"  unless $dateformat;

# forensic station adress
$FORSTATION = "localhost"  unless $FORSTATION;

# afa_server listening port for controling the imaging process
$PORT = "2222"  unless $PORT;

# starting port to receive images from target machine
```

```
$IMGPORT = "2223"  unless $IMGPORT;

# convertion options used by dd command
$DDCONV = "noerror,sync" unless $DDCONV;

# command plus options for message digest
$DIGESTCMD = "$MD5SUM -b" unless $DIGESTCMD;

# device name of the imaged disk on the target machine
$targetdevice = "/dev/hda"  unless $targetdevice;

# number of Linux partition type
$linux_partition = 83 unless $linux_partition;

# name of the files with collected output from fdisk, df and mount
$fdisk_out = "fdisk.out"  unless $fdisk_out;
$df_out = "df.out"  unless $df_out;
$mount_out = "mount.out"  unless $mount_out;

# if set, search fdisk_out using local device name, else use targetdevicename
#$uselocalfdisksearch = '0'  unless $uselocalfdisksearch;

# name of the case under investigation
#$CASE = "" unless $CASE;
```

D.3.2 Arquivo paths.cf

```
#
#  Path to external sub-programs used by AFA software
#
#  Each statement is a Perl executable code.
#  Use the upper case name of the program to name the variables.
#
#  If it is used by gather_target, all sub-programs should be located
#  in a removable media and the paths should be related to the media
#  mount point.
#

$CAT      = "/bin/cat";
$CP       = "/bin/cp";
$DATE     = "/bin/date";
$DD       = "/bin/dd";
$DF       = "/bin/df";
$ECHO     = "/bin/echo";
$FDISK    = "/sbin/fdisk";
$FIND     = "/usr/bin/find";
```

```
$GREP     = "/bin/grep";
$HOSTNAME = "/bin/hostname";
$LS       = "/bin/ls";
$MD5SUM   = "/usr/bin/md5sum";
$MOUNT    = "/bin/mount";
$MV       = "/bin/mv";
$NETSTAT  = "/bin/netstat";
$NC       = "/usr/bin/nc";
$PS       = "/bin/ps";
$RM       = "/bin/rm";
$TEE      = "/usr/bin/tee";
$TEEPL    = "$LIB/tee.pl";
$TOUCH    = "/bin/touch";
$UMOUNT   = "/bin/umount";
```

D.3.3 Arquivo gather.rules

```
#
#  Configuration file for information gathering at the target machine.
#
#  Sintax: <command & arguments>  <output destination filename on forensic station>
#          <destination port on forensic station>
#
#  Notes:
#        * Must use one line for each command;
#        * Must use only the command name (paths are set in paths.cf);
#        * Must use only the name of the output file (it is always located at
#            $INFOGATHERED directory)
#        * Destination port must be increased by two (one is for the information
#            itself and other is for information checksum)
#

fdisk -l -u     fdisk.out       3333

df              df.out          3335

mount           mount.out       3337

ps -ewo pid,ppid,user,lstart,time,etime,%cpu,%mem,rss,sz,longtname,stat,intpri, \
ni,flags,command          ps.out          3339
```

D.3.4 Arquivo plugins.cf

```
#
#  Configuration file for analysis of gathered informations.
```

```
#
#  Sintax: <plugin & arguments>
#
#  Notes:
#        * Must use one line for each plugin;
#        * Must use only the plugin name (plugins must be located at $PLUGINS
#            directory);
#        * The output filename (same used in gather.rules) where the plugin should
#            search for evidences must be preceded by the keyword "$INFOGATHERED/"
#

ps_analyser   $INFOGATHERED/ps.out

packet_analyser -f $INFOGATHERED/tcpdump.out
```

D.4 *Plugins* implementados

Ao longo do desenvolvimento do AFA foram implementados apenas dois *plugins*: o `ps_analyser` e o `packet_analyser`. O funcionamento completo do AFA requer, como condição ideal, a existência de pelo menos um *plugin* para cada fonte de informação a ser analisada.

O desenvolvimento dos *plugins* foi baseado nos conceitos apresentados no Capítulo 5, em especial nas características das fontes de informações e nas evidências mais comumente encontradas em cada uma. Por exemplo, o *plugin* `ps_analyser` analisa as informações relativas aos processos em execução, obtidas através do comando ps, em busca das evidências frequentemente encontradas nessa fonte de informação. De maneira análoga, o *plugin* `packet_analyser` busca por características suspeitas no conteúdo dos pacotes capturados da rede (seu objetivo principal é a identificação de pacotes relacionados com ataques de *buffer overflow*, podendo ser facilmente utilizado no gerador de assinaturas do sistema ADenoIdS, apresentado no Capítulo 4).

É importante observar que, no corrente estágio de prototipação do AFA, a saída produzida pelos *plugins* não está integrada ao funcionamento proposto para o AFA. Apenas são geradas mensagens de texto, notificando a identificação de uma situação suspeita.

As características de utilização e configuração dos *plugins* implementados são apresentadas na sequência.

D.4.1 Manual de utilização do `ps_analyser`

```
NAME
       ps_analyser - AFA (Automated Forensic Analyser) plugin to analyse the out
       put of ps (1) command.
```

```
SYNOPSIS
       ps_analyser ps_output_file

DESCRIPTION
       ps_analyser is the plugin of AFA (Automated Forensic Analyser) that
       searches for evidences in the output of ps (1) command. It is executed by
       afa_server (8) at the forensic station. ps_analyser receives as argument
       the name of the file containing the output of the ps (1) command executed
       at the target machine. Based on its configuration files, ps_analyser
       searches for indications of suspicious processes. The analysis performed
       by ps_analyser is largelly based on threshold detection and keyword
       searching.

ENVIRONMENT
       AFA_HOME, location of AFA software and configuration files.

FILES
       afa.cf: The AFA main configuration file (is Perl executable code).
       paths.cf: Paths to the tools used by AFA software (is Perl executable
       code).
       ps_analyser.rules: The ps_analyser main configuration file.
       shouldberun: Names of the processes that should be running on target
       machine.
       inetdprocs: Names of the processes that should be running under super dae
       mon inetd.
       suspnames: Suspicious names to be matched against command names.

HISTORY
       ps_analyser is under prototype phase and has been freely available since
       2002.

AUTHOR
       Marcelo Abdalla dos Reis <marcelo.reis@ic.unicamp.br>

SEE ALSO
       afa_server(8) ps(1)
```

D.4.2 Arquivos de configuração do ps_analyser

O *plugin* ps_analyser possui quatro arquivos de configuração principais, descritos como segue:

- ps_analyser.rules: é o arquivo de configuração principal, onde se encontram as regras para a detecção de situações suspeitas;

- `shouldberun`: contém o nome dos processos que devem estar executando na máquina comprometida. A ausência de algum desses processos indica uma situação suspeita. Como exemplos desses processos podem ser citados o `syslogd` e o `tcpwrapper`.

- `inetdprocs`: contém o nome dos processos que devem ser executados através do `xinetd`. Muitas vezes os atacantes utilizam os nomes desses processos para "camuflar" seus programas maliciosos;

- `suspnames`: contém uma série de nomes suspeitos, comumente utilizados pelos atacantes para nomear suas ferramentas. Pode conter também o nome de programas que não deveriam estar executando;

Os arquivos de configuração utilizados nos testes preliminares do *plugin* `ps_analyser` são apresentados na sequência. Todos os arquivos possuem comentários (indicados pelo caracter "#" no início da linha) explicando a sintaxe utilizada e a finalidade de cada declaração.

D.4.2.1 Arquivo `ps_analyser.rules`

```
#
#  Configuration file for ps_analyser plugin
#

# The rules may be specified for a particular process
# or they may be global and affect all processes
#
# Important: process specific rules override global ones
#
# Process specific rules are indicated by keywords ''beginprules''
# and ''endprules''. The sintax is as follows:
#
# beginprules command
# rules
# rules
# endprules
#
# ,where command is the name to be matched with COMMAND field of /bin/ps output
# (it may be a regular expression of Perl matching operation).

#
# Global rules
#
```

```
# Resource use threshold (percentage)

minpcpu = 4
maxpcpu = 90
minpmem = 3
maxpmem = 85

# List of suspicious owners
# One can specify the suspicious owners or the complement of them (using !=)
# The list should be one line and separated by comma or spaces

suspowner = ftp,bin,r00t,

# Process owner
# If the process owner is not this, something is wrong
# owner overrides suspowner

owner = root

# Suspicious start time interval (hh:mm:ss - hh:mm:ss)
# One can specify the suspicious interval or the complement of them (using !=)

suspstime != 11:22:00-11:30:00

# Suspicious start date interval (dd:mm:yyyy - dd:mm:yyyy)
# One can specify the suspicious interval or the complement of them (using !=)

suspsdate != 24:03:2002 - 24:03:2002

# Use both suspstime and suspsdate if set
# If suspstime or suspsdate is missing, it considers only the rule that is present

suspsdatetime = 1

#
# Process specific rules
#

# Rules specific to bash
beginprules bash
minpcpu = 0
maxpcpu = 30
minpmem = 0.2
maxpmem = 2.4
owner = root
endprules
```

```
# Rules specific to xemacs
beginprules xemacs
maxpcpu = 0.6
maxpmem = 16.3
suspstime = 13:35:00-13:37:00
suspsdate = 27:02:2002 - 27:07:2005
suspsdatetime = 1
endprules
```

D.4.2.2 Arquivo shouldberun

```
#
#  Names of processes that should be running on target machine
#
#  Notes:
#        * Names are matched against COMMAND field of /bin/ps output
#        * Use one name per line
#  * Names may contain regular expressions of Perl matching operation

sshd
ipchains
syslogd
```

D.4.2.3 Arquivo inetdprocs

```
#
#  Names of the processes that should be running under super daemon inetd
#
#  Notes:
#        * The names are matched against COMMAND field of /bin/ps output
#        * Use one name per line
#  * Names may contain regular expressions of Perl matching operation

telnet
rsh
rlogin
finger
echo
```

D.4.2.4 Arquivo suspnames

```
#
#  Suspicious names to be matched against COMMAND field of /bin/ps output
#
#  Notes:
#        * Use one name per line
#  * Names may contain regular expressions of Perl matching operation

sniffer
snif
sniff
^snif*
rootkit
r00t
killer
*crack*
```

D.4.3 Manual de utilização do packet_analyser

```
NAME
       packet_analyser - open source packet processor and pattern matcher

SYNOPSIS
       packet_analyser [-CdevXy?] [-f rules_file ] [-r rule ] packets_file

DESCRIPTION
       Packet_analyser  is an open source packet processor, capable of performing
       packet decoding and content pattern searching/matching. It is intended  to
       detect buffer overflow attacks, searching for patterns that identify these
       attacks in the packets payload.  Packet_analyser reads the packets from  a
       tcpdump(1)/ snort(8) binary log file and uses a flexible rules language to
       describe the patterns to be matched (in fact it is based  on  snort(8)  ´s
       rules  language  to  describe content search).  Packet_analyser was origi
       nally  designed to be a plugin of AFA  (Automated  Forensic Analyser), but
       it  can  be used as a back-end for any packet processing tool (with proper
       modification of function ProcessPacket in source file packet_analyser.c ).
       Packet_analyser ´s design is largelly based on snort(8) ´s one (that's the
       magic of open source).

OPTIONS
       -C     Print the character data from the packet payload only (no hex).

       -d     Dump the application layer data when displaying packets in verbose.

       -e     Display the link layer packet headers.

       -f rules_file
```

```
        Set the filename of the pattern rules file to rules_file.

    -r rule
        Match the packets against the pattern rule rule.

    -v  Be verbose. Prints packets out to the console.

    -X  Dump the raw packet data starting at the link layer. This switch
        overrides the -d switch.

    -y  Include the year in timestamp display.

    -?  Show the program usage statement and exit.

    packets_file
        Filename of the tcpdump(1)/ snort(8) binary log file.

RULES
    Packet_analyser uses a simple but flexible rules language to describe pat
    terns to be matched.

    rules_file
        Each rule in rules_file must be in a separate line. However, a sin
        gle rule may span multiple lines, ending each line with caracter
        '\'.

        Comments are allowed, begining with caracter '#'.

    rule
        Each rule is an expression that consists of operand ´s associated
        with binary operator ´s and nested with brackets. For example,

        ( operand operator operand ) operator operand

        A rule is evaluated from right to left, conserning precedence of
        nested expressions.

        The elements of a rule are:

        operator
            An operator is a logical operator AND or OR (parsing is
            case-insensitive).

        operand
            An operand is a snort(8) like content rule option. Each
            operand consists of one or more ruleopts enclosed into
```

```
brackets (just like snort(8) 's rule options). Each ruleopts
is terminated with ';'. Possible ruleopts are:

content

        Format:
             content: [!] "content_string";

        The content keyword specifies a pattern to be
        matched (content_string). Be aware that content test
        is case sensitive. content_string can contain mixed
        text and binary data. The binary data is enclosed
        within the pipe ('|') character and represented as
        bytecode (hexadecimal numbers). The characters '"',
        ':', '|' must be escaped inside a content_string.
        Note that multiple content keywords can be specified
        in one single operand. In this case the result of
        the operand is the logical AND of the multiple pat
        terns test. If the content declaration is followed
        by a '!', the pattern test is inverted.

        Example:
             content: "|90C8 C0FF FFFF|/bin/sh";

content-list

        Format:
             content-list: [!] "content_list_filename";

        The content-list keyword specifies the name of a
        file containing a list of content declarations. Each
        content declaration must be on a single line of con
        tent_list_filename and its format is [!] "con
        tent_string". In this case, the result of the
        operand is the logical OR of the multiple patterns
        test. If the content-list declaration is followed by
        a '!', the pattern test is inverted for all pat
        terns specified in content_list_filename and the
        result of the operand is the logical AND of the mul
        tiple inverted patterns test.

uricontent

        Format:
             uricontent: [!] "content_string";
```

The uricontent keyword allows search to be matched against only the URI portion of a request. For a description of the parameters to the uricontent rule option, see content rule option.

offset

Format:
offset: number;

The offset keyword is a modifier to rule options content, content-list or uricontent. This keyword modifies the starting search position for the pat tern match function from the beginning of the packet payload. It affects only the last content declara tion (content, content-list or uricontent). This rule option keyword cannot be used without also specifying a previous content declaration.

depth

Format:
depth: number;

The depth keyword is a modifier to rule options con tent, content-list or uricontent. This sets the max imum search depth for the content pattern match function to search from the beginning of its search region. It is useful for limiting the pattern match function from performing inefficient searches once the possible search region for a given set of con tent has been exceeded. It affects only the last content declaration (content, content-list or uri content). This rule option keyword cannot be used without also specifying a previous content declara tion.

nocase

Format:
nocase;

The nocase keyword is used to deactivate case sensi tivity in a content, content-list or uricontent dec laration. It is specified alone within a rule and any ASCII characters that are compared to the packet

payload are treated as though they are either upper of lower case. It affects only the last content dec laration (content, content-list or uricontent). This rule option keyword cannot be used without also specifying a previous content declaration.

regex

Format:
regex;

The regex keyword allows the use of wildcard in a content_string. The wildcards behave more like shell globbing than Perl-type regular expressions. A '*' in the content_string, along with the regex modifier is interpreted to mean "any character, any number of times". Additionally, the '?' character means "any single character".

An operand must have one or more content declarations (might it be of one kind or a mix of them), and all modifiers may be used together. Note that a rule option must be terminated with ';', otherwise it is dismissed.

HISTORY

Packet_analyser is under prototype phase and has been freely available since 2002.

DIAGNOSTICS

Packet_analyser returns a 0 on a successful exit, 1 if it exits on an error.

BUGS

Send bug reports to marcelo.reis@ic.unicamp.br

AUTHOR

Marcelo Abdalla dos Reis <marcelo.reis@ic.unicamp.br>

SEE ALSO

tcpdump(1), snort(8), pcap(3), afa_server(8)

Glossário

Agente infeccioso: organismo causador de infecções, incluindo, por exemplo, vírus, bactéria e fungo.

Agente móvel: programa que executa tarefas em nome de um usuário e tem autonomia para migrar entre servidores do sistema.

Agente patogênico: vide "agente infeccioso".

Análise de vulnerabilidades: é a análise do estado de segurança de um sistema através da checagem de suas configurações.

Análise forense: o termo "análise forense" compreende todo o processo de investigação de um determinado evento ocorrido em um sistema computacional, incluindo a coleta de informações, a busca de evidências e o processamento dos vestígios encontrados (além das atividades relacionadas como, por exemplo, a planejamento e a documentação).

Anticorpo: proteína produzida pelas células B, que se adere a antígenos específicos de um micróbio, impedindo sua proliferação e ampliando a capacidade de detecção das células do sistema imunológico.

Antígeno: cadeia de proteína contida na superfície dos agentes patogênicos, através da qual o sistema imunológico efetua a detecção dos invasores. Como o objetivo da detecção efetuada pelo sistema imunológico é a identificação dos antígenos *nonself* e não dos micróbios em si, o termo "antígeno" é utilizado, neste trabalho, para referenciar os agentes invasores.

Arquitetura de análise forense: neste trabalho, o termo "arquitetura de análise forense" é utilizado para referenciar a arquitetura extensível de um sistema automatizado de análise forense, desenvolvida ao longo desta pesquisa.

Arquivo de *log*: arquivo contendo registros de eventos.

Assinatura: uma especificação de aspectos, condições, arranjos e inter-relações entre eventos que caracterizam um ataque.

Atacante: um usuário legítimo ou não, que explora ou tenta explorar alguma vulnerabilidade do sistema, de maneira a prejudicar suas operações normais ou com intenção de obter acesso indevido a informações ou recursos, em desacordo com a política de segurança.

Ataque: um conjunto de ações que resultam em uma violação da política de segurança de um sistema computacional.

***Audit reduction*:** pré-processamento dos dados de auditoria, com o objetivo de identificar e remover informações reduntantes ou irrelevantes.

Auditoria de sistemas: processo de geração, armazenamento e revisão de registros cronológicos de eventos do sistema.

Auto-imunidade: disfunção do sistema imunológico, que passa a atacar o próprio corpo.

***Back door*:** algum mecanismo instalado no sistema alvo que permite sobrepujar o seu controle de acesso.

***Booby trap*:** armadilha implantada por um atacante para executar algum tipo de tarefa quando determinadas atividades forem realizadas no sistema invadido. Essas tarefas incluem, por exemplo, a limpeza dos vestígios deixados pelo atacante e a danificação do sistema.

***Buffer overflow*:** ataque que explora a falta de checagem quanto ao tamanho dos dados a serem armazenados em um determinado *buffer*. Nesse caso, os dados são cuidadosamente elaborados para excederem o tamanho da área de memória alocada ao *buffer*, sobrescrevendo o endereço de retorno da função que fez a alocação. Desse modo, é possível desviar a execução do programa vulnerável para qualquer ponto da memória, contendo algum código malicioso.

Casamento de padrões: busca, em um determinado conjunto de dados, de padrões definidos.

Célula B: linfócito do tipo B, responsável pela produção de anticorpos.

Célula T: linfócito do tipo T, responsável pelo controle da resposta adaptativa e pela eliminação das céculas infectadas.

Cifragem: é o processo pelo qual uma mensagem (chamada de texto claro) é transformada em outra (chamada de texto cifrado), através da aplicação de uma função matemática (chamada de algoritmo de cifragem) e de uma senha especial de cifragem (chamada de chave). Tecnologias de cifragem podem ser aplicadas para garantir a privacidade, autenticação, integridade e não-repúdio de informações.

***Core file*:** arquivo contendo uma imagem, em um formato especial, da memória utilizada por um determinado processo.

***Cracker*:** programa que tenta descobrir a senha relacionada a uma informação protegida.

Criminalística: disciplina autônoma, integrada pelos diferentes ramos do conhecimento técnico-científico, auxiliar e informativa das atividades policiais e judiciárias de investigação criminal, tendo por objeto o estudo dos vestígios materiais extrínsecos à pessoa física, no que tiver de útil à elucidação e à prova das infrações penais e, ainda, à identificação dos autores respectivos.

Disco espelho: disco que recebe as informações do disco original no processo de espelhamento.

Disco suspeito: neste trabalho, o termo "disco suspeito" é utilizado para referenciar o disco da máquina invadida.

Equipe de resposta a incidentes: conjunto de profissionais capacitados a executar as primeiras medidas no tratamento de um incidente de segurança.

Evidência: neste trabalho, os termos "evidência", "vestígio" e "indício" são utilizados como sinônimos, no sentido de qualquer informação que possa estar relacionada a uma intrusão. Sob o enfoque criminalístico, existe uma distinção entre os termos "vestígio" e "indício" (nesse caso, sinônimo de "evidência"). Nesse sentido, "vestígio" é todo material recolhido em um local de crime para análise posterior; e "indício" é o vestígio depois de feitas as análise, onde se constata técnica e cientificamente a sua relação com o crime.

Exame forense: vide "análise forense".

***Exploit*:** programa malicioso que explora uma vulnerabilidade particular de um sistema computacional ou *software*.

Fagócito: célula de defesa do sistema imunológico humano. Pertence ao sistema inato e é responsável pela resposta inicial à invasão de um micróbio.

Falso-positivo: é a caracterização equivocada, por parte de um IDS, de uma atividade não intrusiva como sendo relacionada a um ataque.

***Firewall*:** um *firewall* provê uma fronteira de segurança para uma determinada rede através de uma política de controle de acesso. Os mecanismos usados incluem servidores *proxy*, filtros de pacotes e túneis de dados cifrados.

Forense computacional: é a tradução da expressão inglesa *Computer Forensics*, uma área de pesquisa em Ciência da Computação que analisa evidências em sistemas de computação com vistas a recriar etapas e descobrir causas que levaram à consecução de um determinado evento em tal sistema. É uma área de pesquisa recente e é tradicionalmente associada à área de Segurança de Redes de Computadores.

***Hash* criptográfico:** uma função de *hash* toma uma entrada qualquer, de tamanho indefinido, e produz um número pequeno (chamado de *hash*), que é significativamente menor que a entrada. Essa função é determinística e pode gerar *hashes* iguais para entradas diferentes. A função de *hash* criptográfico possui, ainda, duas importantes propriedades: ela não pode ser prevista ou invertida, e uma mudança pequena na entrada resulta em alterações significativas no *hash* produzido.

HOWTO: documento contendo uma explanação resumida e prática sobre um determinado assunto.

Imagem bit-a-bit: uma cópia exata de um disco, contendo todos os seus bits sem qualquer alteração na ordem ou valor.

Imunoglobulina: um classe de proteínas.

Imunologia computacional: área de pesquisa relacionada ao desenvolvimento de modelos imunológicos computacionais para a resolução de problemas.

Indício: vide "evidência".

Inferência Bayesiana: técnica de inferência, baseada na interpretação Bayesiana de probabilidade, para o raciocínio sobre incertezas, imprecisões e imprevistos.

***Information gathering*:** etapa do processo de investigação forense que compreende as atividades relacionadas à coleta de informações para análise.

Inteligência artificial: área de pesquisa dedicada à modelagem do pensamento humano e de processos cognitivos para a resolução automatizada de problemas complexos.

Invasor: vide "atacante".

Investigação forense: vide "análise forense".

Investigador: aquele responsável pela investigação forense.

LAS-IC-UNICAMP: Laboratório de Administração e Segurança de Sistemas do Instituto de Computação da Universidade Estadual de Campinas.

Linfócito: célula de defesa do sistema imunológico humano. Pertence ao sistema adaptativo e é responsável pela detecção específica e resposta eficiente.

***Link* simbólico:** um arquivo que armazena o caminho de referência a um determinado arquivo.

***Man page*:** página de manual comumente utilizada em ambientes UNIX (acessada através do comando `man`).

MHC: *Major Histocompatibility Complex.* São moléculas protéicas, contendo fragmentos de antígenos, produzidas pelas células infectadas por um micróbio.

Micróbio: vide "agente infeccioso".

Modelo de IDS imunológico: neste trabalho, o termo "modelo de IDS imunológico" é utilizado para referenciar o modelo conceitual de IDS híbrido, baseado no sistema imunológico humano, desenvolvido ao longo desta pesquisa.

Modo promíscuo: um dispositivo de rede em "modo promíscuo" capta os pacotes que trafegam no segmento de rede ao qual está conectado, mesmo que estes não lhe sejam destinados.

***Modus operandi*:** termo em latim que significa "método de operação".

***Nonself*:** algo estranho a uma determinada entidade.

NOP: *No Operation.* Instruções que não executam qualquer operação no processador.

Padrão de intrusão: vide "assinatura".

***Patch*:** uma correção ou atualização aplicada a um sistema computacional ou *software*, para sanar uma falha, um erro de projeto ou uma situação não previsa.

Patógeno: vide "agente infeccioso".

Peptídeo: pequeno fragmento do antígeno de um agente patogênico.

Perícia criminal: é aquela que examina todo material sensível relativo às infrações penais, na busca da constatação se ocorreu o delito e da prova material de sua prática.

Persecução criminal: atividade exercida pelo Estado, personificando o interesse da sociedade na repressão às infrações penais, que envolve dois momentos distintos: a investigação do fato infringente da norma e quem tenha sido o seu autor, colhendo os necessários elementos probatórios a respeito; e a ação penal e seu acompanhamento, levando ao conhecimento do Juiz a notícia sobre o fato infringente da norma, a fim de que, apreciando-o, declare se procede ou improcede, se é fundada ou infundada a pretensão estatal.

PID: número de identificação de um processo.

***Plugin*:** dispositivo de *software* que integra um sistema maior e extensível.

Política de segurança: um conjunto de procedimentos e normas que regulam a maneira pela qual uma organização gerencia, protege e distribui informações sensíveis através de seus sistemas computacionais.

***Post-mortem*:** termo em latim que significa "depois da morte".

Receptor: cadeia de proteína contida na superfície das células do sistema imunológico, que reage com os antígenos dos agentes patogênicos permitindo sua detecção.

Rede neural: é uma abordagem de aprendizado baseada na modelagem computacional do processamento que ocorre nos neurônios do cérebro humano.

Registro de auditoria: registro que descreve um determinado evento do sistema.

RFC: *Request for Comments*. Um documento técnico que define ou formaliza um padrão dentro da rede Internet.

***Rootkit*:** Conjunto de ferramentas, utilizadas por usuários mal-intencionados, para conseguir privilégios de *root* em um sistema alvo e esconder suas atividades.

***Scanning*:** processo de rastreamento de informações e vulnerabilidades de um sistema alvo.

***Script*:** programa interpretado, comumente contendo comandos de um sistema operacional.

***Self*:** que é parte de uma determinada entidade.

Sistema de análise: máquina confiável e íntegra onde a análise forense é executada.

Sistemas de produção/especialistas: um sistema especialista encapsula o conhecimento adquirido de um especialista humano e o aplica automaticamente para tomar decisões.

Sistema de segurança: neste trabalho, o termo "sistema de segurança" é utilizado no sentido de um sistema de detecção de intrusão em sua plenitude, ou seja, que implementa, além da identificação de cenários intrusivos, um mecanismo de resposta.

Sistema de segurança imunológico: sistema de segurança baseado no modelo imunológico humano.

***Sniffer*:** programa de captura de tráfego de rede.

***Sniffing*:** processo de captura de tráfego de rede através de dispositivos em modo promíscuo.

***Trojan horse*:** Programa aparentemente inofensivo e correto, mas que contém código malicioso.

Usuário malicioso: vide "atacante".

Vestígio: vide "evidência".

Vulnerabilidade: uma falha nos procedimentos automáticos de segurança, controle administrativo ou controle de acesso, que pode ser utilizada por um atacante para obter acesso não-autorizado ao sistema, às informações armazenadas ou para interromper suas operações normais.

***World Wide Web*:** grande rede mundial de hipertextos.