Uma arquitetura de segurança computacional inspirada no sistema imunológico

Fabrício Sérgio de Paula

Tese de Doutorado

Instituto de Computação
Universidade Estadual de Campinas

Uma arquitetura de segurança computacional inspirada no sistema imunológico

Fabrício Sérgio de Paula

Junho de 2004

Banca Examinadora:

- Prof. Dr. Paulo Lício de Geus
  Instituto de Computação, UNICAMP (Orientador)
- Prof. Dr. Ricardo Dahab
  Instituto de Computação, UNICAMP
- Prof. Dr. Célio Cardoso Guimarães
  Instituto de Computação, UNICAMP
- Prof. Dr. Carlos Alberto Maziero
  Centro de Ciências Exatas e de Tecnologia, PUC-PR
- Prof. Dr. Leandro Nunes de Castro Silva
  Programa de Mestrado em Informática, UNISANTOS
- Prof. Dr. Ricardo Anido (Suplente)
  Instituto de Computação, UNICAMP
- Prof. Dr. Luciano Paschoal Gaspary (Suplente)
  Centro de Ciências Exatas e Tecnológicas, UNISINOS

Uma arquitetura de segurança computacional inspirada no sistema imunológico

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Fabrício Sérgio de Paula e aprovada pela Banca Examinadora.

Campinas, 13 de julho de 2004.

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

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

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Paula, Fabrício Sérgio de

P281a Uma arquitetura de segurança computacional inspirada no sistema imunológico / Fabrício Sérgio de Paula -- Campinas, [S.P. :s.n.], 2004.

Orientador : Paulo Lício de Geus

Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Computadores – Medidas de segurança. 2. Imunologia. 3. Redes de computação – Medidas de segurança. I. Geus, Paulo Lício de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Aos meus pais, por todo carinho e incentivo dispensado.
À Danieli, por seu amor e compreensão em toda esta jornada.
Ao Felipe, que veio trazer muita alegria, contribuindo com este trabalho.

Agradecimentos

Ao Criador, pois tudo fez, tudo formou, e me proporcionou fôlego de vida, dando a mim a oportunidade de realizar este trabalho. Somente a Ele seja dada toda a honra e toda a glória para sempre. Amém.

Também gostaria de prestar meus agradecimentos a todos que compartilharam um bom convívio e me auxiliaram de alguma forma durante o desenvolvimento deste trabalho.

Aos meus familiares, minha gratidão, porque me incentivaram e auxiliaram durante toda a trajetória da minha vida. Que fique registrada a mesma gratidão para os familiares da minha esposa e para aqueles que, devido à sua proximidade, proporcionaram uma ligação bastante familiar.

Aos professores e colegas do DCT/UFMS, que me forneceram uma boa instrução na graduação, a qual possibilitou meu ingresso efetivo na carreira de pesquisa.

Ao meu orientador, Prof. Dr. Paulo Lício de Geus, pela oportunidade concedida, companheirismo e orientação. À sua família, meus sinceros agradecimentos pelos momentos de alegria proporcionados.

À banca examinadora do meu exame de qualificação, Prof. Dr. Ricardo Dahab e Prof. Dr. Adriano Mauro Cansian, pelas valiosas sugestões apresentadas. Nesse ponto, também gostaria de agradecer ao Prof. Dr. Leandro Nunes de Castro por seu interesse e contribuição em relação a esta pesquisa.

Aos revisores da versão preliminar, Diego, Nilton e Adriana, porque me auxiliaram bastante na elaboração desta redação.

Aos colegas do Laboratório de Administração e Segurança de Sistemas, pelas discussões proporcionadas e experiências compartilhadas. Gostaria de agradecer à Cleymone, Felipe e Edmar, pelo empenho dispensado na agilização de tarefas cotidianas. Ao Diego e Marcelo, por todos os momentos convividos e pela amizade conquistada.

À FAPESP e ao CNPq, que possibilitaram o desenvolvimento desta pesquisa através da concessão de apoio financeiro.

Aos professores, funcionários e colegas do Instituto de Computação, que também contribuiram de algum modo para a realização deste trabalho.

Resumo

O sistema imunológico humano é capaz de garantir a sobrevivência de um indivíduo durante toda sua vida, mesmo que ele se depare, a cada dia, com bactérias e vírus potencialmente mortais. Dessa forma, esse sistema biológico provê uma rica fonte de inspiração para a manutenção da segurança computacional. Além de representar um modelo bastante próximo das condições em que a maioria das redes de computadores se encontra, o sistema imunológico engloba uma série de características e princípios desejáveis a um sistema de segurança.

Considerando esses fatos, este trabalho explora as características e princípios do sistema imunológico humano para construir uma arquitetura de segurança computacional. A arquitetura desenvolvida possibilita a identificação de ataques através da análise de evidências de intrusão, respostas inespecífica e específica e a extração automatizada de assinaturas para ataques desconhecidos, tornando o sistema computacional dinamicamente adaptável a novos ataques.

Com base nessa arquitetura foi construído um protótipo, ADENOIDS, que implementa as principais funcionalidades modeladas e considera a importante classe de ataques *buffer overflow*. Resultados experimentais permitem verificar que é possível identificar ataques desconhecidos através da análise de evidências de intrusão. Após identificar um ataque, são ativados os mecanismos de restauração e é iniciado o processo de extração de assinatura. Utilizando um algoritmo probabilístico para a extração de assinaturas, também desenvolvido nesta pesquisa, é possível eliminar falso-positivos e obter uma assinatura para o ataque em questão.

Abstract

The human immune system is able to guarantee the survival of an individual for his/her entire life, even though he/she encounters potentially deadly parasites, bacteria and viruses on a daily basis. In this way, this biological system provides a rich source of inspiration for the security of computer networks. Besides the fact that the human immune system presents a closely-related model of the real network conditions in the present day, it has many features that are desirable for a security system.

Given these facts, this work explores the features and principles of the human immune system for building a network security architecture. The architecture developed here enables the identification of attacks through intrusion evidence analysis, provides specific and unspecific responses and is able to extract signature for new attacks, making the computer system dinamically adaptable against new attacks.

A prototype, ADENOIDS, was implemented based on this architecture, covering the main modeled features and considering buffer overflow attacks. Experimental results show that it is possible to identify new attacks by the intrusion evidence analysis mechanism. After identification, ADENOIDS activates system restoration mechanisms and initiates the signature extraction process. By using an algorithm developed in this research it is possible to discard false-positives and to identify the attack signatures.

Sumário

Lista de tabelas

Lista de figuras

Lista de algoritmos

Capítulo 1

Introdução

A Internet foi inicialmente desenvolvida para propiciar um ambiente aberto e distribuído que estava calcado na confiança mútua existente entre seus usuários. Os benefícios propiciados por esse ambiente são inúmeros e ultrapassam fronteiras geográficas, culturais e sociais. Entretanto, a diversidade verificada atualmente na Internet contempla também indivíduos dispostos a promover perturbações nesse ambiente. Tais indivíduos são movidos por razões ideológicas, psicológicas, financeiras, desejo de vingança e até mesmo por pura diversão.

Por outro lado, assuntos relacionados à segurança computacional raramente têm recebido uma alta prioridade por parte de desenvolvedores de *software*, revendedores, administradores ou consumidores. Como resultado dessa distração um número considerável de vulnerabilidades surge constantemente. Uma vez exploradas por atacantes, essas vulnerabilidades colocam governos, negócios e usuários individuais sob risco [SG96, Pet00].

Considerando essa situação, diversas pesquisas têm sido desenvolvidas na tentativa de aumentar a segurança de um computador ou rede de computadores. Entretanto, o problema da segurança ainda não foi resolvido efetivamente e um amplo campo para estudos continua aberto. O presente trabalho vem atuar justamente nessa área, buscando melhorar as condições de segurança em um ambiente computacional. Porém, antes de apresentar mais detalhes desta pesquisa, a Seção 1.1 faz uma introdução sobre segurança computacional, descrevendo brevemente um importante meio que visa alcançar tal segurança.

1.1 Segurança computacional

Em uma definição prática de segurança, um sistema é considerado seguro quando se comporta de maneira esperada por seus usuários [SG96]. Uma definição mais precisa de segurança computacional é dada em termos da tríade: confidencialidade, integridade

e disponibilidade. A confidencialidade busca garantir que todo acesso à informação seja restrito somente àqueles usuários autorizados para tal acesso. A integridade demanda que a informação seja protegida contra alterações indevidas. A disponibilidade requer que a informação e os recursos do sistema estejam acessíveis a todo momento em que forem solicitados [Bac00].

Diversas tecnologias têm sido empregadas visando garantir esses quesitos de segurança, incluindo *firewalls*, criptografia, análise de vulnerabilidades e sistemas de detecção de intrusão. O uso de sistemas de detecção de intrusão é especialmente interessante porque permite identificar a ocorrência de ataques e habilitar uma resposta aos ataques identificados. Devido a essas características, tais sistemas tornaram-se uma importante peça na provisão de segurança computacional [Bac00, BM01].

Várias técnicas têm sido empregadas no desenvolvimento de sistemas de detecção de intrusão mas, em linhas gerais, elas estão encaixadas em uma dentre duas classes de análise:

1. Baseada em conhecimento: permite identificar precisamente ataques através de assinaturas pré-especificas, mas não permite identificar ataques desconhecidos;

2. Baseada em comportamento: identifica ataques desconhecidos mas é menos precisa que a análise baseada em conhecimento, acabando por gerar muitos alarmes falsos ou falso-positivos.

Se, por um lado, a ocorrência de ataques desconhecidos pode passar despercebida, por outro, a geração de falso-positivos pode levar à desabilitação do sistema de detecção. Tendo como base esses dois problemas principais, pesquisadores de variadas formações nas áreas de segurança e inteligência computacional vêm sendo mobilizados. Embora as estratégias de detecção concebidas procurem estabelecer um ambiente livre de quaisquer ataques, ainda nenhuma solução definitiva foi divisada.

Um melhor nível de segurança pode ser atingido adotando melhores modelos, os quais venham representar de maneira mais próxima as condições em que a maioria das redes de computadores se encontra — um ambiente hostil e sujeito a falhas. 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 imunológico é 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. Além disso, existe uma forte analogia entre o sistema imunológico e um sistema de segurança computacional.

A analogia entre problemas de segurança e processos biológicos foi inicialmente reconhecida no início de 1987, quando o termo "vírus de computador" foi introduzido por

Adelman [Coh87]. Já a conexão entre imunologia e segurança de computadores teve início em 1994 com as publicações [FAPC94, Kep94], desencadeando uma série de outros trabalhos.

Entretanto, a maioria dos esforços têm sido concentrada no desenvolvimento de técnicas de análise baseada em comportamento, o que explora apenas uma parte do modelo oferecido pelo sistema imunológico e incorre em falso-positivos. Utilizando esse modelo, é possível idealizar um sistema de segurança que possa manipular ataques conhecidos, identificar ataques desconhecidos através de evidências de intrusão, prover restauração após uma invasão e estudar ataques desconhecidos de uma forma automatizada.

1.2 Hipótese

Este trabalho consiste em utilizar o sistema imunológico biológico como uma fonte inspiradora na construção de sistemas de segurança computacional. A principal hipótese desta pesquisa pode ser definida como segue.

Considerando que a ocorrência de ataques com sucesso em um ambiente computacional qualquer é inevitável, é possível identificar ataques desconhecidos através da análise de evidências de intrusão e estudar automaticamente esses ataques de forma a torná-los conhecidos.

Dessa forma, esta pesquisa considera que um ataque desconhecido poderá permanecer indetectável até o momento em que o atacante inicia a exploração do alvo. Após esse momento serão geradas evidências de intrusão suficientes que permitam a identificação da intrusão. Uma vez tendo ciência da intrusão o sistema de segurança poderá efetuar uma resposta cabível e iniciar um estudo da intrusão em busca de assinaturas para o ataque.

É importante notar que o mesmo ocorre com o sistema imunológico. Alguns agentes infecciosos invadem o corpo e causam danos nele antes que o sistema imunológico consiga eliminá-los. Porém, a ocorrência desses danos gera uma coestimulação que permite uma melhor resposta ao ataque. Então o sistema imunológico aprende a lidar com esse novo tipo de agente, possibilitando respostas futuras bastante vigorosas.

Para efetuar uma validação da hipótese desta tese foi traçado um conjunto de objetivos, os quais são descritos na Seção 1.3.

1.3 Objetivos

Esta pesquisa possui três objetivos primários:

- Explorar as características e princípios do sistema imunológico humano para construir uma arquitetura de segurança computacional. Essa arquitetura deve possibilitar a identificação precisa de ataques conhecidos e a detecção de ataques desconhecidos através de evidências de intrusão, além de mecanismos de resposta para ambos. Ela deve possibilitar, ainda, a extração de assinatura para ataques desconhecidos, tornando-os, assim, conhecidos;

- Implementar um protótipo de sistema de segurança, ADENOIDS[1], baseado na arquitetura proposta para lidar com ataques *buffer overflow*;

- Validar as principais funcionalidades e idéias apresentadas na arquitetura através da realização de testes com o protótipo.

A justificativa para ADENOIDS lidar com ataques *buffer overflow* vem do fato dessa classe de ataques ser considerada um importante e persistente problema de segurança [NCFF01, CWP+00, LE01]. Exemplos de ataques vão desde o Internet Worm [Spa88], em novembro de 1988, até o recente *worm* W32/Sasser [UC04], em abril de 2004, atingindo importantes serviços de rede publicamente disponíveis. Segundo um relatório anual de vulnerabilidades notáveis divulgado pelo CERT Coordination Center [Cen04b], em um total de 28 vulnerabilidades observadas em 2003, 16 dessas estão relacionadas a *buffer overflow*. Por sua importância essa classe de ataques foi escolhida na implementação de ADENOIDS.

1.4 Organização do texto

A redação deste trabalho está organizada em duas partes maiores. A Parte I compreende a definição de conceitos e a realização da revisão bibliográfica. Essa parte engloba dois capítulos:

- Capítulo 2: apresenta os conceitos relativos à detecção de intrusão e as principais abordagens e técnicas já desenvolvidas. É definida, como uma contribuição deste trabalho, a detecção baseada em evidências de intrusão e são revisados os principais aspectos da detecção de ataques *buffer overflow*;

- Capítulo 3: apresenta o sistema imunológico biológico como uma fonte inspiradora na construção de sistemas de segurança computacional. São discutidas as principais

[1] No sistema imunológico humano, adenóides (*adenoids*, em inglês) são nódulos linfáticos especializados que contêm células imunológicas que protegem o corpo contra invasores do sistema respiratório. O termo "ADENOIDS" é utilizado neste trabalho para referenciar "**A**cquired **D**efense System Based on the **I**mmune System" e também remete à sigla IDS.

funcionalidades e são mostradas as pesquisas mais relevantes que empregam esse sistema biológico na segurança de redes ou computadores.

Após conhecidas as limitações existentes nas pesquisas envolvendo sistemas de detecção de intrusão e sistemas imunológicos voltados para a segurança, a Parte II faz a descrição da arquitetura proposta, do protótipo implementado, dos testes realizados e dos resultados obtidos. Os seguintes capítulos são englobados nessa parte:

- Capítulo 4: descreve a arquitetura desenvolvida, mostrando os componentes especificados e o funcionamento esperado de cada um. Também é apresentado um algoritmo genérico para a extração de assinaturas de ataque, são analisadas as principais questões de implementação e o relacionamento com pesquisas já realizadas;
- Capítulo 5: descreve os aspectos de implementação do protótipo ADENOIDS, mostrando como cada módulo implementado está relacionado com os componentes especificados na arquitetura. Também é ilustrado um exemplo de funcionamento geral do protótipo sob a ocorrência de um ataque;
- Capítulo 6: apresenta as características dos testes realizados e os resultados experimentais obtidos. Esses resultados são analisados de modo a compará-los com as idéias descritas na arquitetura proposta;
- Capítulo 7: realiza uma conclusão deste trabalho, ressaltando suas contribuições e fazendo uma projeção de pesquisas futuras.

Procurou-se desenvolver um texto bastante auto-contido, fazendo uso de exemplos e ilustrações para facilitar a leitura. Termos em inglês são apresentados em *itálico*, excetuando-se os nomes próprios. Exemplos envolvendo nome de ferramentas, trechos de código, comandos e resultados de execução são ressaltados em fonte `verbatim`.

Parte I

Conceitos e revisão bibliográfica

Capítulo 2

Detecção de intrusão

Assim como outras tecnologias recentes, a detecção de intrusão tem significados distintos em diferentes contextos. Ela tem sido usada freqüentemente para referenciar, por exemplo, a auditoria de dados, a filtragem de *firewalls*, a detecção de fraudes no sistema telefônico, o monitoramento em sistemas operacionais, o uso de câmeras para vigilância, e até mesmo a vigilância física visando o cumprimento de leis [Amo99].

No contexto deste trabalho, a detecção de intrusão é definida como o processo de monitoramento e análise de eventos em um sistema computacional em busca de sinais que indiquem a ocorrência de problemas de segurança [Bac00]. Um sistema de detecção de intrusão (IDS[1]) automatiza esse processo de monitoramento e análise de eventos de um computador ou rede de computadores.

Os sistemas de detecção de intrusão possuem as seguintes metas principais:

- Identificação: corresponde à capacidade de verificar a ocorrência de atividades maliciosas [Amo99]. A identificação de um ataque corresponde à detecção propriamente dita;

- Atribuição de responsabilidade: é a capacidade de relacionar uma dada atividade ou evento ao indivíduo responsável por sua ocorrência [BM01]. A atribuição de responsabilidade torna-se essencial quando deseja-se mover uma ação judicial contra um atacante. Essa meta pode ser difícil de ser alcançada uma vez que atacantes podem forjar a identidade da origem do ataque [SG96];

- Resposta: é a capacidade de executar ações para notificar usuários a respeito do ataque e restringir o acesso a recursos de modo a bloquear tal ataque [Bac00]. A atividade mais comum que pode ser caracterizada como uma resposta a um ataque é o registro dos resultados da detecção em um arquivo. Essa resposta, apesar de

[1] A sigla IDS é referente ao termo "*intrusion detection system*".

não atuar ativamente contra o ataque, possibilita que, posteriormente, uma análise detalhada seja conduzida para a descoberta de atividades ilícitas.

A detecção de intrusão começou a se desenvolver a partir de 1980. Desde então, várias pesquisas vêm sendo desenvolvidas, produzindo uma grande variedade de estratégias de solução nessa área [Bac00]. A Seção 2.1 descreve brevemente a origem da detecção de intrusão e a Seção 2.2 apresenta uma terminologia essencial que será utilizada a partir deste capítulo. Na Seção 2.3 são apresentados os principais conceitos e classificações de sistemas de detecção de intrusão e na Seção 2.4 são descritas algumas técnicas aplicadas na análise de informações. Essas seções são baseadas no livro Intrusion Detection de Rebecca Bace [Bac00], com algumas adaptações extraídas de outras referências devidamente citadas. Ao final, a Seção 2.5 descreve o funcionamento de ataques *buffer overflow* e as técnicas empregadas na identificação desses ataques. Essa classe de ataques foi considerada durante as etapas de prototipação e testes deste trabalho.

2.1 Origem da detecção de intrusão

A detecção de intrusão teve origem no sistema de auditoria financeira, onde auditores especializados inspecionavam os registros das transações de uma empresa, procurando por fraudes e erros. A partir de 1950, o trabalho desses auditores especializados passou a ser realizado por computadores. Com o desenvolvimento dos computadores e com sua propagação na década de 70, a demanda por segurança tornou-se aparente. Essa demanda mobilizou especialistas no sistema de auditoria financeira para o desenvolvimento de técnicas que permitissem a detecção de atividades ilícitas, dando início às pesquisas na área de detecção de intrusão.

James P. Anderson foi a primeira pessoa a documentar a necessidade de automatização de um sistema de auditoria voltado para a segurança. Ele escreveu, em 1972, um relatório visando um planejamento de estudo de segurança de computador dentro da Força Aérea dos Estados Unidos [And72]. Em 1980, Anderson propôs uma taxonomia para classificar riscos e ameaças, diferenciando as fontes internas e externas de problemas em um ambiente computacional [And80]. Essa classificação possibilitou a detecção automatizada de abusos em sistemas computacionais.

Entre 1984 e 1986, Dorothy Denning e Peter Neumann trabalharam no desenvolvimento de um modelo de sistema de detecção de intrusão em tempo real [Den86] e em um protótipo chamado Intrusion Detection Expert System (IDES) [L+90]. O modelo desenvolvido assume que as atividades intrusivas diferem das atividades normais. Dessa forma, a tarefa da detecção consiste em encontrar as atividades que não se enquadram nas representações de comportamento normal adotadas. Foram implementadas três versões

do sistema IDES que, posteriormente, foram refinadas e deram origem ao Next Intrusion Detection Expert System (NIDES). Mais recentemente, o sistema NIDES serviu como base para o desenvolvimento do sistema de detecção de intrusão Emerald [Pro01].

Os relatórios publicados por Anderson e o trabalho de Denning e Neumann são considerados os embriões da detecção de intrusão [Bac00, Pro01] e ainda servem como base para muitos trabalhos atuais.

2.2 Terminologia

Para apresentar e discutir os aspectos de segurança computacional relacionados à detecção de intrusão, faz-se necessária a utilização de alguns termos comuns nessas áreas. Os termos mais relevantes para este trabalho e seus significados são descritos como segue:

- Política de segurança: corresponde a uma especificação que define o que é considerado valioso e um conjunto de passos que devem ser seguidos para salvaguardar os bens ou valores [SG96];
- Vulnerabilidade: é uma fraqueza no sistema computacional que pode ser explorada de modo a violar a política de segurança em questão [Bac00];
- Ataque: compreende uma atividade maliciosa que procura violar a política de segurança [T+01]. Um ataque também pode ser descrito como uma tentativa de explorar vulnerabilidades em um sistema;
- Intrusão: é uma violação intencional da política de segurança de um sistema [Bac00]. Uma intrusão também pode ser entendida como um ataque bem sucedido;
- Monitoramento: consiste na ação de coletar informações e encaminhá-las para análise [Bac00];
- Correlação: refere-se à interpretação, combinação e análise de informações sobre um alvo de ataque para propósitos de detecção e resposta [Amo99];
- Alarme: corresponde a um relato da ocorrência de uma violação da política de segurança [Amo99];
- Falso-positivo: compreende a situação onde um alarme é disparado quando não existe uma violação da política de segurança [Pro01];
- Falso-negativo: compreende a situação onde um alarme deixa de ser disparado quando existe uma violação da política de segurança [Pro01];

- Verdadeiro-positivo: compreende a situação onde um alarme é disparado devido a uma violação da política de segurança [Pro01];
- Verdadeiro-negativo: compreende a situação onde um alarme deixa de ser disparado quando não existe uma violação da política de segurança [Pro01].

Vale ressaltar que esses termos não representam uma terminologia exaustiva utilizada na área de segurança computacional ou detecção de intrusão. Entretanto, as informações aqui descritas facilitam o entendimento de novos conceitos e discussões sobre a detecção de intrusão, encontrados a partir deste capítulo.

2.3 Sistemas de detecção de intrusão

Em termos gerais, um sistema de detecção de intrusão é composto por três componentes funcionais:

- Fonte de Informação: provê o fluxo de registros de informação que será utilizado pelo analisador. As informações requeridas por um IDS em geral são encontradas nos níveis de rede, sistema operacional e aplicação;
- Analisador: verifica os registros de informação disponíveis em busca por sinais que indiquem a ocorrência de tentativas de intrusões. Para tanto, o Analisador deve ser capaz de selecionar e organizar os registros relevantes derivados da Fonte de Informação e, de algum modo, compará-los com as especificações da política de segurança em questão;
- Componente de Resposta: executa um conjunto de reações à intrusão, baseado nos resultados obtidos pelo Analisador. Uma reação usual de sistemas de detecção de intrusão consiste simplesmente em informar a ocorrência de um ataque. Outras reações podem atuar de maneira a bloquear recursos, restaurar o sistema afetado ou, até mesmo, efetuar um contra-ataque.

A Figura 2.1 ilustra os componentes de um IDS. Nessa figura também é visualizada a entidade Especificações da Política de Segurança, usada para representar as especificações da política de segurança em questão, que é consultada pelo Analisador.

Existem várias abordagens que podem ser utilizadas no desenvolvimento dos componentes de um IDS. Essas abordagens determinam as características de um IDS específico e influenciam a atuação desse IDS. Ao longo desta seção serão apresentadas as principais abordagens utilizadas no projeto desses componentes, classificadas quanto à arquitetura, estratégia de monitoramento, tipo de análise, momento da análise, objetivo da detecção e estratégia de controle.

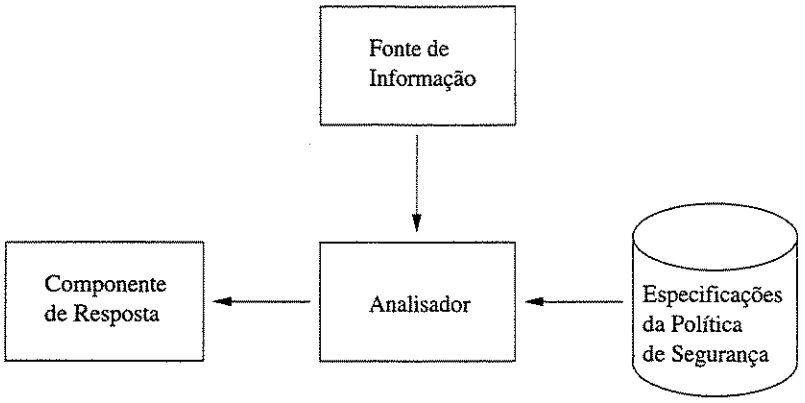

Figura 2.1: Componentes de um sistema de detecção de intrusão. As linhas indicam o sentido do fluxo da informação entre os componentes relacionados.

2.3.1 Arquitetura

Quando os sistemas de auditoria começaram a ser utilizados para a proteção de sistemas computacionais, o armazenamento da informação e o processamento eram realizados em um ambiente separado daquele que se desejava proteger. Nessa arquitetura, o sistema que executa a detecção de intrusão é chamado hospedeiro, enquanto o sistema sob monitoramento é chamado alvo.

Com essa separação, busca-se evitar que um intruso desabilite ou modifique os resultados de um IDS. Além disso, o desempenho do sistema alvo não é prejudicado, uma vez que esse sistema não efetua a detecção.

2.3.2 Estratégia de monitoramento

O primeiro requerimento de um IDS é uma fonte de informações a serem analisadas. De acordo com a localização do componente Fonte de Informação, o monitoramento de um IDS pode ser classificado basicamente em três categorias: baseado em rede, baseado em máquina e baseado em aplicação.

2.3.2.1 Monitoramento baseado em rede

O monitoramento baseado em rede opera coletando pacotes que trafegam na rede. A captura de pacotes usualmente é feita através de dispositivos de rede que são colocados em modo promíscuo [Bac00]. Para efetuar a coleta de informações é necessário que os dispositivos de captura estejam posicionados estrategicamente na rede, de modo que tenham acesso físico à rede que contém as máquinas a serem protegidas.

Através da análise dos protocolos envolvidos em uma comunicação, é possível extrair algumas informações relevantes para a detecção tais como: origem e destino da comunicação, serviço de rede envolvido, comandos enviados a um servidor e respostas obtidas, intensidade do tráfego, duração de uma conexão, etc.

As vantagens de se utilizar um monitoramento baseado em rede são [BM01, Pro01]:

- Um IDS baseado em rede bem posicionado é capaz de monitorar uma grande rede;
- Causa pouco impacto na rede monitorada. Em geral, IDSs baseados em rede são compostos por dispositivos de captura passivos que obtém informações na rede monitorada sem interferir na comunicação;
- Possibilita uma resposta bastante eficiente contra ataques externos conhecidos;
- Pode proporcionar um IDS bastante seguro contra ataques, podendo ser até mesmo invisível para alguns atacantes.

As principais desvantagens dessa estratégia são [BM01]:

- IDSs baseados em rede podem deixar de processar alguns pacotes quando o tráfego é muito intenso e, portanto, podem falhar no reconhecimento de ataques que são lançados durante esse período;
- Equipamentos como *switches* dificultam a captura do tráfego, pois dividem a rede em pequenos segmentos. Além disso, essa estratégia de monitoramento também é dependente da topologia de rede que se deseja monitorar;
- IDSs baseados em rede podem deixar de analisar tráfego cifrado. Essa característica pode ser bastante grave quando uma organização ou atacante usa uma rede privada virtual;
- Em geral, IDSs baseados em rede não conseguem verificar se um ataque ocasionou uma intrusão. Esses IDSs apenas identificam a ocorrência de um ataque. Desse modo, pode ser necessário que um administrador do sistema verifique manualmente a ocorrência de uma intrusão.

2.3.2.2 Monitoramento baseado em máquina

No monitoramento baseado em máquina a informação é coletada a partir de fontes internas a uma máquina, usualmente no nível do sistema operacional. A informação coletada pode incluir registros de auditoria gerados por um mecanismo especializado do sistema operacional e arquivos de *log* do sistema.

Registros de auditoria podem fornecer informações muito importantes para a detecção, como o acesso a arquivos confidenciais, a alteração de arquivos de configuração e binários do sistema, o estabelecimento de conexões de rede não autorizadas, a inserção de um módulo de *kernel* desconhecido, dentre outras. Em geral, registros de auditoria podem ser coletados através da análise de um subconjunto das chamadas ao sistema operacional.

Os *logs* do sistema tipicamente fornecem informações como: comandos executados por usuários, recursos usados e disponíveis no sistema, *logins* efetuados recentemente, *logins* negados recentemente, uso de comandos de superusuário, início e término de serviços de rede, inserção de módulos de *kernel*, etc.

As vantagens de se utilizar um monitoramento baseado em máquina são [BM01, Pro01]:

- Devido à habilidade de monitorar eventos locais a uma máquina, essa estratégia permite detectar ataques que não podem ser notados por um IDS baseado em rede;
- IDSs baseados em máquina podem operar normalmente em um ambiente no qual o tráfego de rede é cifrado, uma vez que a informação pode ser capturada antes da cifragem ou depois da decifragem;
- IDSs baseados em rede independem da topologia e tecnologia de rede empregada;
- Possibilita determinar o nível de comprometimento ocasionado por uma invasão;
- Possibilita verificar a integridade e veracidade da informação manuseada, o que torna essa abordagem bastante aceita para levantar evidências de ataques e para argumentação perante um tribunal;
- Quando um IDS opera sobre registros de auditoria do sistema operacional, ele pode detectar cavalos-de-tróia (*trojan horses* [NdG03], em inglês) ou outros ataques que comprometem a integridade de aplicações. Esses ataques são descobertos através da verificação de inconsistências no processo durante a execução.

As principais desvantagens dessa estratégia são [BM01, Amo99]:

- IDSs baseados em máquina possuem uma manutenção mais difícil, uma vez que a configuração e o gerenciamento necessários devem ser realizados em cada máquina;
- Como essa abordagem considera uma única máquina, não é possível detectar ataques que abrangem uma rede completa como, por exemplo, uma varredura da rede (*scan* [NdG03], em inglês);

- Como pelo menos a Fonte de Informação reside em máquinas que podem ser invadidas, o IDS pode ser desabilitado como parte do ataque;

- Registros de auditoria do sistema operacional podem gerar uma imensa quantidade de informação, impondo custos adicionais de armazenamento e processamento.

2.3.2.3 Monitoramento baseado em aplicação

O monitoramento baseado em aplicação é análogo ao monitoramento baseado em máquina. Nesse caso, entretanto, a informação coletada é proveniente de aplicações específicas. Essa informação compreende *logs* de eventos específicos e dados internos à aplicação.

As vantagens de se utilizar um monitoramento baseado em aplicação são [BM01]:

- IDSs baseados em aplicação permitem monitorar a interação entre um usuário e a aplicação, o que possibilita verificar atividade não autorizada para usuários individualmente;

- Essa estratégia pode atuar em ambientes cifrados, uma vez que é possível interagir diretamente com os dados utilizados pela aplicação.

A principal desvantagem do monitoramento baseado em aplicação reside no fato dos *logs* de aplicação ficarem bastante vulneráveis em um ataque, com um nível de proteção bastante inferior ao proporcionado para os registros de auditoria do sistema operacional.

2.3.2.4 Escolha da estratégia de monitoramento

A estratégia de monitoramento de um IDS deve ser escolhida de maneira a atender o propósito de detecção desse IDS. Desse modo, se um IDS propõe encontrar porções do tráfego de rede contendo requisições anômalas, deverá ser considerado um monitoramento baseado em rede. Se o propósito do IDS é verificar o efeito que ataques causam num sistema, deverá ser considerado um monitoramento baseado em máquina ou aplicação.

É possível, também, projetar um IDS que seja capaz de verificar o efeito que requisições anômalas no nível de rede causam no sistema. Nesse caso, o IDS deverá efetuar tanto o monitoramento baseado em rede quanto o monitoramento baseado em máquina ou aplicação. Além disso, esse IDS deverá implementar alguma estratégia de correlação para encontrar uma relação do tipo causa-efeito.

2.3.3 Método de análise

Após a escolha da estratégia de monitoramento adequada para um sistema de detecção de intrusão, o próximo passo consiste em definir o funcionamento do componente Analisador. Existem dois métodos básicos para analisar informações visando a detecção de ataques: baseado em conhecimento e baseado em comportamento[2] [DDW99]. Serão usados no texto os termos "detecção baseada em conhecimento" e "detecção baseada em comportamento" para referenciar o método de análise baseado em conhecimento e o método de análise baseado em comportamento, respectivamente. Neste trabalho também é definida a detecção baseada em evidências de intrusão, descrita ao final desta seção.

2.3.3.1 Detecção baseada em conhecimento

A detecção baseada em conhecimento consiste em usar o conhecimento acumulado sobre ataques para identificar situações onde esses ataques ocorrem [DDW99]. Esse conhecimento acumulado geralmente consiste em um banco de dados de assinaturas de ataque. Cada assinatura de ataque é composta por uma seqüência de eventos ou estados do sistema monitorado que corresponde a uma tentativa de violar a política de segurança, isto é, um ataque [NCFF01].

As principais vantagens da detecção baseada em conhecimento são [BM01, DDW99]:

- Possibilita uma identificação bastante efetiva de ataques, com a ocorrência de uma aceitável taxa de falso-positivos;
- Pode identificar rápida e precisamente a utilização de técnicas e ferramentas de ataque específicas. Essa característica pode auxiliar o administrador a priorizar medidas corretivas.

As desvantagens de utilizar a detecção baseada em conhecimento são [BM01, DDW99, Axe00]:

- Esse método possibilita identificar apenas os ataques pré-especificados no banco de dados de assinaturas. Sendo assim, esse banco de dados deve ser constantemente atualizado com novas assinaturas para prover a detecção de novas estratégias de ataque;
- Muitas assinaturas de ataque são bastante específicas e, portanto, o IDS torna-se incapaz de detectar variantes do mesmo ataque.

[2]Esses métodos também são conhecidos como detecção por mau uso e detecção por anomalia, respectivamente. Por conveniência esses termos serão evitados neste trabalho.

A Figura 2.2 ilustra o funcionamento da detecção baseada em conhecimento. Nessa figura as atividades realizadas no sistema computacional são classificadas como aceitáveis ou inaceitáveis, de acordo com a política de segurança em questão. Cada região hachurada compreende as atividades que são identificadas por uma assinatura de ataque. Cada assinatura possui uma abrangência local e, em geral, é necessário um grande número de assinaturas para identificar as atividades inaceitáveis. Como as assinaturas são bastante específicas para cada ataque, ocorrem poucos falso-positivos. Entretanto, um número maior de falso-negativos poderá ocorrer devido o surgimento de novos ataques ainda não representados no banco de dados de assinaturas.

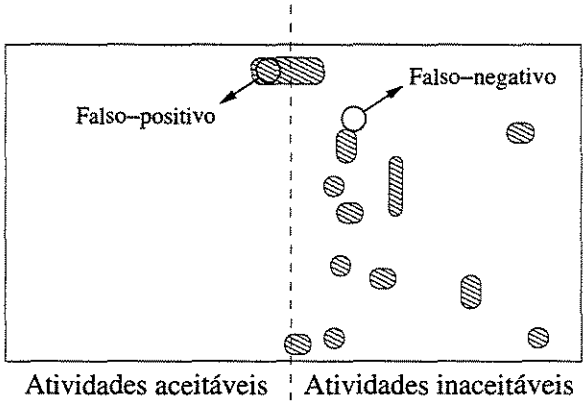

Figura 2.2: Funcionamento da detecção baseada em conhecimento.

Na detecção baseada em conhecimento, o banco de dados de assinaturas compreende uma representação da entidade Especificações da Política de Segurança, apresentada anteriormente na Figura 2.1. Nesse caso, são especificadas somente as ações que causam uma violação da política de segurança em questão.

2.3.3.2 Detecção baseada em comportamento

A detecção baseada em comportamento consiste em construir um modelo de referência para o comportamento usual do sistema monitorado, buscando, posteriormente, identificar situações onde o comportamento analisado desvia-se do comportamento observado usualmente [DDW99]. Esse método considera que ataques produzem um comportamento diferente do comportamento usual [BM01] e reflete a visão de que intrusões estão inseridas em um subconjunto das atividades não usuais [Bac00].

Para conhecer o comportamento usual de um sistema, a detecção baseada em comportamento envolve um processo de estabelecimento de perfis no sistema monitorado. Em geral esses perfis consistem em um conjunto de métricas que são associadas a limiares ou

faixas de valores observados. Várias técnicas podem ser utilizadas na construção de perfis, baseadas, por exemplo, em estatísticas, regras, redes neurais, sistemas imunológicos artificiais, etc. Essas técnicas serão apresentadas na Seção 2.4, juntamente com as principais técnicas existentes para a detecção baseada em conhecimento.

Os perfis são construídos através de um histórico de informação coletado durante um período de atividade normal do sistema. Em geral, é aconselhável que nesse período o sistema monitorado esteja livre de ataques. Após o estabelecimento dos perfis do sistema, quando a atividade monitorada desvia-se do comportamento usual, é disparado um alarme.

As principais vantagens da detecção baseada em comportamento são [BM01, DDW99]:

- Como este método detecta comportamento não usual, ele possibilita identificar sintomas de ataque sem manter conhecimentos específicos sobre o ataque. Dessa maneira a detecção baseada em comportamento torna-se apropriada para detectar ataques desconhecidos;
- A detecção baseada em comportamento pode produzir informações que podem ser utilizadas na confecção de assinaturas de ataques.

As desvantagens de utilizar a detecção baseada em comportamento são [BM01, DDW99, Axe00]:

- Em geral, esse método produz uma alta taxa de falso-positivos devido a constantes mudanças existentes no sistema monitorado;
- A detecção baseada em comportamento freqüentemente requer um extensivo e constante treinamento para possibilitar a caracterização do comportamento usual.

A Figura 2.3 ilustra o funcionamento da detecção baseada em comportamento. Nessa figura as atividades realizadas no sistema computacional são classificadas como aceitáveis ou inaceitáveis, de acordo com a política de segurança em questão. As regiões hachuradas compreendem o modelo do que é considerado usual. O espaço restante compreende atividades que não se encaixam no comportamento usual modelado. Em geral, somente uma pequena parcela do espaço das atividades inaceitáveis acaba sendo modelada como usual. Sendo assim, essa abordagem possibilita a detecção de muitos ataques desconhecidos. Devido a constantes mudanças em um sistema computacional, uma parcela razoável das atividades aceitáveis poderá não ser considerada nos perfis, ocasionando muitos falso-positivos.

Na detecção baseada em comportamento, os perfis do sistema monitorado compreendem um modelo para a entidade Especificações da Política de Segurança, apresentada

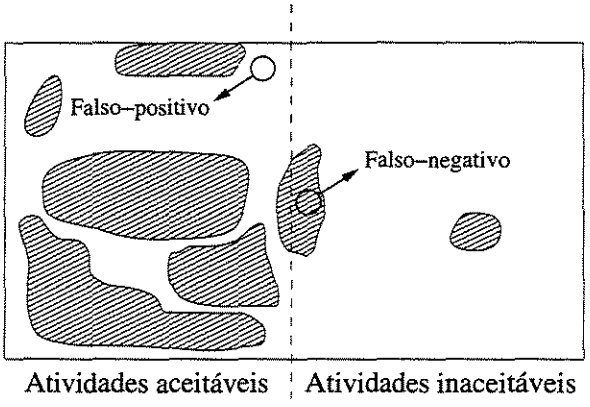

Figura 2.3: Funcionamento da detecção baseada em comportamento.

anteriormente na Figura 2.1. Nesse caso, os perfis procuram representar ações que são permitidas, de acordo com a política de segurança em questão.

2.3.3.3 Combinação dos métodos de análise

Significativas vantagens podem ser obtidas associando os dois métodos de análise expostos. A detecção baseada em comportamento possibilita identificar ataques desconhecidos ou cenários que são inerentes a esse método, como ataques de negação de serviço (*denial-of-service attacks* [NdG03], em inglês). A detecção baseada em conhecimento pode complementar um detector baseado em comportamento, garantindo que um adversário paciente não venha mudar os perfis gradualmente, de tal forma que um ataque seja considerado um comportamento usual [Bac00]. Um IDS híbrido é aquele que combina esses dois métodos de análise.

2.3.3.4 Definindo a detecção baseada em evidências de intrusão

Esta seção apresenta a detecção baseada em evidências de intrusão, que é um método de análise proposto e utilizado neste trabalho. Porém, antes de apresentar esse método, é feito um breve comentário sobre o cenário típico de uma intrusão.

O modo de operação de um intruso pode sofrer algumas variações, de acordo com a sua habilidade e a finalidade do ataque [dR03]. Entretanto, os passos efetuados por um atacante em um cenário típico de intrusão compreendem [IH02, Cen03]:

1. Identificar o alvo do ataque;

2. Identificar vulnerabilidades existentes;

3. Comprometer inicialmente o alvo em busca de acesso ao sistema computacional;
4. Ganhar acesso privilegiado;
5. Explorar o sistema;
6. Retirar as evidências produzidas;
7. Instalar *backdoors* [SG96];
8. Atacar outras máquinas, acessar ou alterar informações, ou iniciar outra atividade não autorizada.

Invariavelmente serão produzidas evidências durante uma intrusão [Ste00a]. De fato, o comprometimento inicial do sistema irá perturbar bastante o alvo do ataque [IH02]. Para efetuar as ações contidas a partir do Passo 3, um atacante irá subverter processos privilegiados de modo a realizar acessos inesperados, executar serviços não autorizados, editar arquivos de configuração, editar *logs* do sistema, estabelecer comunicação não autorizada, alterar informação de uma forma não usual ou efetuar outras violações relacionadas [IH02]. Sendo assim, é possível definir um método de análise baseado na identificação de evidências de intrusão.

Definição *A detecção baseada em evidências de intrusão consiste em identificar atividades que invariavelmente irão ocorrer em um cenário típico de intrusão.*

Durante a identificação de uma intrusão, o Analisador deve considerar apenas as atividades que foram executadas com sucesso no sistema monitorado, isto é, as evidências encontradas. Dessa forma, esse método possibilita uma redução na taxa de falso-positivos, uma vez que acessos não autorizados pelo alvo do ataque são desconsiderados na detecção. Dado que em um cenário típico de intrusão serão produzidas diversas evidências, como as listadas anteriormente, a detecção baseada em evidências de intrusão pode também apresentar uma baixa taxa de falso-negativos. Além disso, por detectar evidências que ocorrem em ataques típicos em geral, esse método possibilita ainda a detecção de ataques desconhecidos. Resultados experimentais descritos no Capítulo 6 também reafirmam essas hipóteses.

Ao trabalhar com evidências de intrusão, as seguintes considerações devem ser feitas antes de utilizar esse método:

- Um ataque que não é realizado com sucesso não é detectado. Entretanto, deve ser notado que outra instância desse ataque poderá resultar em uma intrusão em outro sistema, devido a diferentes versões de *software* e de *hardware*;

- Intrusões baseadas na estratégia tentativa-e-erro, como a adivinhação de senhas de usuário, poderão não ser identificadas. Nesse caso, a detecção baseada em comportamento representa o melhor método dentre os analisados;

- Esse método pode falhar na identificação de cenários de intrusão atípicos. Nesse caso, entretanto, o intruso não produzirá alterações suficientes no sistema monitorado, geralmente indicando uma intrusão de baixa gravidade ou a ação de um intruso com conhecimentos muito elevados;

- A identificação da intrusão acontece somente após a ocorrência inicial da intrusão. Dessa forma, o IDS deve ser protegido de maneira a não ser desabilitado. Essa proteção pode ser implementada no nível do sistema operacional, restringindo o acesso ao IDS através de potenciais alvos como, por exemplo, serviços de rede;

- Como apresentado anteriormente, a estratégia de monitoramento baseada em rede em geral não produz realimentação a respeito do sucesso do ataque. Portanto, a detecção baseada em evidências de intrusão deve considerar especialmente os monitoramentos baseados em máquina e aplicação, herdando suas características já descritas.

A detecção baseada em evidências de intrusão está fundamentada principalmente no sucesso de ações que irão ocorrer em um cenário típico de intrusão. Dessa forma, a identificação de intrusões pode considerar um conjunto de regras de detecção pré-especificadas, o que remete à detecção baseada em conhecimento. Entretanto, ao contrário do uso de assinaturas específicas, essas regras não necessitam ser atualizadas constantemente porque, independente da técnica empregada no ataque ou vulnerabilidade explorada, o atacante irá desenvolver ações típicas.

Esse método pode ser implementado através da análise de regiões de uma máquina que são freqüentemente acessadas indevidamente durante uma intrusão. Essas regiões incluem os diretórios de binários do sistema, os diretórios de arquivos de configuração do sistema ou aplicações, arquivos de *log*, *scripts* de início/término de serviços, etc. Esses acessos também podem ser correlacionados com usuários e/ou processos em execução.

Será utilizado no restante do texto o termo "detecção baseada em evidências" como um sinônimo para a detecção baseada em evidências de intrusão.

2.3.4 Momento da análise

Após a escolha do método de análise adequado para um sistema de detecção de intrusão, é necessário definir o momento em que tal análise irá ocorrer. Esse momento refere-se

ao tempo passado entre o monitoramento das informações e a análise dessas informações [BM01]. A análise pode ser conduzida em tempo real ou em modo *batch* [Bac00, BM01].

Um IDS que trabalha em tempo real analisa continuamente a informação proveniente do componente Fonte de Informação. Nesse caso, o processo de análise deve ser suficientemente rápido para permitir que o processo de identificação viabilize afetar o progresso ou o resultado de uma intrusão. Esse modo de operação é predominante em IDSs que utilizam o monitoramento baseado em rede.

Na análise em modo *batch*, o fluxo de informação proveniente do monitoramento para a análise não é contínuo. Sendo assim, a informação coletada é armazenada em arquivos e, em um momento conveniente, é analisada. A análise em modo *batch* prevalecia nos sistemas de auditoria primordiais, uma vez que a capacidade de processamento e comunicação não suportava um monitoramento e análise em tempo real.

2.3.5 Resposta

Um dos objetivos da detecção de intrusão consiste em, após identificar um ataque, prover uma resposta para dificultar ou impedir o progresso de um ataque. A resposta de um IDS pode ser classificada como passiva ou ativa, descritas como segue.

2.3.5.1 Resposta passiva

Uma resposta passiva provê as informações sobre o ataque aos usuários do sistema, delegando a eles as ações subseqüentes [BM01]. Muitos IDSs comerciais utilizam somente mecanismos de resposta passiva.

Exemplos de resposta passiva compreendem o registro das informações sobre o ataque em um arquivo, uma notificação no console do IDS e o envio de uma mensagem eletrônica ao administrador. Apesar dessa abordagem ser bastante simples, ela é bastante recomendada quando o IDS produz um número considerável de falso-positivos.

2.3.5.2 Resposta ativa

Uma resposta ativa compreende um conjunto de ações automatizadas que são executadas quando certos tipos de ataque são detectados [BM01]. Uma resposta ativa pode ser classificada em uma dentre três categorias:

- Coleta de informação adicional: consiste em coletar informações adicionais a respeito de um provável ataque. Essas informações podem ser obtidas através do aumento do número de eventos registrados ou através de um ajuste na sensibilidade do monitoramento;

- Alteração no ambiente: baseia-se em terminar um ataque em progresso ou bloquear ações subseqüentes. Exemplos de resposta incluem a utilização de um *firewall* para bloquear determinados endereços IP ou portas e a injeção de pacotes TCP com a opção *reset* ativada[3];

- Contra-ataque: é a forma de resposta mais agressiva e consiste em lançar ataques contra o suposto atacante ou tentar ativamente obter informação a seu respeito. Devido a princípios legais, essa abordagem pode representar um grande risco para a organização que a exercita.

O uso da resposta ativa para bloquear recursos do sistema deve ser considerado somente quando a taxa de falso-positivos é baixa. Caso contrário, a resposta poderá impedir comunicação e acesso legítimos, prejudicando a funcionalidade do sistema computacional.

2.3.6 Estratégia de controle

A estratégia de controle descreve como os componentes de um IDS serão controlados. Existem duas estratégias principais para controlar um IDS que monitora múltiplas máquinas ou redes: a centralizada e a distribuída.

A estratégia centralizada considera que um ponto central controla todos os componentes do IDS. Essa estratégia facilita a administração do sistema de detecção, mas deve prover mecanismos para: trocar mensagens entre os componentes de maneira segura, determinar quando um componente está sendo prejudicado, iniciar e terminar a execução de cada componente de maneira correta.

A estratégia distribuída consiste em utilizar uma abordagem baseada em agentes para o monitoramento e análise da informação. Nesse caso, a resposta é executada a partir do local onde é identificado o ataque.

2.4 Técnicas de análise

A Seção 2.3.3 apresentou os dois métodos clássicos para a análise de informação visando a detecção: a detecção baseada em conhecimento e a detecção baseada em comportamento. Entretanto, diversas técnicas podem ser empregadas na implementação de cada um desses métodos. Esta seção descreve as principais técnicas existentes, descrevendo os IDSs ou pesquisas precursoras em cada técnica. Para facilitar o entendimento, essa descrição é dividida em três classes: detecção baseada em conhecimento, detecção baseada em comportamento e métodos alternativos de detecção.

[3]Mais informações a respeito da pilha de protocolos TCP/IP podem ser encontradas em [Ste00b].

2.4.1 Detecção baseada em conhecimento

Para desenvolver um bom detector baseado em conhecimento é necessário entender quais atividades constituem uma violação da política de segurança, manter registros confiáveis das atividades dos usuários e prover uma técnica para analisar esses registros. Em resumo, a detecção baseada em conhecimento é adequada para detectar o uso de padrões conhecidos. A seguir são apresentadas as principais técnicas utilizadas na detecção baseada em conhecimento.

2.4.1.1 Sistemas de produção/especialistas

Os sistemas de produção/especialistas compreendem uma das primeiras técnicas utilizadas na detecção baseada em conhecimento. Essa técnica foi utilizada em sistemas como MIDAS, IDES, Next Generation IDES (NIDES), DIDS [Bac00, Axe00]. Os sistemas MIDAS, IDES e NIDES empregavam o sistema de produção P-BEST [W+88], enquanto o sistema DIDS empregava a ferramenta para desenvolver sistemas especialistas CLIPS [Gia92].

Essas técnicas permitem que usuários do IDS entrem com o conhecimento relacionado a ataques na forma de regras "se-então". Condições que indicam um ataque são especificadas na parte "se" da regra. Quando uma condição é satisfeita, são executadas as ações da parte "então" da regra.

A principal vantagem associada a sistemas de produção/especialistas é permitir que os usuários possam especificar as regras de análise sem conhecer o funcionamento interno da técnica de detecção. Porém, alguns problemas práticos estão associados ao uso dessas técnicas:

- São impróprias para manipular grandes volumes de dados, uma vez que geralmente são implementadas através de sistemas interpretados;
- Não provêem uma manipulação natural de dados ordenados seqüencialmente;
- A qualidade da análise é dependente do nível de conhecimento de quem produz as regras;
- Não permitem manipular situações de incerteza;
- Não são escalonáveis e adaptativos.

Manter uma base de regras também pode proporcionar um grande problema, uma vez que a alteração em uma regra pode afetar o restante das regras na base de conhecimento.

2.4.1.2 Técnicas baseadas em transição de estados

Técnicas baseadas em transição de estados possibilitam estruturar o problema da detecção de forma a permitir o emprego de algoritmos otimizados de casamento de padrões. Essas técnicas destacam-se pela velocidade de análise e flexibilidade de uso, tornando-as ferramentas poderosas de detecção.

A detecção baseada em transição de estados utiliza expressões referenciando o estado do sistema monitorado para identificar intrusões conhecidas. Embora várias abordagens possam ser utilizadas para implementar essas técnicas, três se destacam: a análise de transição de estados, redes de Petri coloridas e as abordagens baseadas em linguagem.

Detectores que implementam a análise de transição de estados utilizam um autômato finito para especificar intrusões. Uma especificação é composta por uma seqüência de ações que conduzem de algum estado inicial do sistema até um estado que representa uma intrusão. O estado inicial representa o estado do sistema antes de uma intrusão, enquanto o estado final representa o estado do sistema após a intrusão. A análise de transição de estados foi inicialmente proposta no IDS STAT [Por92], onde o estado do sistema monitorado era descrito através de atributos desse sistema ou privilégios de usuários.

Outra abordagem baseada em transição de estados, que permite uma detecção bastante otimizada, consiste no uso de redes de Petri coloridas. Essa abordagem é descrita em [KS94b] e implementada pelo IDS IDIOT [Axe00]. Nesse caso, é efetuado um casamento de padrões através da movimentação progressiva de um conjunto de *tokens* a partir de estados iniciais até um conjunto de estados finais, indicando um ataque ou intrusão. Apesar dessa abordagem parecer idêntica à análise de transição de estados, existem significativas diferenças entre elas. A principal diferença reside no fato de STAT identificar uma intrusão através de uma seqüência de passos observados no sistema. A detecção de IDIOT, por sua vez, consiste em um casamento de padrões que é realizado comparando uma assinatura do ataque com os estados atuais dos *tokens*.

As abordagens baseadas em linguagem apresentam um ambiente para a especificação de atividades que representam ataques. Esse ambiente inclui o uso de estruturas de dados, as quais representam o estado atual do sistema, e também a especificação de funções de transição de estados, as quais conduzem a um estado final correspondente à detecção. As abordagens baseadas em linguagem compreendem uma estratégia utilizada por produtos comerciais de detecção baseada em conhecimento, destacando-se o IDS NFR [Sec04].

Em geral, as técnicas baseadas em transição de estados apresentam as seguintes vantagens:

- Possibilitam uma especificação intuitiva de cenários de ataque, de acordo com uma seqüência lógica de eventos;

- Efetuam a identificação de ataques de um modo bastante rápido, permitindo processar um grande volume de informação;
- São capazes de identificar ataques coordenados e lentos.

A principal desvantagem dessas técnicas provém da necessidade de uma especificação manual de todo o conhecimento sobre os ataques que se deseja identificar. As demais limitações existentes são aquelas já discutidas sobre a detecção baseada em conhecimento.

2.4.1.3 Recuperação de informações para análise em modo *batch*

Embora muitos sistemas de detecção de intrusão tenham como objetivo a identificação em tempo real, o emprego da análise em modo *batch* pode ser interessante em situações onde deseja-se buscar pela ocorrência de determinados padrões de atividades. Essa análise é de especial interesse para investigadores e equipes de resposta a incidente.

Essa abordagem é limitada a investigar informações após a ocorrência de certos padrões que são identificados por um investigador como um ataque. Para efetuar a análise são utilizados sistemas de aquisição de informações [AK98], assim como eles são utilizados em ferramentas de busca na Internet, citando, por exemplo, a ferramenta AltaVista [Alt04].

Sistemas de aquisição de informações utilizam um conjunto de arquivos para a indexação da informação, permitindo que a busca por uma palavra-chave ou combinação de palavras-chave seja realizada eficientemente. Esse processo pode identificar rapidamente e de maneira confiável os ataques pesquisados. Os sistemas de aquisição de informações diferem da mineração de dados pela utilização de índices, ao invés do aprendizado de máquina, para identificar padrões.

2.4.2 Detecção baseada em comportamento

A detecção baseada em comportamento assume que o sistema computacional monitorado exibe um padrão de atividade previsível e consistente. Esse método de análise é capaz de se adaptar a mudanças de comportamento ocorridas no sistema monitorado através da manutenção de um conjunto de perfis. Por suas qualidades, esse método tem sido implementado em diversos IDSs através do desenvolvimento de várias técnicas de análise. Esta seção apresenta as principais técnicas desenvolvidas, as quais servem como base para o desenvolvimento de sistemas de detecção atuais.

2.4.2.1 Modelo original de Dorothy Denning

Dorothy Denning apresentou quatro modelos estatísticos que poderiam ser incluídos em seu sistema IDES [Den86]. Cada modelo é considerado apropriado para um tipo particular de métricas do sistema. Esses modelos são descritos como segue:

Modelo operacional

O modelo operacional aplica-se a métricas como, por exemplo, contadores para o número de falhas de autenticação em um particular intervalo de tempo. O modelo compara a métrica com um limiar especificado, disparando um alarme quando a métrica excede o valor desse limiar. Esse modelo, que pode ser aplicado também na detecção baseada em conhecimento, corresponde à detecção por limiar, apresentada adiante.

Modelo de média e desvio padrão

O segundo modelo de Denning propõe o uso dos parâmetros média e desvio padrão para caracterizar a informação. A premissa desse modelo considera que esses parâmetros são as únicas referências do sistema monitorado conhecidas pelo Analisador. Um comportamento é considerado não usual quando ele reside fora de um intervalo de confiança pré-estabelecido. Esse intervalo de confiança é definido como d desvios padrões a partir da média, para algum valor de d. Denning supõe ainda que essa caracterização é aplicável a contadores, temporizadores e outras medidas de recursos.

Modelo multivariável

O modelo multivariável é uma extensão do modelo de média e desvio padrão. Ele baseia-se em efetuar correlações entre duas ou mais métricas, de modo que a detecção possa ser realizada comparando duas ou mais medidas distintas. Por exemplo, ao invés de identificar somente a duração de sessões de usuários, esse modelo permite correlacionar a duração de sessões com o número de ciclos de CPU utilizados.

Modelo do processo de Markov

O último modelo apresentado por Denning é limitado a contadores de eventos. Esse modelo considera que cada tipo de evento analisado corresponde a uma variável representando um determinado estado do sistema. É utilizada, então, uma matriz de transição de estados que caracteriza as freqüências da transição entre estados, e não as freqüências de cada evento individualmente. Uma observação é identificada como não usual quando sua probabilidade, determinada pelo estado anterior e o valor da transição na matriz, é bastante baixa. Esse método permite identificar uma seqüência de comandos não usuais, ao invés de um único evento. Utilizando estados, o modelo do processo de Markov introduz a noção de efetuar a análise baseada em uma memória de estados (*stateful analysis*, em inglês).

2.4.2.2 Análise quantitativa

A abordagem mais comumente utilizada na detecção de anomalia é a análise quantita-

tiva, onde as regras de detecção são expressas em uma forma numérica. Essa abordagem presume o uso de alguma computação, que pode variar desde uma simples soma até complexos cálculos criptográficos. Esta seção descreve os principais tipos de análise utilizados, que servem tanto para a detecção baseada em comportamento quanto para a detecção baseada em conhecimento.

Detecção por limiar

Provavelmente a forma mais comum de análise quantitativa é a detecção por limiar. Nessa detecção, o comportamento é caracterizado através de contadores, que possuem algum nível pré-estabelecido como permissível. Praticamente todos os primeiros IDSs implementavam a identificação de ataques através dessa medida.

Exemplos da utilização dessa técnica incluem a verificação do número de conexões de um tipo particular, o número de tentativas de acesso a arquivos, o número de arquivos e diretórios acessados, a quantidade de senhas incorretas e o número de máquinas acessadas através da rede. A detecção por limiar assume que a verificação dessas medidas é feita considerando um intervalo de tempo determinado. Esse intervalo pode iniciar em um momento fixo do tempo ou pode considerar os últimos eventos ocorridos.

Detecção por limiar heurística

Ao invés de utilizar um limiar fixo, a detecção por limiar heurística consiste em calcular e adaptar esse limiar através de níveis observados. Então, essa técnica permite, por exemplo, disparar um alarme quando o número de falhas de autenticação excede um número usual de falhas observado, incluindo possivelmente algum intervalo de confiança. Esse processo aumenta a precisão da detecção, especialmente quando é utilizado em um ambiente que comporta uma diversidade de usuários ou máquinas.

Verificadores de integridade

Um verificador de integridade baseia-se em identificar alterações ocorridas em objetos do sistema monitorado. Em geral esses objetos devem permanecer inalterados por um longo tempo e raramente são atualizados, como, por exemplo, arquivos binários do sistema e arquivos de configuração. Um verificador de integridade pode ser implementado através da utilização de *hashes* criptográficos[4]. Depois que o *hash* é calculado sobre o objeto monitorado e armazenado em uma mídia segura, o sistema calcula periodicamente um novo *hash* sobre esse objeto, comparando com o valor original. Se é encontrada alguma diferença, um alarme é disparado. O produto Tripwire [KS94a] implementa essa técnica de detecção.

[4]Mais informações a respeito de *hashes* criptográficos podem ser encontradas em [SG96].

Análise quantitativa e redução de dados

Os primeiros IDSs desenvolvidos utilizavam a análise quantitativa para obter redução de dados. Redução de dados é o processo de eliminar informações supérfluas ou redundantes contidas em volumosos registros de eventos. Com essa redução é possível diminuir espaço requerido para o armazenamento dos registros, além de otimizar o processo de detecção. O sistema NADIR [H+93] implementa essa técnica, transformando a atividade de usuários, obtidas através de registros de auditoria, em vetores de medidas quantitativas. Os perfis são agregados por intervalo de tempo e atividades de usuários, sendo posteriormente submetidos a análises estatísticas e sistemas especialistas.

2.4.2.3 Medidas estatísticas

Os primeiros exemplos práticos da detecção baseada em comportamento eram fundamentados no emprego de medidas estatísticas. Esta seção descreve dois sistemas de detecção de intrusão que utilizavam essa técnica: IDES [L+90], citado anteriormente, e Haystack [Sma88].

IDES

A análise estatística empregada por IDES e, posteriormente, NIDES, suporta um histórico de perfis estatísticos construído e mantido para cada usuário e sistema monitorado. Esses perfis são atualizados periódica e gradativamente de maneira a refletir mudanças de comportamento observadas. A atualização de perfis é feita uma vez por dia, incorporando os novos registros de auditoria obtidos de acordo com a atividade de cada usuário ou sistema durante esse dia. As medidas utilizadas compreendem o acesso a arquivos, terminais usados e tempo utilizado de CPU. Os registros de auditoria são correlacionados através de uma representação matricial, gerando um sumário estatístico chamado IDES *score* (IS). Esse sumário representa a similaridade entre o comportamento analisado e as informações contidas nos perfis, com grandes valores indicando comportamento não usual [JV91].

Haystack

Haystack é um sistema de detecção baseada em comportamento desenvolvido para a Força Aérea dos Estados Unidos. Esse sistema emprega a análise estatística utilizando dois critérios de detecção:

1. A proximidade do comportamento de usuários em relação a determinadas características consideradas intrusivas;

2. O desvio existente entre o comportamento de usuários e o comportamento usual.

A utilização desses dois critérios conjuntamente resolve muitos problemas associados à utilização separada de cada critério de detecção [Axe00].

2.4.2.4 Vantagens e desvantagens associadas à análise estatística

Uma vantagem existente é que, quando bem projetada, a análise estatística não requer constantes atualizações nos perfis estabelecidos. O uso de medidas estatísticas na detecção também pode revelar atividades interessantes e suspeitas, levando à descoberta de vulnerabilidades. Uma primeira desvantagem dessa técnica está relacionada à dificuldade encontrada para efetuar a análise em tempo real, ocasionada pela quantidade de memória requerida para armazenar os perfis de comportamento. Outra desvantagem ocorre porque a análise estatística não permite estabelecer uma ordem seqüencial de eventos, mas somente comparar o evento atual com perfis construídos. Por final, a taxa de falso-positivos associada à análise estatística pode ser muito alta, levando alguns usuários a ignorar ou desabilitar o IDS.

2.4.2.5 Medidas estatísticas não paramétricas

Os primeiros métodos estatísticos utilizados na detecção consideravam uma abordagem paramétrica para caracterizar o comportamento monitorado. Essa abordagem assume que os dados analisados obedecem a uma determinada distribuição de probabilidade. Por exemplo, nas primeiras versões de IDES assumia-se que os padrões de usuários obedeciam a uma distribuição gaussiana ou normal. Portanto, quando essas premissas eram inválidas o número de falso-positivos tornava-se muito alto.

Pesquisadores da Universidade de Tulane propuseram uma maneira de superar esse problema através do uso de técnicas não paramétricas na detecção [LB91]. O método proposto consiste em classificar a informação através do emprego de técnicas de clusterização de dados. Um pré-processamento é realizado extraindo características particulares de um objeto monitorado — geralmente um usuário do sistema — que são convertidas em uma representação de vetor. Posteriormente, um algoritmo de clusterização é usado para agrupar esses vetores em classes de comportamento. Essas classes são construídas de modo que membros de uma mesma classe sejam bastante similares entre si, e membros de classes diferentes sejam bastante distintos.

A detecção baseada em comportamento não paramétrica presume que as atividades analisadas distinguem-se em duas classes: uma indicando atividade usual e outra indicando atividade não usual. Resultados experimentais mostraram que essa técnica era capaz de identificar operações similares no sistema, como uma compilação ou edição de arquivos, e também agrupava padrões de atividade de um determinado usuário. Apesar de apresentar benefícios em relação à análise estatística, uma generalização dessa técnica prejudica a eficiência e a precisão da análise.

2.4.2.6 Técnicas baseadas em regras

Uma técnica de detecção baseada em regras apresenta o mesmo esquema associado à análise estatística. A principal diferença é que essa abordagem utiliza um conjunto de regras para representar e armazenar os padrões observados. Dois sistemas que se destacam nessa abordagem são descritos a seguir: o Wisdom and Sense (W&S) [LV92] e o Time-Based Inductive Machine (TIM) [TCL90].

W&S

O sistema W&S foi o primeiro a efetuar detecção baseada em comportamento através de regras. W&S opera em uma variedade de plataformas e pode caracterizar atividades tanto no nível do sistema operacional como no nível de aplicação. A base de regras pode ser populada manualmente ou gerada através de um histórico de dados de auditoria. As regras refletem um comportamento monitorado anteriormente e são armazenadas em uma estrutura chamada floresta. Os dados valores monitorados são agrupados em classes que comportam operações ou regras associadas. Uma classe pode conter, por exemplo, informações relacionadas a um determinado usuário. Quando uma transação é processada, ela é comparada a eventos da classe em questão. Se os eventos não são contemplados pelo histórico de atividades da classe, ocorre a detecção de um comportamento não usual.

TIM

O sistema TIM utiliza uma abordagem indutiva para gerar dinamicamente regras que definem intrusões. A diferença entre TIM e outros sistemas de detecção baseados em comportamento é que TIM busca identificar padrões em seqüências de eventos, ao contrário de padrões em eventos individuais. Ele implementa um modelo probabilístico do processo de Markov, como proposto por Dorothy Denning. TIM observa um histórico de seqüências de registros, caracterizando a probabilidade de seqüências particulares de eventos ocorrerem. Embora esse sistema tenha a vantagem de identificar seqüências de atividades que levam a um ataque, ele produz uma alta taxa de falso-positivos, especialmente no início da operação do sistema.

2.4.2.7 Redes neurais

A abordagem baseada em redes neurais utiliza técnicas de aprendizado para caracterizar o comportamento monitorado. Essa abordagem não paramétrica trabalha com conjuntos de dados de treinamento, os quais são presumidamente livres de qualquer indicativo de intrusão ou comportamento indesejável. A aplicação de redes neurais na detecção baseada em comportamento tem sido explorada em algumas pesquisas como [DBS92, Can97].

Redes neurais são constituídas por elementos de processamento simples chamados de unidades, os quais interagem ponderadamente através de pesos em suas conexões. O conhecimento de uma rede neural é codificado em sua estrutura, em termos de conexões existentes e seus pesos. O processo de aprendizado ocorre através da modificação desses pesos e da adição ou remoção de conexões.

O processamento de uma rede neural envolve dois estágios. No primeiro estágio, a rede é populada através dos conjuntos de dados de treinamento. No segundo estágio, a rede recebe os eventos e compara-os às referências de comportamento do histórico, determinando similaridades e diferenças. O principal problema em utilizar redes neurais está no fato da detecção não prover qualquer explicação para o comportamento não usual encontrado.

2.4.3 Métodos alternativos de detecção

Algumas pesquisas mais recentes em detecção de intrusão desenvolveram abordagens que não são classificadas como detecção baseada em conhecimento ou comportamento [Bac00]. Através de uma outra visão do problema, essas abordagens podem ser aplicadas para viabilizar ou refinar a detecção de intrusão. Essas abordagens são descritas como segue.

2.4.3.1 Abordagens baseadas no sistema imunológico

Em uma pesquisa inovadora, pesquisadores da Universidade do Novo México abordaram de uma maneira ampla a questão da segurança de computadores [FAPC94, FHSL96, FHS97]. Eles propuseram estudar a questão de como equipar sistemas computacionais de maneira a prover uma auto-proteção. Para tanto, eles descrevem uma série de similaridades existentes entre sistemas imunológicos biológicos e mecanismos de proteção de sistemas computacionais.

Nessas abordagens, a chave para a detecção consiste na capacidade de fazer a distinção entre *self* e *nonself*, isto é, a capacidade de diferenciar o que é próprio do sistema e o que é indesejado ao sistema, respectivamente. Para tanto, algoritmos inspirados no funcionamento do sistema imunológico foram propostos e aplicados basicamente na detecção baseada em comportamento. Resultados demonstraram uma grande capacidade na identificação de ataques com a ocorrência de um baixo número de falso-positivos, o que tornam essas abordagens bastante promissoras.

Os sistemas de detecção de intrusão baseados no sistema imunológico serão apresentados em mais detalhes no Capítulo 3.

2.4.3.2 Algoritmos genéticos

Outra abordagem sofisticada que pode proporcionar a detecção baseada em comporta-

mento consiste no uso de algoritmos genéticos para analisar a informação monitorada. Algoritmos genéticos são elaborados aplicando operadores de seleção, reprodução e mutação sobre uma população de indivíduos candidatos à solução de um problema [Mé98].

A ferramenta GASSATA [Mé98] implementa um algoritmo genético para a detecção de intrusão baseada em conhecimento, analisando registros de auditoria do sistema operacional. Nessa abordagem os ataques conhecidos são representados por uma matriz AE, onde o elemento AE_{ij} indica o número de eventos do tipo i gerados para o cenário de ataque j. É conhecido um vetor R, onde R_j indica o risco inerente ao cenário de ataque j, e um vetor O, onde O_i indica o número de eventos do tipo i observados nos registros de auditoria. O problema da detecção consiste em encontrar um vetor de hipótese H, onde H_j indica se há ou não um cenário de ataque j. Deseja-se maximizar o produto $R \times H$ de tal forma que a hipótese seja realista. Uma hipótese é considerada realista quando a quantidade de cada tipo de evento requerida pela hipótese é menor ou igual ao número constante nos registros de auditoria, isto é $(AE \times H)_i \leq O_i$ para todo i.

Essa técnica demonstrou baixas taxas de falso-positivos e falso-negativos para reconhecer ataques pré-especificados. Por causa da especificação binária do vetor de hipóteses, não é possível identificar múltiplas instâncias de um mesmo ataque. Ainda, essa técnica não permite identificar a localização dos ataques nos registros de auditoria.

2.4.3.3 Detecção baseada em agentes autônomos

Abordagens de detecção baseadas em agentes autônomos são compostas por entidades de *software* independentes. Cada entidade é controlada unicamente pelo sistema operacional e possui uma função bastante específica. Esta seção descreve as principais implementações existentes: Autonomous Agents for Intrusion Detection (AAFID) e EMERALD.

AAFID

O protótipo AAFID implementa uma arquitetura para detecção de intrusão baseada em agentes autônomos [BGFI+98, SZ00]. Essa arquitetura possibilita manipular agentes e possibilitar a comunicação entre eles. Um número qualquer de agentes pode atuar em uma máquina. Todos os agentes dessa máquina repassam suas descobertas para um único *transceiver*. Um *transceiver* monitora a operação de todos os agentes que estão atuando na máquina e tem a capacidade de iniciar, terminar e reconfigurar os agentes. Um *transceiver* ainda pode efetuar uma redução de dados com base nas informações dos agentes e repassar essas informações a monitores. Por sua vez, monitores são responsáveis por controlar e consolidar informações provindas de vários *transceivers*. Dessa forma, é possível detectar ataques envolvendo mais de uma máquina.

Essa abordagem é mais resistente a ataques direcionados ao sistema de detecção e

compreende uma arquitetura bastante flexível e escalável. Monitores, entretanto, podem constituir um ponto central de falhas. Problemas também podem ocorrer devido ao tempo de propagação de eventos até alcançar as entidades monitoras.

EMERALD

O IDS EMERALD utiliza uma abordagem de agentes distribuídos para efetuar tanto a detecção baseada em conhecimento como comportamento [NP99]. EMERALD inclui numerosos monitores locais e permite distribuir seus resultados a detectores globais, que por sua vez consolidam os alarmes.

A análise de informações pode ocorrer em diferentes níveis de abstração. Os monitores podem ser organizados em níveis distintos de maneira a suportar uma redução de dados hierárquica, o que torna o IDS escalável. EMERALD também oferece um grande leque de respostas automatizadas, as quais são de interesse para administradores de redes de larga escala.

2.4.3.4 Detecção baseada em grafos

A identificação de ataques em larga escala foi estudada em [SCC+96], dando origem ao sistema de detecção de intrusão GrIDS. Esse IDS considera a natureza repetitiva de ataques que se espalham rapidamente através de redes de computadores, como é o caso dos *worms* [NdG03]. GrIDS trabalha com uma representação baseada em grafos para identificar múltiplas instâncias de um ataque, representadas através de árvores de propagação. Quando esse grafo alcança um tamanho suficiente, é identificada a ocorrência de um ataque em larga escala.

GrIDS constrói grafos de atividade, os quais representam máquinas e atividades observadas na rede monitorada. No caso de um *worm*, cada nó nesse grafo corresponde a uma máquina infectada. Uma conexão direcionada de um nó A para um nó B indica que B foi infectado através de A. Se, dentro de um intervalo de tempo, um grafo não progride de modo a alcançar um número considerável de nós, então esse grafo é descartado. Porém, se o *worm* continua se espalhando por outras máquinas, esse grafo continuará sendo atualizado e o ataque em larga escala será identificado.

Para identificar um padrão de atividade, GrIDS pode analisar similaridades na rede, como, por exemplo, portas de destino do tráfego de rede e sistemas operacionais envolvidos na comunicação. GrIDS, além de identificar a ocorrência de ataques extensos, pode também encontrar as prováveis fontes iniciais do ataque, considerando a natureza direcionada dos grafos construídos.

Essa abordagem pode ser utilizada para detectar precisamente a ocorrência de padrões que se espalham rapidamente por uma grande rede monitorada. Entretanto, GrIDS pode falhar ao detectar um ataque que, apesar de ocorrer em larga escala, espalha-se lentamente

pela rede.

2.4.3.5 Mineração de dados

O objetivo da mineração de dados na detecção de intrusão é encontrar padrões consistentes de características do sistema que possam ser usados para descrever o comportamento de programas ou usuários [LSM99, LS00]. Essas características, por sua vez, podem ser utilizadas na forma de classificadores para a detecção baseada em conhecimento ou comportamento.

A mineração de dados refere-se ao processo de extrair modelos de uma grande quantidade de dados. Esses modelos freqüentemente descobrem fatos nos dados que não são aparentes através de outros métodos de inspeção. Os algoritmos de mineração mais indicados para o processamento de registros de auditoria são:

- Classificação: separa os dados em categorias pré-definidas. Em detecção de intrusão, um classificador é utilizado para categorizar os dados como usuais ou não usuais;
- Análise relacional: identifica relacionamentos e correlações entre ítens de dados. Em detecção de intrusão, é usada para correlacionar eventos que possam ser utilizados na identificação de intrusões;
- Análise de seqüência: identifica quais seqüências de eventos ocorrem freqüentemente juntas. Esses algoritmos podem ser utilizados para expandir técnicas de detecção de intrusão para incluir medidas estatísticas temporais. Essas medidas podem ser utilizadas, por exemplo, para reconhecer ataques de negação de serviço.

Resultados iniciais demonstram que essa abordagem pode ser utilizada para a construção de modelos de detecção baseada em comportamento. Entretanto, pesquisas adicionais são planejadas para refinar e testar de uma forma mais efetiva essa abordagem.

2.4.3.6 Detecção baseada em especificação

Considerando que aplicações atacadas desenvolvem um comportamento inadequado, algumas pesquisas foram realizadas visando identificar ataques em aplicações [KFL94, SBS99, Pro03]. Essas pesquisas concentram-se no desenvolvimento de mecanismos para especificação e verificação de políticas de acesso orientadas a aplicação.

Para cada aplicação que se deseja monitorar é desenvolvida uma política relacionada contendo, por exemplo, permissões de acesso a arquivos, comunicação e execução de outros processos. Tais políticas são, em geral, especificadas pelo administrador e podem ser construídas para verificar a ocorrência de ataques conhecidos ou desconhecidos.

Essa técnica é bastante eficiente para identificar ataques, mas as pesquisas desenvolvidas apresentam um processo de especificação bastante complicado. A especificação

pode envolver até mesmo a construção de um verdadeiro programa verificador para cada aplicação monitorada [SBS99].

Apesar de, aparentemente, essas pesquisas efetuarem uma espécie de detecção baseada em evidências de intrusão, elas desconsideram o sucesso das operações requisitadas por uma aplicação. Dessa forma, as atividades realizadas por usuários legítimos desprivilegiados podem contribuir para a geração de falso-positivos. Esses alarmes falsos podem ser gerados até mesmo de uma forma intencional, levando o administrador a desabilitar o sistema de detecção.

2.5 Detecção de ataques *buffer overflow*

Dentre várias classes de ataque a sistemas computacionais, os ataques *buffer overflow* destacam-se por sua persistência e importância [NCFF01, CWP+00, LE01]. Esses ataques ganharam notoriedade a partir de 2 de novembro de 1988, como parte do incidente Internet Worm [Spa88, Spa89]. Embora a correção de vulnerabilidades individuais possa ser simples, esse tipo de ataque tem persistido até o presente momento como uma das principais causas de ataques a aplicações [Cen04a]. O resultado de um ataque *buffer overflow* geralmente é uma *shell* para o atacante com privilégios que podem variar desde o usuário `nobody` até o superusuário `root`. É importante notar que um acesso local não privilegiado pode ser utilizado como um trampolim para explorar outras vulnerabilidades, inclusive no nível do sistema operacional [Raf03, Cap04].

Esta seção descreve o funcionamento básico de ataques *buffer overflow*, os quais são objetos de estudo neste trabalho. Em seguida também são apresentadas, em resumo, as principais pesquisas efetuadas na detecção desses ataques.

2.5.1 Funcionamento de ataques *buffer overflow*

Um ataque *buffer overflow* consiste em preencher um vetor de dados, isto é, um *buffer*, com informações de maneira a ultrapassar o limite definido para esse vetor. Desse modo, é possível sobrescrever informações armazenadas em posições de memória adjacentes, o que resulta na alteração da execução legítima de um processo. Esse tipo de problema pode ocorrer em programas que são construídos utilizando linguagens de programação sem checagem de limites, como, por exemplo, a linguagem C. Para alcançar seu objetivo, um atacante pode efetuar a alteração de dados ou injetar código malicioso e executá-lo em seguida.

A seguir é apresentado um exemplo de programa em C vulnerável a ataques *buffer overflow*. Nesse programa, a função `scanf()` efetua a leitura do vetor de caracteres `string`. Entretanto, não é verificado o limite da variável lida, que é de 4 bytes, incluindo o carac-

```
#include <stdio.h>

int main()
{
    char ch = 'a';
    char string[4] = "aaa";

    printf("Antes  => string: %s,  ch: %c\n", string, ch);
    printf("Entre com uma string (no maximo 3 caracteres): ");
    scanf("%s", string); // Não verifica o limite da variavel
    printf("Depois => string: %s,  ch: %c\n", string, ch);

    return 0;
}
```

tere terminador '\0'. Portanto, um atacante pode explorar a leitura dessa variável para sobrescrever o valor da variável ch. Uma possível execução desse programa é apresentada como segue.

```
[root@timo test]# ./programa_vulneravel
Antes  => string: aaa,  ch: a
Entre com uma string (no maximo 3 caracteres): abcdefgX
Depois => string: abcdefgX,  ch: X
[root@timo test]#
```

Nessa execução, foi fornecida a entrada "abcdefgX" para a variável string, apesar do programa solicitar gentilmente no máximo 3 caracteres. Após a leitura, é possível verificar que o valor da variável ch também foi alterado, de 'a' para 'X'.

Uma técnica bastante comum em ataques *buffer overflow* consiste em reescrever, na pilha, o endereço de retorno de uma função [One96]. Nesse caso, o atacante envia uma entrada contendo duas partes: um código malicioso, em sua forma nativa do processador, e um novo endereço de retorno que aponta para esse código. Assim, quando a execução do procedimento local termina, o código inserido é executado, ocorrendo a intrusão.

Embora esses ataques possam receber classificações distintas, de acordo com a região onde ocorre o *overflow*, neste trabalho será usado o termo "*buffer overflow*" para generalizar essa classe completa.

2.5.2 Técnicas de detecção

As principais técnicas utilizadas na detecção de ataques *buffer overflow* são sumarizadas como segue:

- StackGuard [CWP+00] e StackShield [Ven00] são extensões de compilador que efetuam checagens adicionais no endereço de retorno das funções na pilha. Desse modo, evitam apenas ataques *buffer overflow* que alteram os endereços de retorno. PointGuard [CWP+00] é uma generalização do StackGuard de maneira a proteger outros apontadores para código. Em [LE01] e [WFBA00] são apresentadas estratégias para reportar erros na manipulação de vetores através da análise do código fonte de um programa. A maior desvantagem dessas abordagens consiste na necessidade de recompilar todas as aplicações com essas extensões e, talvez por isso, não sejam muito usadas;

- Em [Des02] é apresentada uma estratégia para tornar a pilha não executável. PaX [Tea02] estende essa idéia para a porção de dados de um processo. StackGhost [FS01] torna a pilha não executável através de uma facilidade de *hardware* encontrada na arquitetura Sparc. Um esforço de fabricantes de *hardware* e desenvolvedores de sistemas operacionais também tem sido realizado para colocar a tecnologia *flag* NX em funcionamento [Gre04]. Essa tecnologia permite o sistema operacional utilizar uma *flag* do processador que indica quais páginas são não executáveis. Essas abordagens evitam somente ataques com injeção de código e, em geral, entram em conflito com aplicações que utilizam emissão legítima de código nessas áreas;

- Em [SBS99] é relatada uma abordagem de detecção baseada em comportamento para processos. Para tanto, deve-se estabelecer através de regras, para cada processo, quais eventos são permissíveis. Essas regras envolvem chamadas ao sistema e valores associados a elas. A verificação dos eventos permissíveis é realizada antes da execução das chamadas ao sistema, o que pode ocasionar falso-positivos. Essa abordagem tem a desvantagem de necessitar de uma especificação manual e complexa;

- Estratégias de aprendizado automático são empregadas em [SF00] e [Esc99] para monitoramento de processos. Essas técnicas detectam comportamento não usual observando chamadas ao sistema operacional. Elas necessitam de uma boa fase de aprendizado, sofrem de falso-positivos e detectam o ataque somente após sua ocorrência inicial;

- O uso de *wrappers* para monitoramento de aplicações é apresentado em [KFBK00], utilizando individualmente as detecções baseadas em regras, em assinatura de ataque, e no aprendizado automático. Os resultados refletem o uso dessas técnicas individualmente;

- Uma estratégia bastante inovadora, utilizada para detectar código executável no tráfego de rede é mostrada em [TK02]. Considera-se que pacotes válidos contenham

pequenas seqüências de dados que podem ser vistas como uma cadeia de código executável. Ao encontrar uma cadeia de código com tamanho suficientemente grande, um ataque é detectado. Em testes iniciais essa estratégia mostrou ser virtualmente livre de falso-positivos. No entanto, ela funciona apenas para ataques que injetam código executável;

- Em [KTK02] é descrito uma técnica de detecção baseada em comportamento que analisa o tráfego de rede, tratando serviços distintos de uma forma particular. Embora não seja um sistema projetado especificamente para ataques *buffer overflow*, também pode detectá-los antes de sua ocorrência através da análise do tamanho das requisições. Entretanto, também sofre do problema de falso-positivos, pois utiliza medidas estatísticas e acomoda lentamente a mudança de comportamento nos acessos;

- O conceito "guia de programa" é apresentado em [KBA02], que permite restringir ataques de injeção de código e quase todos de injeção de dados manipulados. É baseado em regras e na especificação de políticas de acesso. Não necessita recompilação de código, uma vez que o programa é executado através de uma espécie de otimização de código dinâmica por um mecanismo proprietário;

- Em [XKPI02] e [SLD02] são apresentados mecanismos de *hardware* para prevenir ataques. A estratégia de [XKPI02] trabalha com uma cópia do endereço de retorno para detectar endereços reescritos e, portanto, não compreende uma solução para o problema geral. Um mecanismo para observar o fluxo dinâmico de informação é descrito em [SLD02]. Esse mecanismo é responsável por detectar desvios indevidos no fluxo de controle de um programa ou a inserção de código executável. Sendo assim, pode deixar de detectar ataques que modificam apenas dados adjacentes ao *buffer* explorado.

Como pode-se notar, diversas abordagens foram pesquisadas buscando soluções específicas para ataques *buffer overflow* e ataques relacionados. Essas abordagens implementam desde técnicas de compilação até a adição de *hardware* especial. Entretanto, nenhuma solução definitiva ainda foi divisada, enquanto diversas vulnerabilidades são descobertas a cada mês.

2.6 Conclusão

Este capítulo apresentou os principais conceitos relacionados à detecção de intrusão. Utilizando os conceitos apresentados foram discutidas as principais abordagens e pesquisas

constantes no desenvolvimento de sistemas de detecção de intrusão. IDSs podem proteger um sistema computacional contra ataques conhecidos ou até mesmo desconhecidos.

A detecção de intrusão baseada em conhecimento incorre em uma baixa taxa de falso-positivos, sendo bastante empregada comercialmente. Entretanto, ela não possibilita identificar a ocorrência de novos métodos de ataque. A detecção de intrusão baseada em comportamento é bastante pesquisada por sua inegável habilidade de detectar novos métodos de ataque. Porém, a ocorrência constante de falso-positivos pode levar um usuário a ignorá-la ou até mesmo desabilitá-la.

Sendo assim, mais pesquisas devem ser realizadas intencionando melhorar a abrangência e a eficácia dos mecanismos de identificação de ataques. A adoção de modelos de detecção/proteção mais apropriados para a segurança de sistemas computacionais também pode trazer incontáveis benefícios a esta importante área.

Capítulo 3

Sistema imunológico e segurança

Uma vez descritos os conceitos e os métodos aplicados na detecção de intrusão, é possível buscar novos modelos que representem melhor as condições atuais da segurança computacional. Sem dúvida, a natureza tem sido utilizada como uma grande fonte de inspiração e, através da sua observação, diversos modelos, teorias e tecnologias foram desenvolvidos [dCZ04]. Em meio a essa fonte de inspiração, destacam-se atualmente o emprego de metáforas, teorias e modelos baseados no sistema imunológico para a solução de problemas de otimização, navegação autônoma e segurança computacional [dCT02].

Este capítulo introduz o sistema imunológico humano de uma forma sucinta, descrevendo os aspectos relevantes para esta pesquisa. Em seguida, serão apresentadas as principais pesquisas em sistemas imunológicos artificiais relacionadas com este trabalho.

A Seção 3.1 descreve o sistema imunológico humano, citando seus principais componentes e o funcionamento da resposta imunológica. A Seção 3.2 apresenta as principais pesquisas que utilizam o sistema imunológico biológico como fonte de inspiração no desenvolvimento de sistemas de detecção de intrusão e sistemas de proteção contra vírus de computador.

3.1 O sistema imunológico humano

É impossível entender como o sistema imunológico pode ser usado para inspirar o desenvolvimento de um sistema de segurança de computadores sem antes compreender seu funcionamento. Esta seção apresenta conceitos fundamentais para a compreensão do sistema imunológico como um sistema de segurança. A seção 3.1.1 apresenta as estruturas básicas do sistema imunológico, enquanto a seção 3.1.2 descreve as etapas da resposta imunológica. A revisão aqui apresentada é amplamente baseada em [dCT02, RBM96, APL97, Qui98, FH00].

3.1.1 Organização estrutural

A defesa oferecida pelo sistema imunológico humano é resultado da ação de dois subsistemas: o sistema imunológico inato e o sistema imunológico adaptativo. Eles compõem o sistema imunológico global, possibilitando respostas contra agentes infecciosos conhecidos ou até então desconhecidos.

O sistema imunológico inato é caracterizado por sua natureza congênita e por sua capacidade limitada de diferenciar um agente patogênico[1] de outro, reagindo de maneira semelhante contra a maioria dos agentes infecciosos. O sistema inato constitui 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. Seus principais componentes são as barreiras físicas e químicas, e as células conhecidas como fagócitos, responsáveis pela ingestão e digestão de uma grande variedade de materiais, tais como outras células, micróbios e partículas estranhas.

Em contraste com o sistema inato, o sistema imunológico adaptativo é capaz de identificar especializadamente um determinado agente patogênico, permitindo uma resposta bastante eficiente. Além disso, ele é capaz de armazenar informações a respeito de um agente infeccioso, de maneira a responder mais vigorosamente em novas exposições a esse agente. Os componentes do sistema adaptativo são os linfócitos — linfócitos T e linfócitos B — e seus produtos, como os anticorpos e as linfocinas[2].

O sistema imunológico inato realiza um papel importantíssimo ao prover sinais coestimuladores para o sistema imunológico adaptativo. Esses sinais são usualmente verificados quando o organismo está sendo danificado de algum modo. Para muitos tipos de patógenos, o sistema adaptativo não pode agir sem essa coestimulação provida pelo sistema inato.

Os mecanismos de ambos os sistemas, inato e adaptativo, constituem um sistema de defesa multicamada em que um grande número de células e moléculas agem cooperativamente. A Figura 3.1 ilustra a arquitetura multicamada do sistema imunológico, com mecanismos de defesa em vários níveis. O sistema inato não provê somente a primeira linha de defesa, mas também atua em diversas etapas da resposta do sistema adaptativo.

3.1.2 Resposta imunológica

A principal função do sistema imunológico é reconhecer e responder às substâncias chamadas de antígenos[3]. Para efetuar essa resposta, o sistema deve executar tarefas de reconhecimento de padrões para diferenciar as células e moléculas do corpo, chamadas *self*, das

[1] Um agente patogênico, ou patógeno, é um microorganismo que causa algum tipo de doença.
[2] Sinais químicos atuantes na resposta imunológica.
[3] Um antígeno é qualquer substância capaz de induzir uma resposta imunológica.

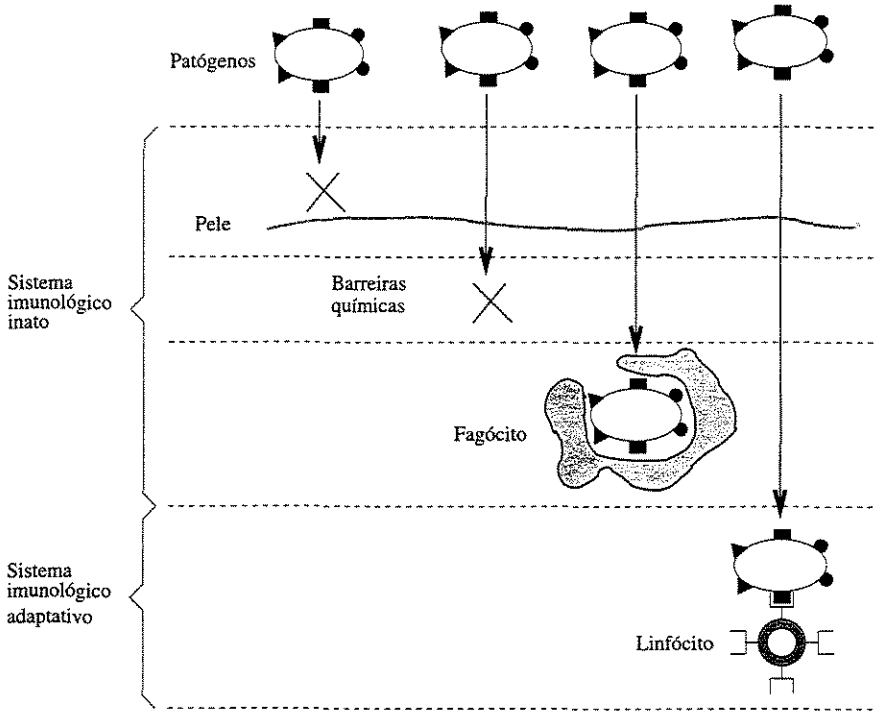

Figura 3.1: Arquitetura multicamada do sistema imunológico.

substâncias estranhas, chamadas *nonself*. Esse reconhecimento de padrões é realizado através da reação entre antígenos e proteínas contidas na superfície das células do sistema imunológico, chamadas receptores. Os receptores funcionam, portanto, como uma espécie de complemento dos antígenos.

Os receptores de um fagócito podem reconhecer um conjunto de antígenos estruturalmente relacionados e, sendo assim, sua detecção não é específica. Os receptores de linfócitos B são produzidos na forma solúvel de anticorpos e podem reconhecer diretamente antígenos livres. Por outro lado, os receptores de linfócitos T não podem reconhecer antígenos livres, mas somente moléculas MHC[4] que expressam fragmentos de antígenos, chamados de peptídeos.

Tanto os receptores de linfócitos B quanto os receptores de linfócitos T efetuam uma detecção específica de antígenos. A Figura 3.2 ilustra a reação de um antígeno com os receptores dos linfócitos, bem como a diferença entre os receptores dos linfócitos B e T.

A habilidade de detectar uma grande quantidade de patógenos é alcançada parcial-

[4]Um complexo de genes que codifica a superfície celular e está envolvido no controle de vários aspectos da resposta imunológica. *Major Histocompatibility Complex*, em inglês.

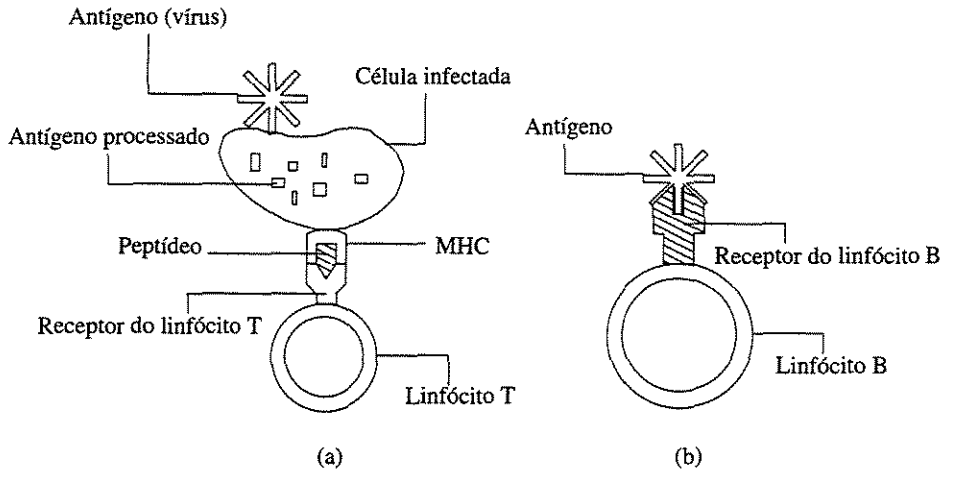

Figura 3.2: A detecção é conseqüência da reação entre estruturas químicas complementares. Em (a) o receptor de um linfócito T reage com uma molécula MHC contendo um peptídeo de antígeno. Em (b) o receptor de um linfócito B reage diretamente com um antígeno.

mente através de um processo de geração aleatória de receptores. Porém, somente os receptores que não reconhecem proteínas *self* são escolhidos através de um processo chamado seleção negativa. Durante esse processo, receptores recentemente criados são expostos a muitas proteínas *self*. Se qualquer receptor reconhecer uma dessas proteínas ele é eliminado. Desse modo, a seleção negativa constrói o conhecimento a respeito do que deve ser considerado anormal para o sistema imunológico.

O sistema imunológico tem, ainda, a capacidade de tornar sua proteção mais específica através de processos de aprendizado e memória. Se o sistema imunológico encontra um patógeno desconhecido, é iniciada uma resposta primária. Durante essa resposta ele assimila a estrutura específica do patógeno, evoluindo um conjunto de células com alta afinidade para tal patógeno através de um processo chamado maturação de afinidade. Em subseqüentes encontros com o mesmo antígeno, ou com antígenos estruturalmente relacionados, o sistema monta uma resposta secundária usando células de memória de alta afinidade, o que torna a resposta mais rápida e precisa. A Figura 3.3 ilustra o tempo necessário para a detecção e eliminação de antígenos em respostas primária e secundária. A resposta secundária é beneficiada pela produção de anticorpos, os quais se ligam aos antígenos, auxiliando no reconhecimento e eliminação da ameaça.

Todas as células e produtos do sistema imunológico circulam pela corrente sanguínea, tecidos e vasos linfáticos, atuando como sentinelas na busca por antígenos. Quando

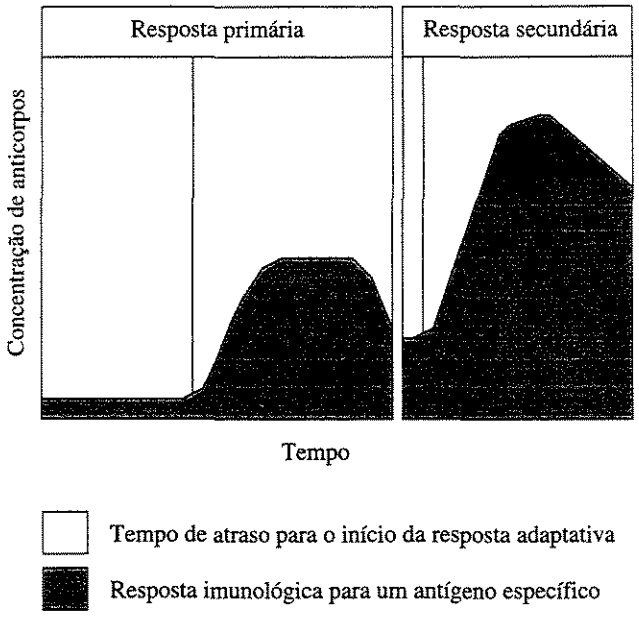

Figura 3.3: Respostas imunológicas primária e secundária.

receptores reagem com antígenos em uma concentração suficiente, um reconhecimento ocorre e um complexo de eventos é disparado, ocorrendo um processo chamado resposta imunológica. A resposta imunológica pode ser dividida em três fases como segue.

3.1.2.1 Fase de detecção

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

Se o antígeno já é conhecido pelo sistema imunológico, anticorpos específicos podem reagir diretamente com ele. Quando anticorpos combinam-se com antígenos, os patógenos tornam-se mais atrativos aos fagócitos e ativam uma cascata de proteínas complementares, que ajudam na destruição dos invasores.

Os linfócitos B também podem reagir com antígenos, 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 os linfócitos T sejam ativados.

3.1.2.2 Fase de apresentação de antígenos e ativação do sistema adaptativo

Linfócitos B e fagócitos, expressando em sua superfície moléculas MHC com peptídeos, atraem linfócitos T que estavam circulando inativos. Se um linfócito T reconhece um determinado complexo MHC-peptídeo, ele torna-se ativo. Os linfócitos T ativos começam então a se reproduzir, diferenciando-se em três tipos:

- Linfócitos T ajudantes: estimulam a ação e multiplicação de linfócitos B e fagócitos;
- Linfócitos T citotóxicos: identificam e destroem as células contaminadas;
- Linfócitos T de memória: armazenam informações a respeito do antígeno para serem utilizadas durante futuras exposições.

Os linfócitos B, estimulados pelos linfócitos T ajudantes, multiplicam-se e diferenciam-se em dois tipos:

- Linfócitos B plasmáticos: secretam moléculas de anticorpos em grande quantidade, possibilitando a neutralização do patógeno;
- Linfócitos B de memória: possibilitam uma melhor identificação desse patógeno e similares em futuras exposições.

Como conseqüência, um repertório específico para o antígeno em questão é reunido.

3.1.2.3 Fase de eliminação de antígenos

Anticorpos específicos reagem com os antígenos, marcando-os para serem destruídos pelos fagócitos e proteínas complementares. Os linfócitos T citotóxicos 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.

3.1.3 Princípios do sistema imunológico

O estudo do sistema imunológico revela um conjunto de características extremamente relevantes para o contexto de segurança computacional [APL97, SHF97, SFA97, dPdCdG04]:

- Distribuição e paralelismo: a detecção de uma célula de defesa não é iniciada por algum mecanismo centralizador, mas quando alguma condição não usual é detectada pela célula. Essa característica descentralizada indica a inexistência de um ponto central de falhas no sistema, garantindo uma maior robustez. Além disso, o sistema imunológico é capaz de agir em diferentes localidades em paralelo;

- Multicamada: nenhum mecanismo do sistema imunológico garante isoladamente a proteção 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: introduzindo uma variedade de indivíduos com características particulares de defesa é possível garantir a sobrevivência de uma espécie. Em outras palavras, a ação de um determinado agente patogênico pode ser fatal para um indivíduo e inofensiva para outro indivíduo;

- Descartabilidade: nenhum componente isolado do sistema imunológico é essencial, podendo ser substituído. O sistema imunológico gerencia essa característica de forma que células afetadas sejam destruídas e substituídas pela produção de novas células;

- Autonomia: o sistema imunológico, como um todo, não requer um gerenciamento ou manutenção externos. Ele pode autonomamente detectar, classificar e eliminar os agentes patogênicos, bem como efetuar a remoção e substituição de suas células danificadas;

- Adaptabilidade e memória: o sistema imunológico possui a capacidade de assimilar a detecção de novos patógenos, armazenando a habilidade de reconhecimento adquirida na memória imunológica. A capacidade de reconhecimento especializa-se a cada agente patogênico similar reconhecido, tornando as respostas futuras mais eficientes se comparadas às anteriores;

- Nenhuma camada de segurança: qualquer célula do corpo humano pode ser atacada por um patógeno, inclusive as células do sistema imunológico. Entretanto, os linfócitos podem proteger o corpo contra linfócitos comprometidos por uma infecção;

- Mudança dinâmica de cobertura: visando maximizar a capacidade de detecção, o sistema imunológico mantém uma amostra aleatória de detectores que circulam pelo corpo. Essa amostra é constantemente renovada através da morte de células e a produção de novas células;

- Identificação por comportamento: uma célula pode ser analisada através das substâncias que ela secreta. Em uma infecção, uma célula contaminada é identificada através da ocorrência de moléculas MHC contendo fragmentos de antígenos;

- Detecção de anomalia: o sistema imunológico tem a capacidade de detectar agentes patogênicos através da identificação de atividade não usual ou anômala;

- Detecção de danos: muitos patógenos não causam ameaças ao corpo e, nesse caso, uma resposta imunológica para eliminá-los poderia prejudicar o próprio corpo. Entretanto, a ocorrência de danos em tecidos de um organismo é essencial para a ativação dos linfócitos T, que irão sobreviver enquanto houver evidências desses danos. Essa característica coestimulatória diminui as chances do sistema imunológico desencadear uma reação autoimune;

- Detecção imperfeita: por possibilitar uma detecção imperfeita ou parcial, o sistema imunológico aumenta a flexibilidade na alocação de recursos. Por exemplo, detectores menos específicos podem responder a uma variedade maior de patógenos, mas irão ser menos eficientes na detecção de um patógeno específico;

- Recrutamento dinâmico: o sistema imunológico regula o número de células de maneira a desenvolver um contra-ataque suficiente para a quantidade atual de agentes patogênicos. Assim que a ameaça é eliminada, essa quantidade de células é reduzida.

Esses princípios organizacionais podem ser utilizados para orientar o desenvolvimento de modelos, teorias e aplicações visando a solução de problemas computacionais diversos. Em particular, esses princípios são bastante apropriados para guiar o desenvolvimento de sistemas de segurança computacional baseados no sistema imunológico.

3.2 Imunologia e segurança computacional

Sistemas biológicos, como os encontrados em seres humanos, podem ser considerados como sofisticados sistemas de processamento de informações [DAO97, Das99a]. Dessa forma, esses sistemas são estudados de maneira a prover inspiração para o desenvolvimento de diversos campos da ciência e engenharia [DAO97, Das99a, dC01]. Por exemplo, a Tabela 3.1 mostra como os princípios do sistema imunológico descritos na Seção 3.1.3 podem ser utilizados para inspirar o desenvolvimento de artefatos para a segurança computacional.

O propósito desta seção é apresentar as principais pesquisas que utilizam uma inspiração baseada no sistema imunológico biológico para desenvolver sistemas de segurança computacional. Em geral, essas pesquisas empregam os princípios do sistema imunológico para modelar sistemas de detecção de intrusão e aplicam o funcionamento desse sistema biológico no desenvolvimento de algoritmos de identificação de ataques. Essas pesquisas são descritas como segue.

Princípios do sistema imunológico	**Segurança computacional**
Distribuição e paralelismo	Utilização de módulos/agentes independentes para monitoramento, detecção e resposta, trabalhando concorrentemente em uma mesma máquina ou distribuídos em uma rede
Multicamada	Diversos mecanismos de proteção podem ser combinados para prover uma maior segurança: IDSs baseados em rede e máquina, detecção baseada em conhecimento e comportamento, *firewalls*, criptografia e outros
Diversidade	Emprego simultâneo de diferentes implementações de sistemas operacionais, *daemons* e aplicativos
Descartabilidade; Nenhuma camada de segurança	Identificação automática de falhas nos mecanismos de segurança, bem como uma restauração desses mecanismos após uma falha
Autonomia	Sistemas de identificação de ataques, resposta e restauração automatizados
Adaptabilidade e memória	Capacidade de analisar uma intrusão de maneira a extrair uma assinatura de ataque, possibilitando uma detecção precisa durante futuras exposições ao mesmo ataque; generalização de assinaturas, visando detectar ataques similares
Mudança dinâmica de cobertura	Renovação de perfis do sistema monitorado para a detecção baseada em comportamento, bem como a adição de variações aleatórias em limiares de detecção
Identificação por comportamento	Detecção baseada em comportamento
Detecção de anomalia	Identificação de comportamento não usual através da detecção baseada em comportamento
Detecção de danos	Tolerância à intrusão e emprego da detecção baseada em evidências de intrusão para coestimular respostas a intrusões
Detecção imperfeita	Casamento parcial de padrões ou de assinaturas de ataque
Recrutamento dinâmico	Alocação dinâmica de recursos, priorizando o processamento dos mecanismos de segurança em situações emergenciais

Tabela 3.1: A aplicação dos princípios do sistema imunológico no desenvolvimento de artefatos de segurança computacional.

3.2.1 Distinguindo *self* de *nonself* em um computador

O sistema imunológico defende o corpo contra doenças e infecções. Essa defesa é possibilitada pela capacidade de reconhecer células ou moléculas estranhas e eliminá-las do corpo. Para tanto, devem ser realizadas tarefas de reconhecimento de padrões para distinguir as células e moléculas do corpo das substâncias estranhas. Sendo assim, o problema de detecção do sistema imunológico consiste em distinguir *self* de *nonself* [SHF97].

Utilizando essa idéia, Forrest, Allen, Perelson e Cherukuri apresentaram um algoritmo para a detecção probabilística de alteração em informações monitoradas [FAPC94]. Esse algoritmo, que é baseado no processo de seleção negativa dos linfócitos T, possui duas fases:

1. A geração de um conjunto de detectores: é construído um conjunto de detectores onde cada detector é uma *string* que não "casa" com os dados protegidos. Essa fase do algoritmo é ilustrada na Figura 3.4;

2. O monitoramento das informações: os dados protegidos são comparados com os detectores. A ativação de um detector indica que ocorreu uma alteração na informação analisada. Essa fase é ilustrada na Figura 3.5.

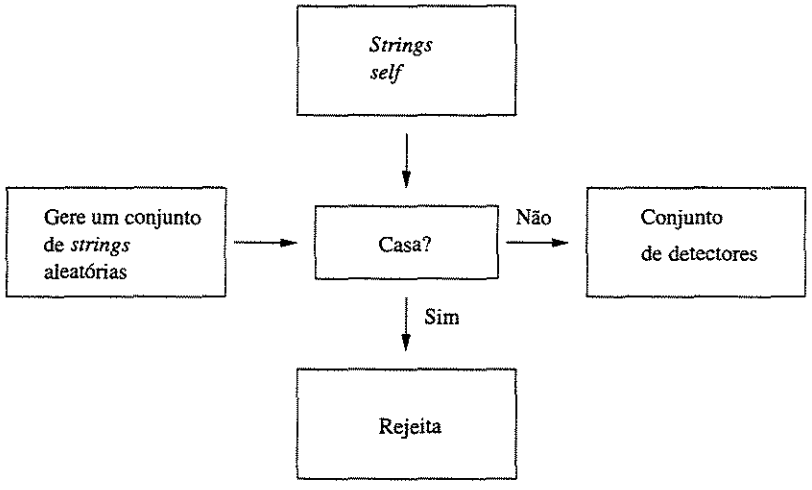

Figura 3.4: Funcionamento da fase de geração de detectores. *Strings* (detectores) são geradas aleatoriamente. Se durante um processo de treinamento uma *string* casa (detecta) com as informações monitoradas (*strings self*), ela é rejeitada. Ao final, o conjunto de detectores será composto somente por aquelas *strings* que detectam o que não é *self*, ou seja, *nonself*.

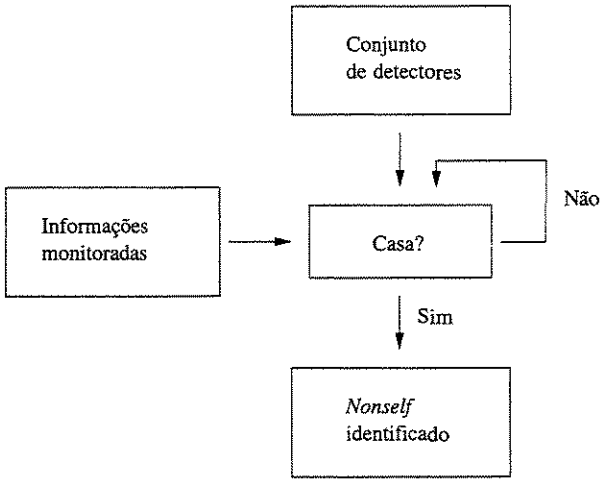

Figura 3.5: Funcionamento da fase de monitoramento. Quando um detector casa (detecta) com as informações monitoradas, ocorre a identificação de uma alteração (identificação de *nonself*).

Nesse algoritmo, cada *string* que compõe um detector é definida como uma seqüência de bits. O casamento de padrões realizado entre as informações monitoradas e os detectores é definido pelo casamento de r bits contíguos, para algum limiar pré-definido r. Quando pelo menos r bits contíguos de um detector casa com a informação monitorada, então ocorre uma detecção.

Esse algoritmo foi aplicado na identificação de alterações em arquivos através da inserção de código, mutação de bytes, e contaminação por vírus de computador. Resultados experimentais demonstraram a eficiência do algoritmo e encorajaram o desenvolvimento de outras pesquisas relacionadas [D'h96, DFH96].

3.2.2 Um significado para *self* em um sistema Unix

Motivados pelos resultados iniciais observados na aplicação do sistema imunológico para a proteção computacional, Forrest, Hofmeyr, Somayaji e Longstaff identificaram um significado para *self* em sistemas Unix [FHSL96]. Esse significado está relacionado com as chamadas ao sistema executadas por um processo sob monitoramento, provendo, dessa forma, um mecanismo capaz de identificar uma série de ataques a aplicações.

Os autores argumentam que ataques ocorrem através de programas em execução, os quais executam freqüentemente chamadas ao sistema operacional. Sendo assim, toda a atenção dispensada no desenvolvimento desse trabalho é voltada para a análise de

chamadas ao sistema. É dada uma atenção especial para processos que executam em modo privilegiado, em geral provendo algum serviço público através da rede.

É argumentado, ainda, que cada programa especifica implicitamente um conjunto de seqüências de chamadas ao sistema, que são produzidas durante a execução. Essas seqüências são determinadas pela ordem de ocorrência das chamadas ao sistema, considerando todas as possíveis alternativas de execução para um programa. Durante uma execução normal, algumas dessas seqüências serão produzidas. Para uma execução que incorpora um ataque, espera-se que haja a ocorrência de seqüências não previstas no comportamento normal. Como a utilização de seqüências com um grande número de chamadas ao sistema poderia inviabilizar o desenvolvimento do sistema de detecção, foi considerada a utilização de seqüências pequenas, indicando uma ordenação local dessas chamadas e sugerindo uma definição simples para *self*.

Por exemplo, supondo que um programa produz em uma execução a seguinte ordem de chamadas ao sistema:

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

e considerando que o tamanho de cada seqüência é 3, então as seguintes seqüências são observadas nessa execução:

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

Nesse trabalho, cada chamada ao sistema é identificada apenas por seu número e são desconsiderados os argumentos repassados pelos programas monitorados. O comportamento usual, isto é, *self*, é definido em termos de seqüências de tamanho 5, 6 e 11.

Para identificar o comportamento não usual são utilizados dois estágios:

1. Modelar *self*: consiste em construir uma base de dados contendo o comportamento usual para cada processo de interesse. Essa base de dados será específica e de acordo com uma arquitetura particular, versão de software e configuração, política administrativa local, e padrões de uso dos programas;

2. Monitoramento e detecção: uma vez que a base de dados construída para um processo esteja estável, essa base pode ser utilizada para monitorar um comportamento observado. Nesse estágio, quando uma seqüência observada em um processo não é contemplada na base de dados ocorre a identificação de uma discrepância. Quando a quantidade de discrepâncias observadas durante uma execução ultrapassa um limiar pré-definido, ocorre a identificação de um comportamento não usual.

Embora esse trabalho tenha considerado sistemas operacionais derivados do Unix, a idéia é portável a qualquer outro sistema operacional que permita observar as chamadas ao sistema que são executadas. Posteriormente, essa linha de pesquisa desencadeou uma série de trabalhos, podendo ser citadas as publicações [SF00, FHS97, HFS98, WFP99].

Resultados experimentais demonstram que essa técnica pode identificar uma variedade de ataques. Entretanto, uma alta taxa de falso-positivos pode ocorrer quando os programas são utilizados de uma forma legítima não usual, como, por exemplo, executando um programa com novos argumentos aprendidos pelo usuário.

3.2.3 Diversidade em sistemas computacionais

Diversidade é uma importante fonte de robustez em sistemas biológicos. Um ecossistema estável, por exemplo, tem seus indivíduos distribuídos em uma grande variedade de espécies. Se essa diversidade é perdida e poucas espécies tornam-se dominantes, o ecossistema fica suscetível a perturbações catastróficas como incêndios, invasões e doenças [SFA97]. Em contraste, sistemas computacionais são notáveis por sua falta de diversidade. Desenvolvedores produzem uma imensa quantidade de cópias idênticas de uma versão de *software* ou *hardware* com o objetivo de tornar esses sistemas compatíveis.

Baseados nessa diferença existente entre sistemas biológicos e sistemas computacionais, Forrest, Somayaji e Ackley estudaram métodos para aumentar a diversidade em computadores, enfatizando problemas relacionados à segurança [SFA97]. Aumentando a diversidade em computadores, almejava-se prevenir o alastramento de ataques através de métodos que dificultassem a replicação de intrusões.

As seguintes diretrizes foram consideradas na busca pela diversidade computacional:

- Preservação da funcionalidade: deve ser mantido, no nível do usuário, o comportamento esperado para diferentes sistemas. Também, o comportamento da entrada e saída de programas deve ser idêntico para diferentes implementações;
- A introdução da diversidade deve ocorrer em determinadas regiões do sistema computacional, de forma a dificultar o sucesso de ataques conhecidos;
- Minimizar custos: considerar o tempo de execução de aplicações, bem como o custo de implantar e manter a diversidade;
- Introduzir diversidade através de aleatorização: embora técnicas de ataque baseadas em conhecimento anterior possam driblar esse mecanismo, é improvável que esse tipo de ataque venha a se espalhar rapidamente.

Os métodos considerados nessa pesquisa incluem variações na localização de instruções durante a compilação, na ordem de execução de instruções, na localização das instruções

na memória durante a execução, no acesso a rotinas externas e arquivos. Como uma demonstração, foi implementado um mecanismo para aleatorizar a pilha de execução de um processo, o que possibilitou interromper um tipo particular de ataques *buffer overflow*.

Recentemente, outra pesquisa abordou o problema da diversidade computacional introduzindo variações no conjunto de instruções a serem executadas [BAF+03]. Nessa abordagem, o conjunto de instruções é embaralhado antes da carga de um programa, sendo posteriormente desembaralhado no momento de execução. Para tanto, foi utilizada uma técnica de tradução de binários combinada com a emulação de instruções. Resultados experimentais demonstram que é possível interromper a execução de códigos maliciosos inseridos durante ataques. Entretanto, essa técnica adiciona um custo significativo durante a execução de programas protegidos e fica à mercê da execução de código aleatório.

3.2.4 Arquitetura para um sistema imunológico artificial

Hofmeyr e Forrest desenvolveram uma arquitetura para o projeto de sistemas imunológicos artificiais chamada ARTIS [FH00]. Essa arquitetura incorpora muitas propriedades do sistema imunológico biológico, incluindo diversidade, distribuição, tolerância a erros, aprendizado dinâmico, adaptação e auto-monitoramento.

ARTIS modela um detector como uma cadeia de bits de tamanho fixo l. Esses detectores combinam propriedades de várias células do sistema imunológico e assumem diferentes estados em diferentes situações. A geração dos detectores é aleatória e obedece o algoritmo de seleção negativa já descrito. A detecção ocorre de acordo com o casamento de r bits contíguos, sendo que um detector torna-se ativo somente após casar com pelo menos τ *strings* dentro de um intervalo de tempo, onde τ é um limiar pré-definido. ARTIS confia o processo de coestimulação a um operador humano, que decide se uma identificação realmente constitui *nonself* ou não. Esse modelo tem um propósito genérico e visa a redução de falso-positivos e falso-negativos.

A Figura 3.6 ilustra o ciclo de vida de um detector na arquitetura ARTIS. Um detector é criado aleatoriamente e permanece imaturo por um certo período de tempo. Se o detector imaturo casar com alguma *string self*, ele morre e é substituído por outro detector aleatório. Se o detector sobreviver a esse processo de seleção negativa, ele viverá por um tempo finito, sendo, ao final de sua vida, substituído por outro detector. Um detector maduro é ativado somente se exceder o limiar τ. Se um detector ativo receber coestimulação, ele torna-se um detector de memória. Caso contrário, ele morre e é substituído.

Baseado nessa arquitetura, foi desenvolvido o sistema de detecção de intrusão LISYS (Lightweight Intrusion Detection System), que é sintetizado a seguir.

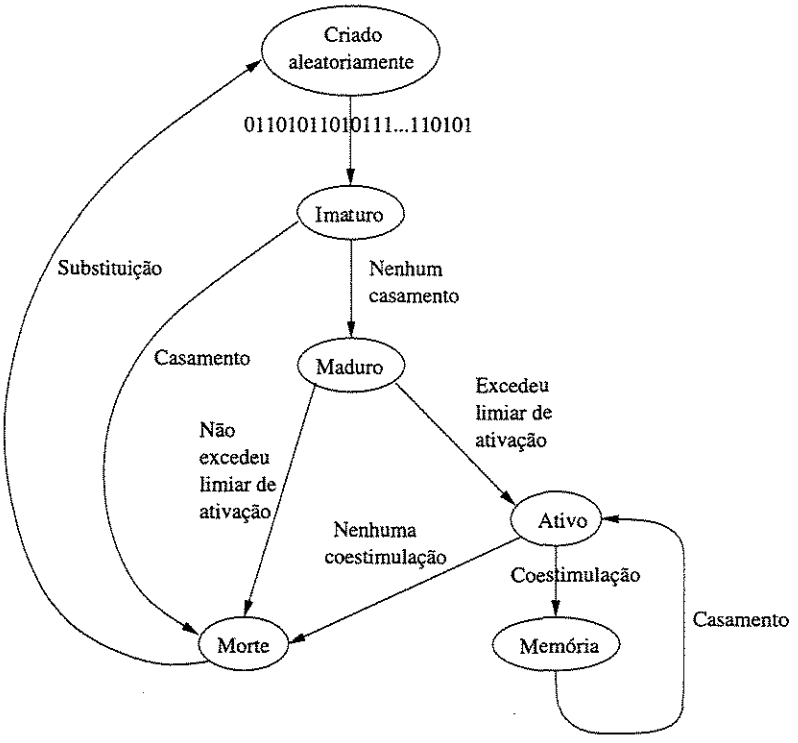

Figura 3.6: Ciclo de vida de um detector modelado na arquitetura ARTIS.

3.2.4.1 O sistema de detecção de intrusão LISYS

LISYS é um sistema de detecção de intrusão distribuído que monitora o tráfego em uma rede *broadcast* TCP/IP [FH00, HF99, FH99, Hof99, HFD98, BFG02, FBGA02]. Nesse domínio, *self* é definido como o conjunto de conexões observadas normalmente na rede monitorada. Cada máquina nessa rede possui um conjunto independente de detectores que é utilizado na identificação dos ataques. Cada detector é representado por uma tupla contendo: endereço IP de origem, endereço IP de destino e o número da porta do serviço acessado.

LISYS utiliza uma representação compactada dessas informações, possibilitando a codificação de cada detector através de uma cadeia de 49 bits. Essa cadeia de bits define unicamente cada conexão. Sendo assim, LYSIS não monitora as informações trocadas em uma comunicação, mas assume que um ataque utiliza conexões imprevistas entre máquinas para um determinado serviço. Resultados experimentais demonstram que esse IDS pode identificar positivamente ataques com uma baixa taxa de falso-positivos. Porém LYSIS falha ao identificar ataques destinados a máquinas que se comunicam com muitas

outras como, por exemplo, servidores FTP e servidores HTTP.

Um trabalho posterior propôs uma versão modificada de LISYS visando utilizar uma nova técnica de casamento de padrões desenvolvida [BEFG02]. Essa nova técnica, chamada *r-chunnks* [EFH04], visa aumentar a eficiência da detecção mas, entretanto, não apresentou superioridade nos testes realizados.

3.2.5 Homeostase e resposta automática a ataques

Todos os sistemas biológicos mantêm um estado interno estável através do monitoramento e resposta a mudanças internas e externas. Esse auto-monitoramento é conhecido como homeostase. Minimizando variações no estado interno de um organismo, os mecanismos homeostáticos ajudam a garantir a operação de numerosos sistemas químicos e mecânicos que constituem um organismo [Som02].

Embora os mecanismos homeostáticos tenham sido extensivamente estudados, muitos ainda permanecem incompreendidos [Som02]. Entretanto, Somayaji e Forrest estudaram dois sistemas homeostáticos, o controle de temperatura e o sistema imunológico, para posteriormente desenvolverem um sistema computacional inspirado nesses sistemas [SF00, Som02].

Esses pesquisadores desenvolveram um sistema de detecção e resposta automática contra ataques, chamado pH (*process homeostasis*). Esse sistema efetua a identificação de ataques através da análise de seqüências de chamadas ao sistema, como descrita na Seção 3.2.2. Porém, o sistema de resposta a ataques é inspirado nos mecanismos de regulação homeostáticos.

A resposta de pH consiste em inserir atrasos na execução de processos. Um atraso ocorre sempre que um processo realiza seqüências de chamadas ao sistema que não estão previstas no comportamento usual modelado. Além disso, esses atrasos são regulados automaticamente do seguinte modo:

- Aumento exponencial: o atraso inserido em um processo é uma função exponencial da quantidade de anomalias verificadas recentemente. Sendo assim, quando o comportamento de um processo desvia-se pouco do comportamento normal esse processo não é muito prejudicado. Em contrapartida, se o comportamento de um processo desvia-se muito do comportamento normal — provavelmente devido a um ataque — então a penalidade imposta será bastante alta, podendo esse atraso chegar à ordem de dias;

- Diminuição gradual: levando em conta que algumas anomalias detectadas podem constituir falso-positivos, o atraso total inserido é diminuído quando o comportamento do processo afetado volta a ser usual. Nesse caso, essa diminuição do atraso é decremental.

Essa abordagem mostrou ser eficiente para a detecção de ataques, podendo inclusive bloqueá-los. Também, a ocorrência de falso-positivos nem sempre prejudica as aplicações monitoradas, o que é devido ao mecanismo de inserção e remoção de atrasos. Entretanto, aplicações que são usadas de uma nova forma, legítima mas diferente do usual, em geral são bastante prejudicadas, necessitando de intervenção humana. Outra desvantagem dessa abordagem ocorre quando um atacante executa poucas chamadas ao sistema para efetuar uma intrusão, impossibilitando uma reação a tempo.

3.2.6 Detecção baseada em agentes imunológicos

O sistema imunológico protege o corpo por meio de um complexo de células e moléculas que possuem funções específicas e trabalham de um modo bastante independente, utilizando, ainda, mecanismos de comunicação ou sinalização. Considerando essas características, Dasgupta propôs uma arquitetura para proteção de computadores baseada em agentes móveis para oferecer as seguintes facilidades: mobilidade, adaptabilidade e cooperação [Das99b].

Nessa arquitetura, os agentes imunológicos podem interagir livremente com outros agentes dentro do ambiente computacional. A mobilidade permite que agentes possam se deslocar na rede para explorar ou monitorar uma dada situação. Esses agentes podem reconhecer a atividade dos outros agentes ao seu redor e produzir uma resposta específica. Além disso, um agente pode se adaptar a seu ambiente dinamicamente e, em colaboração com outros agentes, pode possibilitar uma tomada de decisão complexa.

Essa arquitetura serviu como base para a implementação do sistema de detecção de intrusão SANTA (Security Agents for Network Traffic Analysis) [DB01]. Esse IDS é composto por um conjunto de agentes de *software* que executam funções específicas na rede e colaboram para lidar com o problema da detecção de intrusão. Os seguintes agentes são considerados no sistema SANTA:

- Agentes de monitoramento: capturam informações em computadores executando ferramentas tais como `snoop`, `ps`, `mpstat`, buscando por comportamento não usual. Cada máquina monitorada conta com quatro agentes de monitoramento para observar eventos no nível de rede, processo, sistema operacional e usuário;

- Agentes de comunicação: são responsáveis por repassar mensagens entre os demais agentes. Essa característica é provida nativamente pela plataforma de agentes móveis considerada na implementação;

- Agentes de decisão/ação: utilizam lógica nebulosa para inferir a severidade de um comportamento não usual identificado. Esses agentes são posteriormente utilizados para ativar um ou mais agentes de resposta;

- Agentes ajudantes: são agentes de resposta que fornecem uma interface gráfica entre SANTA e o administrador. Esses agentes tornam-se ativos quando algum aviso é necessário ou quando um agente eliminador é ativado;

- Agentes eliminadores: são agentes de resposta que terminam a execução de processos responsáveis por comportamento intrusivo na rede monitorada.

A Figura 3.7 ilustra esses agentes e o relacionamento existente entre eles. Nessa figura, agentes específicos ativam outros agentes ou trocam informações através de agentes de comunicação.

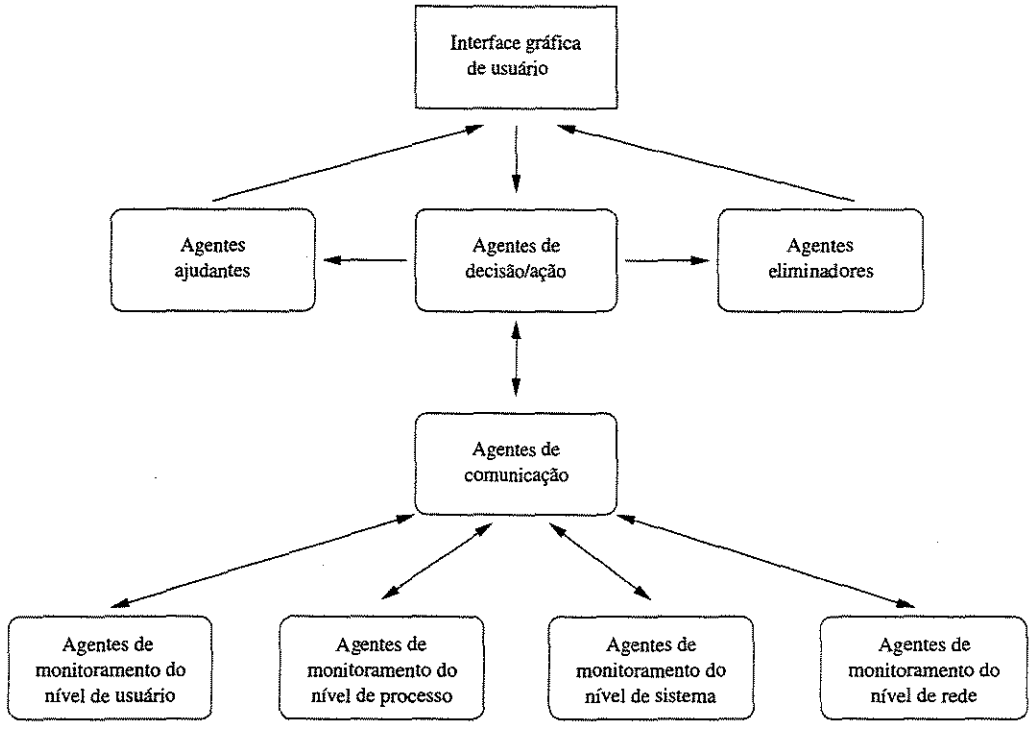

Figura 3.7: Conjunto de agentes propostos e interação entre eles.

Na identificação de comportamento é utilizado um sistema classificador baseado em redes neurais. Outra estratégia para a detecção baseada em agentes também foi proposta, utilizando um classificador genético [DG01]. Testes demonstraram a ocorrência de altas taxas de falso-positivos na identificação de ataques. É importante notar ainda que uma abordagem baseada em agentes deve prover mecanismos de auto-proteção, de modo que esses agentes não fiquem, por sua vez, suscetíveis a ataques [Uto03].

3.2.7 Abordagem imunogenética para detecção de intrusão

González e Dasgupta propuseram uma técnica para detecção baseada em comportamento inspirada pelo algoritmo de seleção negativa [GD02, DG02]. Enquanto o algoritmo de seleção negativa, na sua concepção original, é capaz de separar informações em duas classes, *self* e *nonself*, a técnica proposta é capaz de estender essa classificação para múltiplas classes. Especificamente, nessa pesquisa o espaço *nonself* é dividido em múltiplas subclasses de modo a determinar o nível de anormalidade observada. A idéia principal dessa técnica consiste em utilizar um algoritmo genético para evoluir um conjunto de regras que irão identificar o espaço *nonself*.

Esses pesquisadores observaram que, embora o sistema imunológico faça distinção entre *self* e *nonself*, em geral não existe uma distinção rígida entre estados normais e anormais, podendo ser identificados vários níveis de normalidade ou anormalidade. Apesar dessa definição de normalidade ou anormalidade parecer mais plausível, em um sistema computacional é necessário decidir sob quais situações um alarme deve ser disparado. Para tanto, é definido um limiar t de modo que todo evento com nível de normalidade menor ou igual a t é considerado *nonself* e os demais eventos são considerados *self*.

Para resolver o problema da identificação de ataques é empregado um algoritmo genético com um esquema de *niching* seqüencial para evoluir detectores no espaço *nonself*. Esses detectores correspondem a hiper-retângulos em um espaço multidimensional. Cada hiper-retângulo pode ser visto como um conjunto de condições existentes nas regras de detecção evoluídas. A função de avaliação ou *fitness* utilizada tem por objetivo maximizar o volume de cada hiper-retângulo, penalizando aqueles que cobrem amostras de comportamento usual observado durante a fase de treinamento. Desse modo busca-se encontrar um conjunto de regras evoluídas que generalizam a detecção de comportamento não usual. A Figura 3.8 ilustra como um conjunto de regras evoluídas pode ser utilizado para identificar *nonself* em um espaço de dimensão 2.

Durante os testes foram observados os seguintes parâmetros embutidos nas regras: número de bytes por segundo, número de pacotes por segundo e número de pacotes ICMP por segundo. Resultados experimentais com ataques no nível de rede demonstraram que essa técnica pôde identificar quatro dentre cinco ataques, apresentando uma taxa de falso-positivos de até 1%. Uma extensão desse trabalho também pode ser encontrada em [GGD03].

3.2.8 Um modelo imunológico para detecção de intrusão baseada em rede

Kim e Bentley propuseram revisar a analogia existente entre o sistema imunológico humano e sistemas de detecção de intrusão baseados em rede [KB99b]. Suas metas iniciais

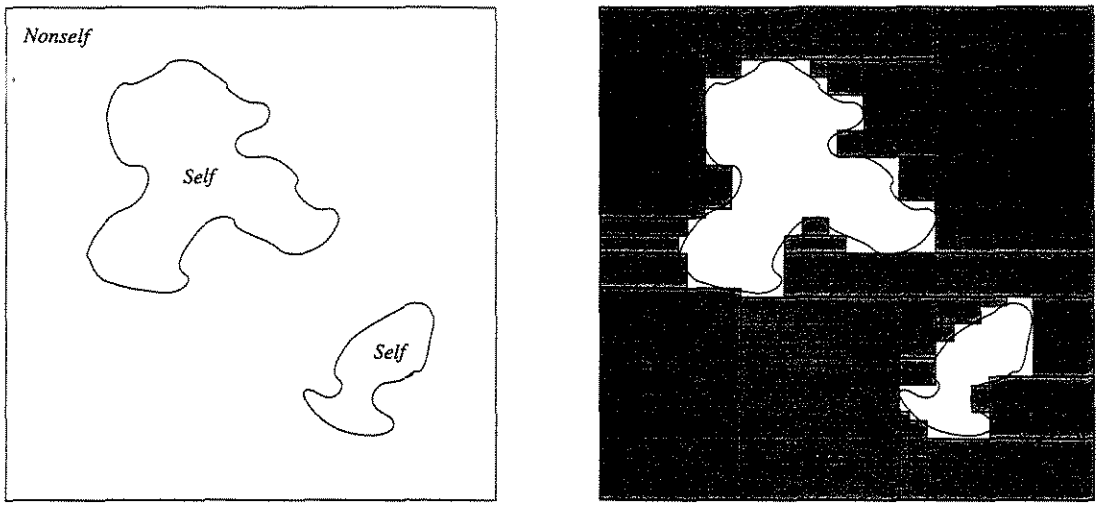

Figura 3.8: Utilização de um conjunto de regras evoluídas para identificar *nonself* em um espaço de dimensão 2. Na parte (a) da figura são apresentados as regiões *self* e *nonself* que se deseja distinguir. Na parte (b) da figura é apresentado o resultado final do algoritmo genético: um conjunto de regras evoluídas que cobrem o espaço *nonself*, onde cada regra é representada por um retângulo.

eram modelar e desenvolver um IDS distribuído, auto-organizável e com um baixo consumo de recursos. Alcançando essas metas, os autores sugeriram que tal IDS seria capaz de satisfazer os seguintes requisitos: robustez, configurabilidade, extensibilidade, escalabilidade, adaptabilidade, análise global e eficiência. Nesse trabalho, os autores identificam um método para a geração de detectores diversificados, baseado na construção de arranjos de genes a partir de uma biblioteca disponível. Posteriormente, esses arranjos são submetidos a processos de maturação e seleção negativa objetivando a construção de detectores eficientes. Detectores de memória são gerados através de uma seleção clonal, que prioriza aqueles detectores que já identificaram antígenos.

Dando continuidade a esse trabalho, esses pesquisadores apresentaram um modelo imunológico para detecção de intrusão baseada em rede [KB99a]. Esse modelo apresenta uma arquitetura física que consiste em um IDS primário e um conjunto de IDSs secundários, como ilustra a Figura 3.9.

Esse modelo especifica que o IDS primário, assim como a medula óssea ou o timo no sistema imunológico humano, funciona gerando numerosos conjuntos de detectores que serão distribuídos aos IDSs secundários. Conforme ilustrado, todo o tráfego que entra na rede monitorada passa pelos roteadores e também pelo IDS primário. Os detectores são

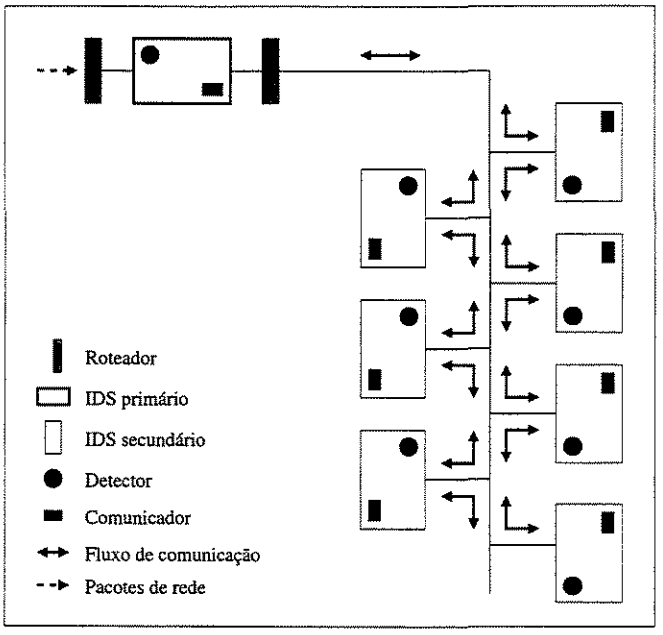

Figura 3.9: Arquitetura física do modelo imunológico proposto por Kim e Bentley. Assume-se que essa arquitetura é utilizada para monitorar um único domínio de rede.

responsáveis por identificar comportamento não usual no tráfego de rede observado. Cada máquina nessa rede é vista como um nódulo linfático secundário, os detectores são vistos como anticorpos e as intrusões na rede são vistas como antígenos. Os IDSs secundários, que são as próprias máquinas da rede, mantêm os detectores recebidos através da execução de processos em *background*, que visam a identificação de ataques destinados a essas máquinas.

O funcionamento do modelo imunológico é apresentado na Figura 3.10. Nessa figura, o Estágio 1 indica a evolução da biblioteca de genes, o Estágio 2 representa o processo de seleção negativa e o Estágio 3 mostra a seleção clonal.

O IDS primário realiza os dois primeiros estágios: a evolução da biblioteca de genes e a seleção negativa. No estágio de evolução da biblioteca de genes esse IDS obtém um conhecimento global de detectores efetivos. No estágio de seleção negativa, almeja-se gerar uma diversidade de detectores que não identificam *self*, transferindo conjuntos únicos de detectores para as demais máquinas. Para tanto, inicialmente uma biblioteca de genes é gerada e mantida através de um processo evolutivo. Cada gene expressa um atributo utilizado em perfis para descrever comportamento não usual. Esses atributos são escolhidos de maneira a contemplar os problemas de segurança existentes. No segundo

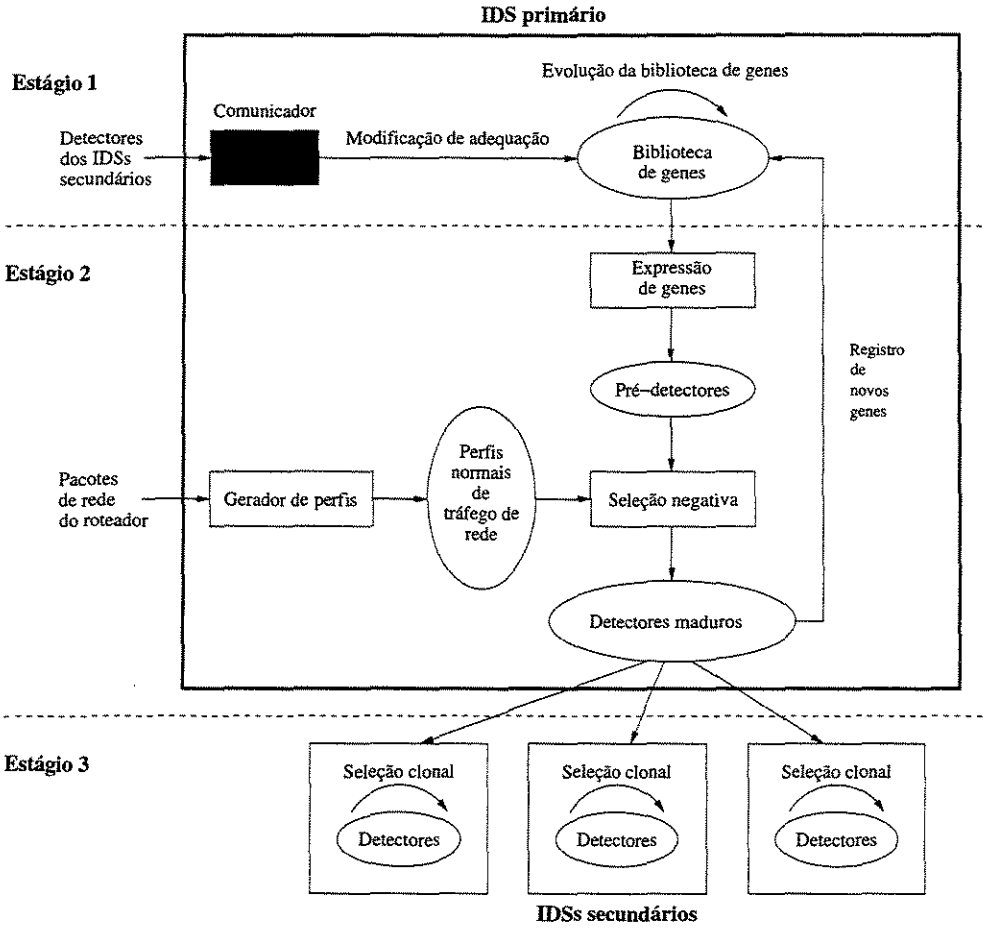

Figura 3.10: Funcionamento do modelo imunológico em três estágios: evolução da biblioteca de genes, processo de seleção negativa e processo de seleção clonal.

estágio o processo de expressão de genes gera vários pré-detectores através de um arranjo de genes selecionados da biblioteca. Esses pré-detectores são expostos ao processo de seleção negativa e, aqueles que não são descartados nesse processo, tornam-se detectores maduros. Esses detectores maduros são utilizados para realimentar a biblioteca de genes e são entregues aos IDSs secundários, distribuídos na forma de conjuntos únicos.

Cada IDS secundário realiza o último estágio evolutivo: a seleção clonal. A principal tarefa desses IDSs é identificar uma variedade de ataques com um número limitado de detectores, criando clones dos detectores que efetuaram um papel importante na detecção. Esses detectores privilegiados também são utilizados para realimentar a biblioteca de genes do IDS primário.

Os autores frisam que esse modelo utiliza vários mecanismos do sistema imunológico, ao contrário dos trabalhos anteriores que tinham como base somente a seleção negativa, mostrando ser uma abordagem bastante promissora.

3.2.9 Uma discussão do papel da seleção negativa na detecção

Uma vez concluída a modelagem, Kim e Bentley investigaram algumas técnicas para possibilitar a implementação de um sistema de detecção de intrusão baseado na abordagem apresentada na Seção 3.2.8 [KB99c, KB01b, KB01a, KB02]. Foram realizados alguns testes com informações relevantes para avaliar o sistema de detecção de intrusão, enquanto outros testes foram feitos considerando informações para avaliar sistemas evolutivos em geral. Resultados experimentais indicam que o sistema de detecção de intrusão não produziu um efeito satisfatório, sendo também levantada uma questão polêmica a respeito do papel da seleção negativa na detecção de intrusão baseada em rede.

Kim e Bentley discutem o papel do algoritmo de seleção negativa em [KB01a]. Esse algoritmo é bastante popular no desenvolvimento de IDSs baseados no sistema imunológico, sendo largamente utilizado por grupos de pesquisa expressivos, como o de Stephanie Forrest e Dipankar Dasgupta.

O algoritmo da seleção negativa modela o espaço *nonself* utilizando um conjunto de detectores que são descartados quando, em fase de aprendizado, identificam *self*. Após esse aprendizado, a identificação realizada por esses detectores é um indício de comportamento não usual, geralmente relacionado a um ataque. Entretanto, para que esse algoritmo seja eficiente, deve ser utilizado um conjunto limitado de detectores que seja bastante representativo. Sendo assim, o espaço *nonself* deve ser representado de uma forma bastante precisa por esse conjunto de detectores, de forma que falso-positivos e falso-negativos possam ocorrer somente em taxas aceitáveis.

De acordo com os resultados analisados, Kim e Bentley questionaram a eficiência desse algoritmo e propuseram que um melhor uso da seleção negativa pode ser alcançado na filtragem de detectores inválidos, e não na geração de detectores efetivos [KB01a]. Em outras palavras, sob a visão desses pesquisadores, o algoritmo de seleção negativa é bastante eficiente quando utilizado para descartar a ocorrência de falso-positivos. Entretanto, a geração aleatória de detectores pode apresentar resultados aquém do esperado, ocasionando falso-negativos.

Mais informações a respeito do trabalho desenvolvido por Kim e Bentley também podem ser encontradas na tese de doutorado de Jungwon Kim [Kim02].

3.2.10 Teoria do Perigo e detecção de intrusão

Na década passada uma nova teoria tornou-se popular entre imunologistas. Essa teoria é chamada Teoria do Perigo (Danger Theory, em inglês), tendo como defensora Polly Matzinger [Mat96, AC02]. Essa teoria controversa desafia a clássica idéia que se tem a respeito do funcionamento do sistema imunológico — a distinção entre *self* e *nonself* — e propõe que a ocorrência de danos induzidos no corpo é essencial para desencadear uma resposta imunológica.

A idéia central da Teoria do Perigo é que o sistema imunológico não responde a *nonself*, mas a evidências de perigo [AC02]. Da mesma forma que as teorias de distinção entre *self* e *nonself*, essa teoria também inclui uma forma de distinção. Entretanto, ela propõe que, ao invés de reagir contra substâncias estranhas ao corpo, o sistema imunológico reage contra fontes de perigo.

A Teoria do Perigo é baseada na observação de que não há necessidade de responder a todo tipo de substância estranha, como ocorre com as bactérias encontradas no intestino e o alimento ingerido, que também é composto por substâncias estranhas ao corpo. O perigo é medido através de danos ocasionados às células, sendo observado por meio de sinais emitidos quando essas células morrem de um modo não natural.

A Figura 3.11 mostra como ocorre a identificação de antígenos e a resposta imunológica considerando essa teoria. Como pode-se notar, as células imunológicas só irão produzir uma resposta contra um antígeno quando ele for identificado e essa identificação ocorrer dentro de uma zona de perigo. Essa zona de perigo é demarcada pelos sinais de perigo produzidos pelas células danificadas.

Quando essa teoria é considerada deve-se tomar o cuidado de avaliar a ocorrência de substâncias *nonself* mas inofensivas, bem como substâncias *self* mas danosas.

Aickelin e Cayzer estudaram a Teoria do Perigo e, sem defendê-la, procuraram verificar que contribuições ela poderia trazer para o desenvolvimento de sistemas imunológicos artificiais [AC02]. As principais considerações práticas levantadas por esses pesquisadores são:

- Esse modelo requer células de apresentação de antígenos que possam apresentar um determinado sinal de perigo;
- O termo "perigo" é relativo e, em algumas aplicações, pode não estar relacionado com algum perigo real;
- O sinal de perigo pode ser modelado positivamente, na presença de sinal, ou negativamente, na ausência de sinal;
- A zona de perigo biológica é espacial. Em sistemas imunológicos artificiais pode ser utilizada outra medida de proximidade como, por exemplo, uma medida temporal;

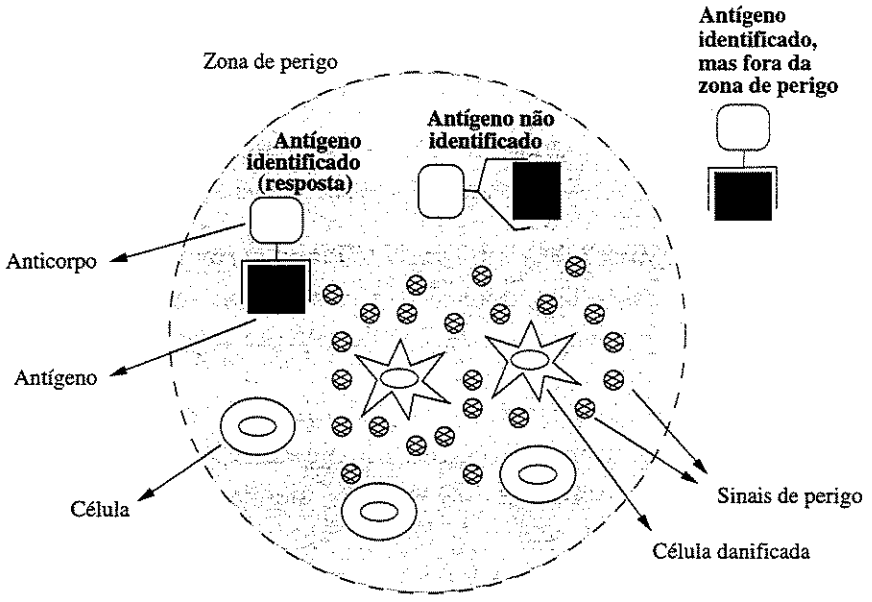

Figura 3.11: Identificação de antígenos e resposta imunológica considerando a Teoria do Perigo.

- Uma resposta imunológica não deve causar novos sinais de perigo. De fato, no sistema biológico, as células imunológicas eliminadoras causam mortes naturais de células e não sinais de perigo.

Aickelin e Cayzer também discutem que sistemas de detecção de intrusão podem incluir alguns sinais de perigo, tais como baixo ou alto uso de memória, atividade de disco não usual, freqüência inesperada na alteração de arquivos, geração do sinal SIGABRT em processos Unix, e alguma presença de *nonself*.

Baseados nas características dessa teoria, outros pesquisadores da área de sistemas imunológicos artificiais iniciaram novos trabalhos em busca de melhores resultados para o problema da detecção de intrusão [ABC+03, Twy04].

3.2.11 Um sistema imunológico para a identificação de vírus

Kephart, Chess e White efetuaram, em 1993, um estudo relacionando vírus biológicos com vírus de computador [KCW93]. O enfoque desse trabalho era apresentar e discutir fatores que influenciam a disseminação desses vírus. Em 1994 Kephart propôs um sistema para identificação de vírus baseado em características do sistema imunológico biológico [Kep94]. Esse sistema automatizava o processo de análise, possibilitando extrair assinaturas de

detecção de vírus. Tal sistema foi concebido para ser incorporado gradualmente no sistema anti-vírus produzido pela IBM.

Kephart estava motivado para essa pesquisa devido a duas razões:

- Vírus estavam sendo desenvolvidos rapidamente e em ritmo cada vez mais acelerado, impossibilitando uma análise manual através de especialistas na área;
- A crescente interconectividade e interoperabilidade entre computadores estava contribuindo para uma rápida disseminação de vírus. Sendo assim, a estratégia de periodicamente distribuir novas atualizações do *software* anti-vírus poderia se tornar impraticável.

Kephart notou que o sistema imunológico traria algumas propriedades excelentes para um sistema de identificação de vírus descentralizado, incluindo:

1. Reconhecimento de vírus conhecidos;
2. Eliminação e neutralização;
3. Capacidade de analisar vírus previamente desconhecidos;
   (a) Determinar a ocorrência de vírus;
   (b) Aprender a reconhecê-lo;
   (c) Memorizar esse reconhecimento.
4. Uso de uma proliferação seletiva e auto-replicação para um reconhecimento rápido e resposta.

A Figura 3.12 mostra o esquema do sistema proposto, que foi desenvolvido considerando as propriedades citadas.

A identificação do vírus é iniciada através de um processo de detecção baseado em comportamento. Em seguida o sistema é verificado em busca de um vírus conhecido. Caso seja conhecido, ele é eliminado e sinais de alerta podem ser enviados a máquinas vizinhas. Se não é conhecido, o sistema colhe um conjunto de *decoys*[5] para análise, os quais em geral estarão infectados. Nessa análise o sistema compara os *decoys* originais com os infectados e extrai as diferenças encontradas. O analisador algorítmico também extrai informações úteis para a remoção do vírus. O extrator de assinatura recebe as informações relevantes para a assinatura e compara essas informações com um conjunto diversificado de arquivos usuais. Dessa maneira, busca-se gerar assinaturas que causem

[5] Arquivos ou programas que têm somente a função de atrair vírus, como em uma espécie de armadilha.

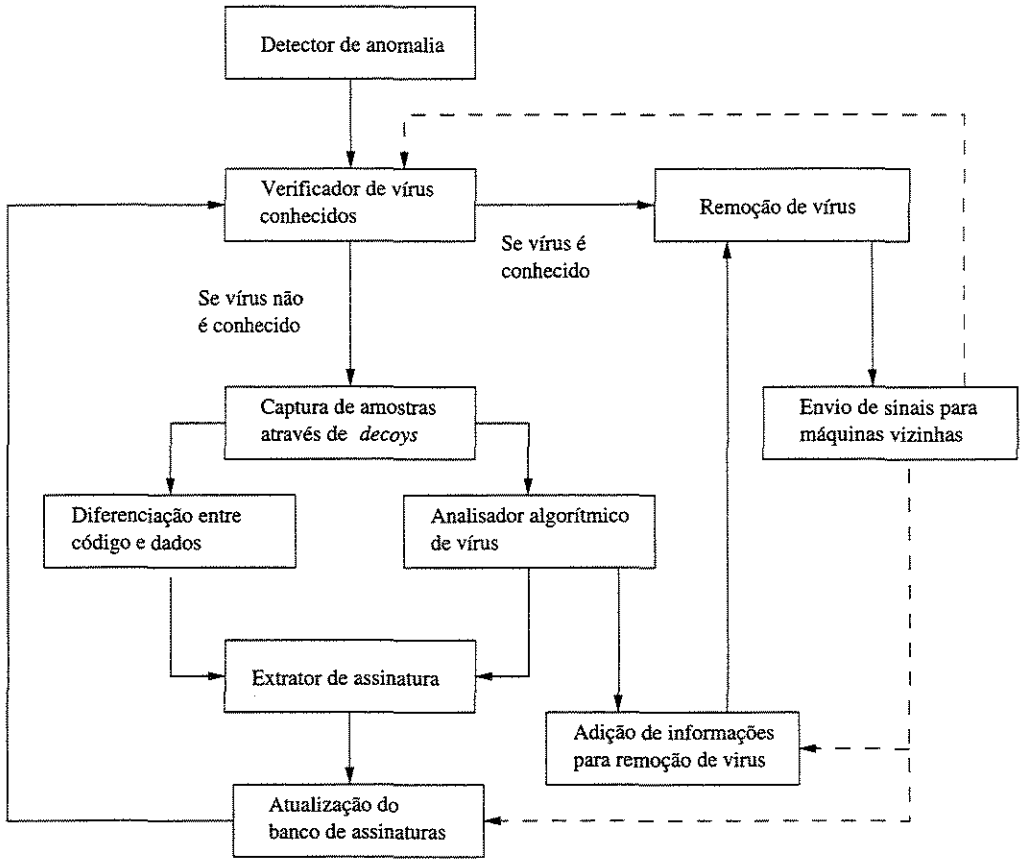

Figura 3.12: Sistema anti-vírus baseado no sistema imunológico proposto por Kephart. O envio de sinais de alerta para máquinas vizinhas é representado pelas linhas direcionadas pontilhadas.

uma baixa taxa de falso-positivos. Ao final, a base de assinaturas é atualizada e são tomadas as ações cabíveis para remover o novo vírus.

Resultados experimentais com um protótipo indicam que esse sistema foi capaz de prover alguma informação útil para 90% dos vírus analisados. O extrator de assinatura também pode ser utilizado para avaliar assinaturas concebidas por especialistas na área.

Embora outros artigos também tenham sido publicados [KA94, KSSW97], não foram apresentados novos resultados experimentais a respeito desse sistema.

3.3 Conclusão

Este capítulo apresentou a estrutura e o funcionamento básico do sistema imunológico biológico, sem entrar em detalhes minuciosos mas provendo um embasamento para discutir pesquisas relevantes para este trabalho. Em seguida foram mostrados os principais princípios do sistema imunológico, oferecendo exemplos de aplicações em segurança computacional que podem ser desenvolvidas observando esses princípios.

Observando os princípios e funcionamento do sistema imunológico, diversos pesquisadores atentaram para a área de segurança computacional, desenvolvendo sistemas de detecção e resposta à intrusão. Os principais trabalhos desenvolvidos, apresentados neste capítulo, são sintetizados na Tabela 3.2.

Trabalho	**Contribuições para a detecção de intrusão**
Distinção entre *self* e *nonself*	O algoritmo adaptativo de seleção negativa para detecção de alteração em objetos monitorados
Significado para *self* em Unix	Detecção de intrusão através da análise de seqüências de chamadas ao sistema
Diversidade de sistemas	Métodos para dificultar a disseminação de ataques
ARTIS e LISYS	Detecção de intrusão através da seleção negativa e um IDS distribuído baseado em rede
Homeostase de processos e resposta automática	Detecção baseada em chamadas ao sistema e resposta adaptativa automatizadas
Detecção baseada em agentes	Modularização e distribuição na detecção/resposta
Abordagem imunogenética	Evolução de regras para a identificação de ataques
Papel da seleção negativa	Questionar o uso da seleção negativa na detecção
Teoria do Perigo	Uma oportunidade de melhorar a detecção
Sistema imunológico para identificação de vírus	Pode inspirar a detecção baseada em *decoys* ou *honeypots* e a extração de assinaturas de ataque

Tabela 3.2: Principais trabalhos e contribuições envolvendo sistemas de segurança baseados no sistema imunológico.

É importante notar que as aplicações desenvolvidas durante esses trabalhos consideram a detecção baseada em comportamento. Inclusive o sistema de identificação de vírus proposto por Kephart tem seu funcionamento dependente de uma boa detecção baseada em comportamento. Na prática, nota-se que a detecção baseada em comportamento

incorre em altas taxas de falso-positivos, sendo comercialmente aplicada em poucos casos.

Por outro lado, a Teoria do Perigo pode trazer grandes benefícios à detecção de intrusão através da análise de evidências de danos no sistema. Se essa análise de evidências for combinada com um sistema para a extração de assinaturas de ataque, pode-se obter um modelo de segurança bastante desejável a sistemas computacionais. Com a utilização das assinaturas extraídas é possível identificar novas instâncias de um ataque no futuro e combatê-las de uma forma eficiente.

Parte II

Modelagem, implementação e testes

Capítulo 4

Uma arquitetura de segurança inspirada no sistema imunológico

Como pode ser notado, o sistema imunológico possui uma analogia bastante clara com a segurança de sistemas computacionais e serve como base para diversas pesquisas nessa área. Essas pesquisas estão concentradas principalmente no desenvolvimento de sistemas de detecção de intrusão e, em geral, incorporam somente a detecção baseada em comportamento. Entretanto, esse sistema biológico oferece um vasto campo de atuação, podendo ser utilizado para inspirar uma arquitetura de segurança mais ampla.

Este capítulo descreve uma arquitetura de segurança inspirada no sistema imunológico desenvolvida neste trabalho. Essa arquitetura utiliza as características e os princípios do sistema imunológico para modelar um sistema de segurança capaz de analisar intrusões, extrair assinaturas de ataque e prover mecanismos de resposta para amenizar a intrusão e bloquear futuras exposições a um mesmo ataque. Sendo assim, essa arquitetura apresenta características que vão além de um sistema de detecção de intrusão tradicional, provendo mecanismos para manipular ataques conhecidos e estudar ataques desconhecidos de uma forma automatizada.

A Seção 4.1 descreve o modelo de ameaça considerado e a Seção 4.2 descreve a arquitetura proposta identificando seus componentes, os quais são detalhados na Seção 4.3. A Seção 4.4 apresenta o funcionamento geral dessa arquitetura, enquanto a Seção 4.5 discute as analogias existentes com o sistema biológico. A Seção 4.6 mostra alternativas práticas para a implementação de um sistema de segurança. Ao final, a Seção 4.7 aponta os trabalhos relacionados com esta pesquisa.

4.1 Modelo de ameaça considerado

Antes de projetar uma arquitetura ou sistema de segurança é necessário compreender o ambiente que se deseja proteger e quais são as ameaças consideradas. Neste trabalho o ambiente a ser protegido compreende um conjunto de computadores interligados através de redes e com um possível acesso à rede global Internet.

A quantidade de ameaças que podem rondar esse tipo de ambiente é bastante grande. Exemplos incluem ameaças ambientais como fogo, terremoto, explosões e inundações, falhas estruturais, perda de pessoal, falta de eletricidade, falha na rede, roubo de equipamentos, introdução de vírus, falhas em *software* e intrusões eletrônicas [SG96]. Este trabalho considera o modelo de ameaça como apresentado na Figura 4.1.

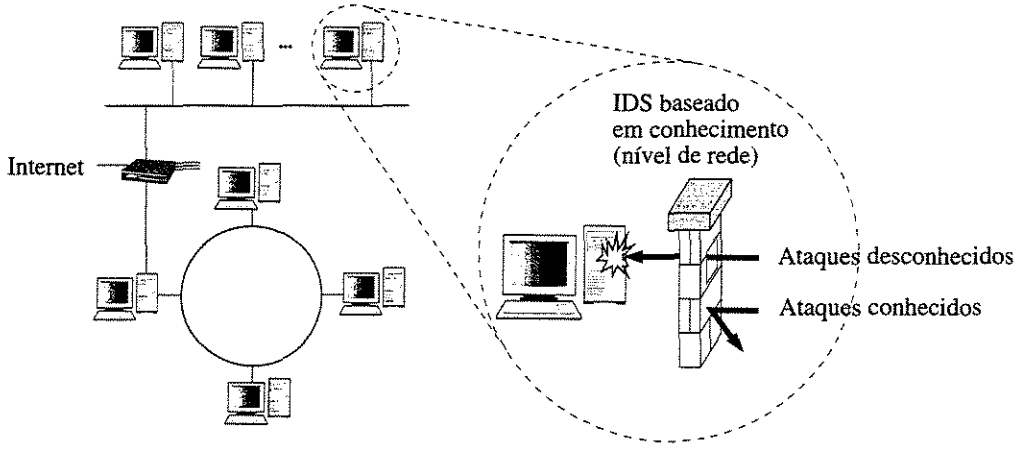

Figura 4.1: Modelo de ameaça considerado.

Nesse ambiente cada computador está disponível para algum tipo de acesso através da rede. Usuários maliciosos, internos ou externos a esse ambiente, podem utilizar a rede de comunicação para lançar ataques contra esses computadores. Um ataque conhecido pode ser identificado e bloqueado por um IDS baseado em conhecimento que monitora o fluxo de rede em busca de assinaturas de ataque. Porém, ataques desconhecidos podem passar despercebidos e são concluídos com sucesso, perturbando de alguma forma a máquina atacada.

Utilizando esse modelo, este trabalho propõe uma arquitetura de segurança que possa lidar com ataques conhecidos e desconhecidos. Quando um ataque desconhecido obtém sucesso, ele é estudado de maneira a extrair uma assinatura automaticamente, tornando esse ataque conhecido.

Embora o modelo de ameaça apresentado considera apenas ataques através da rede,

a mesma idéia pode ser aproveitada em outros contextos. Por exemplo, se as assinaturas fossem especificadas no nível do fluxo da informação de aplicações, então cada assinatura poderia levar em conta as chamadas ao sistema, os parâmetros passados a processos, as entradas de usuários locais e a comunicação entre processos.

4.2 Arquitetura de segurança

Esta seção introduz a arquitetura de segurança proposta, a qual pode suportar várias características desejáveis em um sistema computacional. Essa arquitetura tem a intenção de absorver, em especial no nível de modelagem, várias características do sistema imunológico, algumas delas pouco exploradas em outros trabalhos. Exemplos dessas características são: tolerância a intrusão, detecção baseada em evidências de intrusão e automatização da extração de assinaturas de ataque e restauração do sistema.

Um dos aspectos mais importantes dessa arquitetura consiste em assumir que ataques com sucesso são inevitáveis. Sua característica mais marcante reside na habilidade de tratar tal situação. De fato, isso também ocorre no sistema imunológico humano. Alguns agentes causadores de doenças invadem o organismo e causam danos nele antes que o sistema imunológico consiga eliminá-los. Entretanto, após essa invasão, o sistema imunológico aprende a lidar com esse tipo de agente e alguma estratégia de restauração é utilizada para recuperar as partes danificadas.

A arquitetura em questão foi elaborada através de uma evolução de especificações [dPdCdG04, dPdRFdG02, dRdPFdG02, dRFdPdG01] e tem os seguintes objetivos principais:

1. Detecção precisa de ataques conhecidos e uma resposta efetiva para bloqueá-los;
2. Detecção de ataques previamente desconhecidos através da análise de intrusões no sistema;
3. Habilidade de lidar com ataques previamente desconhecidos, de modo a proporcionar:
   (a) Provisão de medidas de contenção para manter o sistema em condições aceitáveis enquanto uma análise mais detalhada é feita;
   (b) Aprendizado a respeito da intrusão, na tentativa de extrair uma assinatura para a detecção do ataque efetuado;
   (c) Armazenamento de informação relevante ao ataque em um repositório de forma a possibilitar uma futura análise forense;
   (d) Restauração das partes do sistema afetadas pela intrusão.

4. Detecção precisa e resposta contra ataques que o sistema aprendeu automaticamente a reconhecer.

Para alcançar essas metas, essa arquitetura utiliza dez componentes — cada qual com uma função particular — e especifica o relacionamento existente entre eles. A Figura 4.2 ilustra esses componentes e o fluxo de comunicação existente. Nessa figura, os componentes são representados por retângulos de linha sólida. Um agrupamento de componentes é representado por um retângulo de linha tracejada. As linhas direcionadas sólidas indicam o fluxo da informação e as linhas direcionadas tracejadas mostram o fluxo de controle correspondente à ativação de componentes. Cada fluxo de comunicação ocorre entre dois componentes ou entre o Console e todos os demais componentes dentro do agrupamento. Cada componente é detalhado na Seção 4.3.

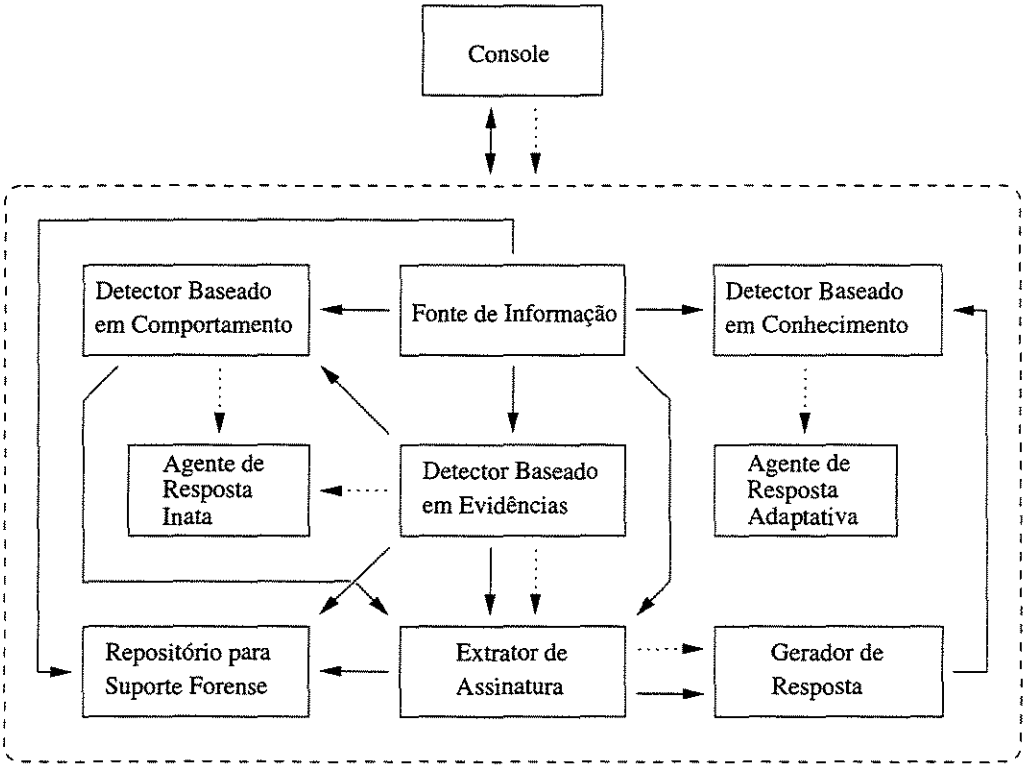

Figura 4.2: Arquitetura de segurança inspirada no sistema imunológico humano.

Em linhas gerais, a manipulação de ataques conhecidos é realizada através da detecção baseada em conhecimento associada a uma resposta específica, sendo representada pelos componentes Detector Baseado em Conhecimento e Agente de Resposta Adaptativa. A

identificação de ataques previamente desconhecidos é feita através da detecção baseada em evidências de intrusão efetuada pelo componente Detector Baseado em Evidências. A habilidade de lidar com ataques previamente desconhecidos é proporcionada pelos componentes Agente de Resposta Inata, Detector Baseado em Comportamento, Extrator de Assinatura, Gerador de Resposta e Repositório para Suporte Forense.

4.3 Componentes da arquitetura

Alguns componentes da arquitetura proposta compreendem mecanismos clássicos da detecção de intrusão, tais como o Console, a Fonte de Informação, o Detector Baseado em Conhecimento e o Detector Baseado em Comportamento. Os demais componentes são especificados pela arquitetura de forma a proporcionar os objetivos já mencionados. A descrição de cada um desses componentes é apresentada como segue.

4.3.1 Console

Alguns componentes, como o Detector Baseado em Comportamento, têm seu funcionamento baseado na análise de um conjunto de parâmetros pré-definidos. Dependendo do ambiente onde o sistema de segurança atua, é necessário redefinir alguns desses parâmetros para adequar tal sistema a esse ambiente.

O Console consiste em uma interface entre o sistema de segurança e o administrador do sistèma. É através dessa interface que o administrador pode modificar parâmetros de detecção, incluir, remover ou alterar assinaturas de ataque, habilitar ou desabilitar componentes e visualizar resultados e alertas do sistema de segurança.

Essa interface pode ser implementada através de um conjunto de arquivos de configuração e arquivos de *log*, podendo oferecer ainda uma interface gráfica de usuário e algum mecanismo de acesso remoto.

4.3.2 Fonte de Informação

A Fonte de Informação é responsável pela coleta e armazenamento temporário de dados relevantes ao funcionamento dos componentes que estão engajados em algum tipo de análise, armazenamento ou visualização de informações. Esse componente pode ser implementado separadamente ou pode ser constituído por um grupo de módulos embutidos nos componentes requisitantes dos dados. Em algumas situações críticas, como na captura de chamadas ao sistema, a utilização de módulos embutidos nos componentes requisitantes pode resultar em uma grande vantagem no desempenho final do sistema de segurança.

A Tabela 4.1 mostra, em linhas gerais, o tipo de monitoramento necessário para o funcionamento de cada componente que recebe um fluxo de dados da Fonte de Informação. Como a arquitetura proposta utiliza tanto o monitoramento baseado em máquina ou aplicação — devido à detecção baseada em evidências — quanto o monitoramento baseado em rede — devido à usual especificação de assinaturas de ataque nesse nível — a Fonte de Informação deve ser capaz de atuar em pelo menos dois níveis. Os componentes Console, Detector Baseado em Conhecimento e Repositório para Suporte Forense irão atuar sobre informações que sejam relevantes para uma implementação específica e podem, nesse caso, considerar um novo tipo de monitoramento.

Componente	**Tipo de monitoramento**
Console	Dependente da implementação
Detector Baseado em Conhecimento	Baseado em rede
Detector Baseado em Comportamento	Dependente da implementação
Detector Baseado em Evidências	Baseado em máquina ou aplicação
Extrator de Assinatura	Baseado em rede
Repositório para Suporte Forense	Dependente da implementação

Tabela 4.1: Tipo de monitoramento devido a cada componente requisitante.

4.3.3 Detector Baseado em Conhecimento

Assim como na detecção de intrusão clássica, o Detector Baseado em Conhecimento é responsável por identificar ataques através da análise da ocorrência de assinaturas de ataque no fluxo de informação monitorado. A Figura 4.3 mostra um detalhamento desse componente.

Essencialmente, esse componente analisa o fluxo de eventos no nível de rede proveniente da Fonte de Informação. Quando determinados padrões de eventos são reconhecidos por alguma assinatura de ataque armazenada no Banco de Dados de Assinaturas e Respostas, ocorre a identificação de um ataque. Uma vez identificado um ataque, o Detector Baseado em Conhecimento habilita uma resposta específica contra esse ataque por meio da ativação do componente Agente de Resposta Adaptativa.

O Banco de Dados de Assinaturas e Respostas deve ser capaz de manipular tuplas que representam seqüências de eventos que constituem as assinaturas e o tipo de resposta que está associado a cada assinatura. É permitido ao Gerador de Resposta incluir novas tuplas nessa base de dados. Através do Console também é possível acessar essa base de

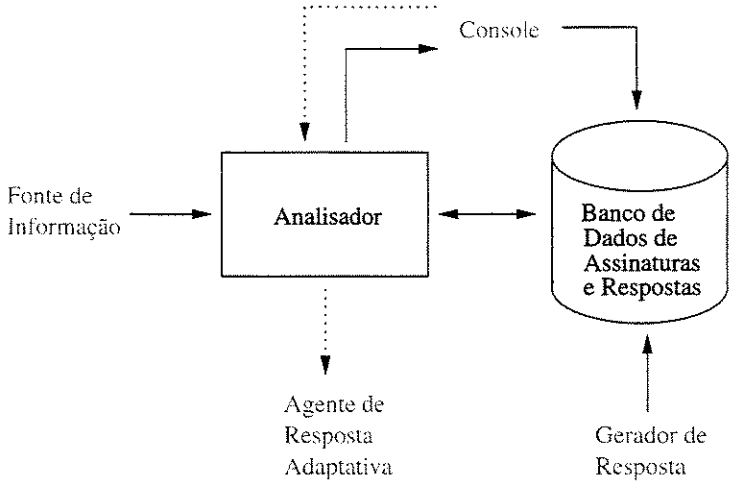

Figura 4.3: Detalhamento do Detector Baseado em Conhecimento.

dados, obter resultados e até mesmo desabilitar/habilitar a análise do Detector Baseado em Conhecimento.

4.3.4 Agente de Resposta Adaptativa

O Agente de Resposta Adaptativa é responsável por realizar um conjunto de medidas para bloquear um ataque identificado. Essas medidas estão identificadas no Banco de Dados de Assinaturas e Respostas, cabendo a esse agente a tarefa de executar ações pré-especificadas. Essas respostas devem ser suficientemente precisas de modo a impedir a ocorrência de danos no sistema monitorado.

A Tabela 4.2 exemplifica como um tipo de resposta pode ser representado na base de dados através de um número identificador. Cada tipo de resposta tem um significado prático e está associado a um conjunto de ações ou medidas a serem executadas pelo Agente de Resposta Adaptativa.

Quando o Detector Baseado em Conhecimento identifica um ataque, ele repassa o identificador de resposta associado à assinatura em questão para o Agente de Resposta Adaptativa. Esse agente, que conhece o significado da resposta requerida, executa as medidas necessárias para bloquear o ataque.

Tipo de resposta	Significado	Ação executada pelo Agente de Resposta Adaptativa
0	bloquear certo tráfego de rede	`iptables ... -j DROP`
1	desabilitar interface de rede	`ifconfig iface down`
2	terminar um processo em execução	`kill -9 pid`
3	descarregar módulo de *kernel* e apagar binário	`rmmod mod ; rm mod.o`

Tabela 4.2: Exemplos de tipo de resposta, significado de resposta e ação executada pelo Agente de Resposta Adaptativa, considerando o sistema GNU/Linux.

4.3.5 Detector Baseado em Evidências

O Detector Baseado em Evidências é um componente vital na arquitetura proposta. É através desse componente que o sistema de segurança identifica precisamente a ocorrência de ataques desconhecidos e inicia os processos de restauração e extração de assinatura. Ele funciona de acordo com a definição da detecção baseada em evidências de intrusão apresentada na Seção 2.3.3 e é detalhado na ilustração da Figura 4.4.

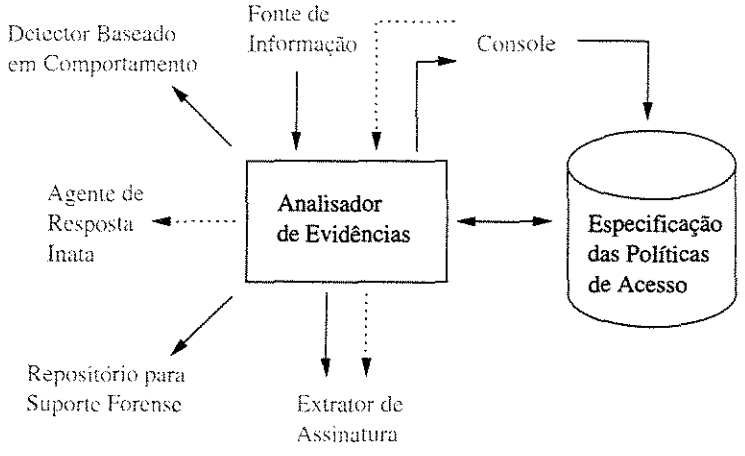

Figura 4.4: Detalhamento do Detector Baseado em Evidências.

Esse componente analisa o fluxo de eventos no nível de máquina ou aplicação proveniente da Fonte de Informação. Uma intrusão é identificada quando alguma política constante na base de dados Especificação das Políticas de Acesso é violada. Uma vez

identificada uma intrusão, o Agente de Resposta Inata e o Extrator de Assinatura são ativados. Nesse caso, o Extrator de Assinatura e o Repositório para Suporte Forense também recebem informações adicionais a respeito da violação ocorrida, como por exemplo, a aplicação, o usuário e as portas de comunicação envolvidas. Por outro lado, o Detector Baseado em Evidências fornece constantemente ao Extrator de Assinaturas e ao Detector Baseado em Comportamento seu estado corrente: "ataque não identificado" ou "ataque identificado". Esse estado pode ser consultado de modo a possibilitar um melhor funcionamento do sistema de segurança. Através do Console é possível acessar a base Especificação das Políticas de Acesso, obter resultados e desabilitar/habilitar a análise de evidências de intrusão.

Trabalhando com eventos no nível de máquina ou aplicação, o Analisador de Evidências pode verificar uma vasta gama de ações no sistema monitorado. Exemplos de eventos bastante úteis no monitoramento de evidências em aplicações para um sistema GNU/Linux podem incluir:

- Operações de alteração nas hierarquias de diretórios `/bin`, `/boot`, `/dev`, `/etc`, `/initrd`, `/lib`, `/sbin`, `/usr` e `/var/named`;
- Operações de leitura em arquivos restritos como `/etc/shadow` e arquivos confidenciais;
- Envio e recebimento de sinais/informações entre processos locais;
- Estabelecimento ou aceitação de comunicação remota imprevista;
- Execução de novos processos a partir do processo corrente;
- Inserção ou remoção de módulos de *kernel*.

Esses eventos tornam-se mais interessantes ainda quando eles são correlacionados com a aplicação executante. Por exemplo, os arquivos `/etc/passwd` e `/etc/shadow` podem ser alterados através da aplicação `useradd`. Porém, a aplicação `named` não deve alterá-los e tampouco invocar a aplicação `useradd` para tal ação. É interessante notar que o Detector Baseado em Evidências pode limitar o monitoramento a aplicações mais suscetíveis a ataques como *daemons* e aplicações que provêem acesso público.

Embora esse componente identifique um ataque somente após o sucesso de algumas ações maliciosas, a arquitetura proposta está preparada para lidar com essa situação e restaurar as partes danificadas.

4.3.6 Detector Baseado em Comportamento

Esse componente atua do mesmo modo que na detecção baseada em comportamento clássica. Quando o Detector Baseado em Comportamento identifica um comportamento não usual excedendo um limiar pré-especificado ocorre uma identificação de um provável ataque. A Figura 4.5 ilustra o detalhamento desse componente.

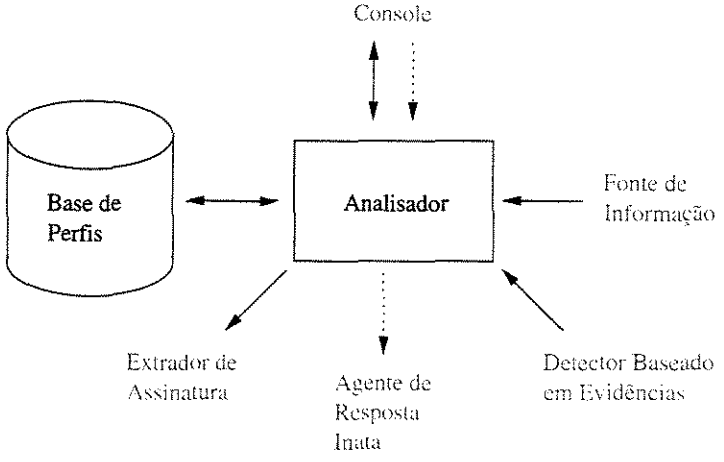

Figura 4.5: Detalhamento do Detector Baseado em Comportamento.

Para identificar prováveis ataques, o Detector Baseado em Comportamento analisa eventos provenientes da Fonte de Informação de acordo com o modelo de comportamento usual armazenado na Base de Perfis. Quando ocorre uma identificação, o componente Agente de Resposta Inata é ativado. Além disso, os eventos não usuais observados em um período de tempo — por exemplo, nas últimas 24 horas — ficam disponíveis para consulta ao Extrator de Assinatura. O Detector Baseado em Comportamento também pode aprimorar sua detecção através de consultas ao estado corrente do Detector Baseado em Evidências. Utilizando o Console é possível acessar parâmetros de análise como, por exemplo, limiares, obter resultados e desabilitar/habilitar o Analisador.

A detecção baseada em comportamento é bastante importante porque:

- Ela pode preceder a detecção de evidências de intrusão. Sendo assim, possibilita ativar o Agente de Resposta Inata para minimizar danos no sistema e reservar recursos computacionais enquanto uma análise mais detalhada é conduzida;

- Ela é capaz de detectar eventos não usuais relacionados a ataques desconhecidos. Dessa maneira, pode ser utilizada para levantar um conjunto de assinaturas de ataque candidatas após a identificação de evidências de intrusão. Nesse caso é

necessário monitorar eventos no nível de rede, uma vez que almeja-se especificar as assinaturas de ataque nesse nível.

É necessário entender que a arquitetura proposta utiliza as virtudes da detecção baseada em comportamento sem, entretanto, delegar a essa detecção a responsabilidade da identificação precisa de ataques. Sendo assim, o Detector Baseado em Comportamento pode conviver com a ocorrência de falso-positivos sem afetar consideravelmente o sistema monitorado. Essa flexibilidade também é garantida pela aplicação de respostas suaves através do Agente de Resposta Inata descrito na Seção 4.3.7.

Utilizando o estado do sistema proveniente do Detector Baseado em Evidências, o Detector Baseado em Comportamento pode ajustar seu modelo de comportamento usual e melhorar a detecção em dois casos:

1. Quando o comportamento observado aparenta ser não usual e a detecção baseada em evidências indica "ataque não identificado". Nesse caso, o comportamento observado pode ser incorporado no perfil normal;

2. Quando o comportamento observado aparenta ser usual mas a detecção baseada em evidências indica "ataque identificado". Nesse caso, o comportamento observado pode ser removido do perfil normal.

Em ambos os casos é possível reduzir futuras identificações errôneas, as quais levam a falso-positivos e falso-negativos. Porém, é necessário observar que a detecção baseada em comportamento pode preceder a detecção baseada em evidências e, portanto, o estabelecimento de algum critério de sincronização torna-se essencial.

4.3.7 Agente de Resposta Inata

O Agente de Resposta Inata é responsável por iniciar uma série de medidas de contenção visando dificultar um provável ataque, reservar recursos computacionais ou restaurar partes do sistema afetadas em uma intrusão. A sua resposta não é específica e funciona de um modo semelhante para diferentes tipos de ataque.

Esse agente, que pode ser ativado pelo Detector Baseado em Comportamento ou pelo Detector Baseado em Evidências, possui duas classes de resposta:

1. Medidas de contenção suaves e reversíveis: compreendem ações para dificultar um provável ataque e reservar recursos computacionais. Exemplos dessas ações incluem limitar a largura de banda ou acesso a disco, alterar a prioridade de processos, limitar a execução de processos para uma determinada porcentagem de uso da CPU e inserir atrasos na execução de processos. Essas ações devem ser reversíveis de modo a lidar com a ocorrência de respostas contra falso-positivos;

2. Restauração do sistema computacional: compreendem ações, em geral irreversíveis, para restaurar as partes do sistema afetadas em uma intrusão. Essas ações devem ser capazes de colocar o sistema computacional em um estado seguro, de tal forma que futuras análises — como, por exemplo, a extração automatizada da assinatura do ataque e a identificação de ataques posteriores — não venham a ser prejudicadas. Exemplo de ações incluem restauração do sistema de arquivos, reinicialização de processos ou até mesmo uma reinicialização da máquina. Devido a problemas com falso-positivos, essa classe de resposta pode ser requisitada somente pelo Detector Baseado em Evidências.

O Detector Baseado em Evidências também pode terminar uma resposta inata em progresso, caso dentro de um período de tempo não sejam encontradas evidências de intrusão. Isso acontece devido à sua capacidade de identificar ataques de uma maneira bastante confiável.

4.3.8 Extrator de Assinatura

O propósito do Extrator de Assinatura é analisar o sistema após um ataque com sucesso na tentativa de extrair uma assinatura que identifique tal ataque. Essa assinatura irá habilitar uma detecção bastante eficiente do ataque no futuro e, por conseguinte, uma resposta vigorosa. A Figura 4.6 mostra um detalhamento desse componente.

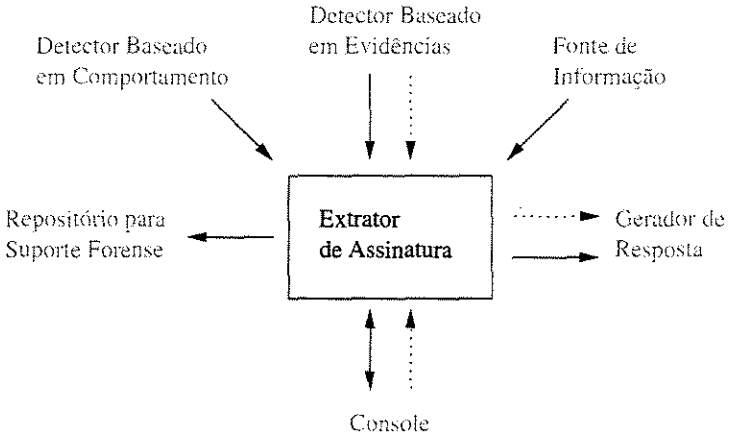

Figura 4.6: Detalhamento do Extrator de Assinatura.

Inicialmente o Extrator de Assinatura é ativado pelo Detector Baseado em Evidências, recebendo informações a respeito de uma intrusão identificada, tais como aplicação,

usuário, portas de comunicação e outras informações envolvidas com a intrusão. Após a restauração do sistema, certamente invocada pelo Detector Baseado em Evidências, o Extrator de Assinatura inicia um processo de busca por uma assinatura do ataque em questão. Essa busca é iniciada com o levantamento de um conjunto de assinaturas candidatas. Após uma fase de seleção ou maturação das candidatas, o Extrator de Assinatura ativa o Gerador de Resposta e repassa a assinatura encontrada. Essa assinatura também é repassada ao Repositório para Suporte Forense e pode ser acessada através do Console, que ainda permite alterar parâmetros de seleção e desabilitar/habilitar o mecanismo de extração.

Embora o papel do Extrator de Assinatura seja bastante claro, as etapas do levantamento das assinaturas candidatas e da seleção dessas assinaturas não são intuitivas. Considerando isso, foi desenvolvido um algoritmo para a extração de assinaturas de ataque que é apresentado na seqüência.

4.3.8.1 Um algoritmo para a extração de assinaturas de ataque

Um algoritmo genérico para o problema da extração de assinatura é proposto neste trabalho [dPdCdG04]. Ele é inspirado pelo processo de seleção negativa do sistema imunológico humano e é adequado para ataques em geral. Conforme já era esperado, esse algoritmo divide a extração de assinatura em duas fases: a busca pelas assinaturas candidatas e a maturação das candidatas.

Ao contrário de outros trabalhos que geram detectores candidatos aleatoriamente [FH00, KB99a, KB99c], o algoritmo proposto aplica a capacidade da detecção baseada em evidências e seleciona eventos anteriores à identificação da intrusão para compor cada assinatura candidata. Como a intrusão, e por conseguinte o ataque, é um fato — em função da detecção baseada em evidências — alguns eventos relacionados com esse ataque devem ocorrer antes dessa detecção baseada em evidências. Uma assinatura candidata é considerada relevante quando ela carrega consigo informações que permitem identificar com precisão o ataque em questão. A abordagem proposta parece ser mais apropriada para encontrar assinaturas candidatas relevantes que a geração aleatória. De fato, o uso mais apropriado da seleção negativa pode ser como um filtro para detectores inválidos, e não para a geração de detectores efetivos [KB01a].

A abordagem proposta neste trabalho é apresentada no Algoritmo 1. Assume-se que antes do início desse algoritmo as partes do sistema afetadas na intrusão já foram restauradas. O Passo 1 envolve a busca pelas assinaturas candidatas e os Passos 2 até 7 realizam a maturação dessas candidatas. O conjunto C deve ser construído de modo a conter eventos relacionados ao ataque sendo analisado. Essas assinaturas candidatas podem ser escolhidas dentre os eventos não usuais provenientes do Detector Baseado em Comportamento ou dentre os eventos capturados pela Fonte de Informação. Em ambos os casos,

devem ser considerados eventos no nível de rede que ocorreram em instantes precedentes à detecção baseada em evidências. Por exemplo, podem ser escolhidos os 1000 eventos que precederam essa detecção ou os eventos que ocorreram nas últimas 24 horas que antecedem a detecção baseada em evidências.

Algoritmo 1: Extração probabilística de assinaturas de ataque.
Entrada: Um número real $p \in]0;1]$, um conjunto E de todos os eventos gerados antes da detecção baseada em evidências e um conjunto N de eventos gerados no sistema na ausência de ataques, onde $N \cap E = \emptyset$. Assume-se que quaisquer eventos gerados em momentos distintos são eventos distintos, independente de haver igualdade entre seus atributos.
Saída: Um conjunto $C \subseteq E$ de eventos, que são as assinaturas de ataque extraídas. Essas assinaturas possuem probabilidade estimada inferior a p de que haverá falso-positivos utilizando-as conjuntamente para a identificação de ataques.
EXTRAIASSINATURAS(E, N, p)
(1) Construa o conjunto $C \subseteq E$ de assinaturas candidatas;
(2) *progresso* $\leftarrow 0$;
(3) **enquanto** *progresso* $< \left\lceil \frac{|C|}{p} \right\rceil$ **faça**
(4) Pegue um novo evento $n \in N$ durante o funcionamento legítimo;
(5) **para cada** $c_i \in C$ **faça**
(6) **se** c_i identifica n **então** $C \leftarrow C \setminus \{c_i\}$;
(7) *progresso* $\leftarrow$ *progresso* $+ 1$;
(8) **retorne** cada assinatura em C ou nulo se $|C| = 0$;

O conjunto N de eventos pode ser obtido de três maneiras:

1. Coletando eventos antes da identificação da intrusão. Nesse caso, os eventos coletados devem ser armazenados de tal modo que eles possam ser recuperados no futuro. Além disso, deve haver uma garantia de que a quantidade de eventos armazenados seja suficiente para possibilitar a execução completa do algoritmo para qualquer tamanho de C e valor de p;

2. Coletando eventos após a identificação da intrusão. Nesse caso, se novas evidências de ataque são encontradas durante ou logo após o processo de extração de assinatura, esse algoritmo deve ser reiniciado com o conjunto inicial C, uma vez que esse novo ataque pode descartar eventos relevantes do ataque anterior;

3. Através de uma combinação das duas opções anteriores. Nesse caso, parte da extração de assinatura é feita utilizando eventos anteriores à identificação da intrusão e o restante é realizado com eventos produzidos após essa identificação.

Em todos os casos, a ocorrência de ataques repetidos pode criar uma oportunidade para correlacionar seus eventos e extrair assinaturas de uma forma mais eficiente.

O critério de identificação do Passo 6 deve levar em conta as classes de ataque consideradas pelo desenvolvedor do sistema de segurança. O comando identifica pode, por exemplo, representar um casamento perfeito ou um casamento parcial de um ou mais atributos dos eventos envolvidos. Como será visto no Capítulo 5, os eventos considerados por ADENOIDS durante a extração de assinatura são requisições no nível de aplicação, as quais são recebidas através da rede. Nesse caso, o critério de identificação utilizado considera apenas os ataques *buffer overflow*, uma vez que somente o tamanho das requisições é analisado.

O algoritmo proposto é capaz de descartar as assinaturas candidatas que identificam o comportamento legítimo do sistema, observado durante a fase de maturação. Portanto, considerando um valor adequado para p, um bom critério de identificação de assinaturas e um conjunto C contendo boas assinaturas candidatas, esse algoritmo irá extrair as assinaturas relacionadas ao ataque e assinaturas para identificar eventos muito raros como, por exemplo, erros.

Quando, ao final do algoritmo, nenhuma assinatura é produzida há algumas suspeitas principais a considerar:

- O Extrator de Assinatura foi ativado devido a um falso-positivo ocasionado pelo Detector Baseado em Evidências. Nesse caso, é importante revisar o método de identificação de evidências considerado;
- O conjunto C não continha boas assinaturas candidatas. Nesse caso, é importante revisar o intervalo de tempo considerado para construir C e/ou as técnicas de análise empregadas pelo Detector Baseado em Comportamento;
- O critério de identificação do Passo 6 do algoritmo não contempla a intrusão identificada pelo Detector Baseado em Evidências. Nesse caso, é necessário revisar esse critério visando cobrir novas classes de ataque.

Levando em conta que a construção de C pode ser realizada continuamente através da análise dos eventos E gerados, o tempo de execução assintótico desse algoritmo depende principalmente de dois fatores: $|C|$ e p. Dessa forma o algoritmo executa, no pior caso, em tempo $O\left(\frac{|C|^2}{p}\right)$. Entretanto, quando os eventos de N são coletados após a identificação da intrusão, o tempo de espera pela geração desses eventos pode influenciar bastante o tempo de execução real do algoritmo.

Resultados experimentais apresentados no Capítulo 6 reafirmam que esse algoritmo pode produzir assinaturas de ataque relevantes, as quais podem ser utilizadas tanto no Detector Baseado em Conhecimento como em outros IDSs baseados em conhecimento.

4.3.9 Gerador de Resposta

O Gerador de Resposta é responsável por escolher um conjunto de medidas para bloquear ataques relacionados a cada assinatura de ataque extraída. Esse componente é ativado pelo Extrator de Assinatura, que também repassa as assinaturas extraídas. De acordo com os atributos de uma assinatura, o Gerador de Resposta escolhe o tipo de resposta mais adequado para essa assinatura. Por exemplo, quando a assinatura contempla uma requisição HTTP, o tipo de resposta mais adequado consiste no bloqueio de um certo tráfego de rede — tipo de resposta 0, considerando o exemplo da Tabela 4.2 na Seção 4.3.4. Nesse caso, o tráfego de rede a ser bloqueado tem como destino a porta 80 e contém a *string* expressa na assinatura. Em um segundo exemplo, quando a assinatura contém uma tentativa de estabelecimento de um canal de comunicação PPPoE, uma resposta adequada poderia consistir em desabilitar a interface de rede ppp0 local — tipo de resposta 1, considerando o exemplo da Tabela 4.2.

Após a escolha das medidas cabíveis, as quais devem ser suportadas pelo Agente de Resposta Adaptativa, a assinatura e o tipo de resposta são inseridos no Banco de Dados de Assinaturas e Respostas do Detector Baseado em Conhecimento. Com isso, uma nova exposição a esse ataque poderá ser bloqueado antes que ocorram danos ao sistema.

4.3.10 Repositório para Suporte Forense

Esse componente é responsável pelo armazenamento de informações coletadas, principalmente durante a identificação positiva de ataques pelo Detector Baseado em Evidências. Ele é modelado para prover suporte à análise forense manual, preservando dados que não podem ser corrompidos mesmo após a restauração do sistema [Ste00a].

Observando os dados coletados pelo Repositório para Suporte Forense é possível desvendar os passos que foram empreendidos no ataque. Dessa forma, as informações coletadas podem servir como sustentação durante uma ação judicial [dR03]. Esse repositório também pode ser utilizado para possibilitar uma análise manual que tem por objetivo verificar a qualidade das assinaturas obtidas através do Extrator de Assinatura.

Exemplos de informações que podem ser relevantes para esse repositório incluem processos em execução, arquivos abertos, conteúdo de arquivos de configuração, principais binários do sistema, módulos de *kernel* carregados, quantidade de memória disponível, quantidade de acessos a disco, interfaces de rede ativas, tráfego de rede nas últimas horas, e até mesmo um *dump* da memória RAM.

4.3.11 Auto-proteção

Como a arquitetura apresentada assume que intrusões são inevitáveis, seus componentes devem ser protegidos de tal modo que eles não sejam danificados durante um ataque. Essa proteção pode ser implementada basicamente no nível do sistema operacional, restringindo o acesso a partir de potenciais alvos de ataques. Esses potenciais alvos, de acordo com o modelo de ameaça considerado, consistem basicamente em aplicações de usuário que podem ser utilizadas e/ou subvertidas na tentativa de danificar o sistema de segurança. Sendo assim, o acesso à configuração desse sistema de segurança e o controle da execução dos componentes podem estar restritos somente ao acesso local, sem intermédio de aplicações suspeitas.

Em um exemplo, o acesso aos arquivos de configuração do sistema de segurança pode ser negado a aplicações monitoradas pelo Detector Baseado em Evidências, independente das permissões do usuário em questão. Nesse caso, essas aplicações são possíveis alvos de ataque e podem ser utilizadas como um trampolim na tentativa de burlar o sistema. Em uma análise mais cuidadosa, nenhum acesso a esses arquivos a partir de processos descendentes de uma aplicação monitorada pode ser autorizado.

Tal esquema de auto-proteção pode ser obtido através de alterações mínimas no sistema operacional ou através da utilização de *patches* já desenvolvidos, como LIDS (Linux Intrusion Detection System) [Hua00].

4.4 Funcionamento geral

Uma vez conhecida a funcionalidade de cada componente, é importante entender o funcionamento global da arquitetura proposta. A Figura 4.7 apresenta um fluxograma que descreve o funcionamento geral dessa arquitetura. Considerando essa perspectiva nota-se que:

1. Inicialmente um evento é gerado no sistema computacional;

2. Esse evento é analisado pelo Detector Baseado em Conhecimento e, caso represente um ataque conhecido, acaba sendo bloqueado pelo Agente de Resposta Adaptativa. Considerando um funcionamento ideal da arquitetura, um ataque conhecido é bloqueado antes que eventos não usuais sejam identificados pelo Detector Baseado em Comportamento e antes que alguma evidência de intrusão seja identificada pelo Detector Baseado em Evidências;

3. Se esse evento não representa um ataque conhecido, então ele será incluído nos processos de análise comportamental e de análise baseada em evidências de intrusão.

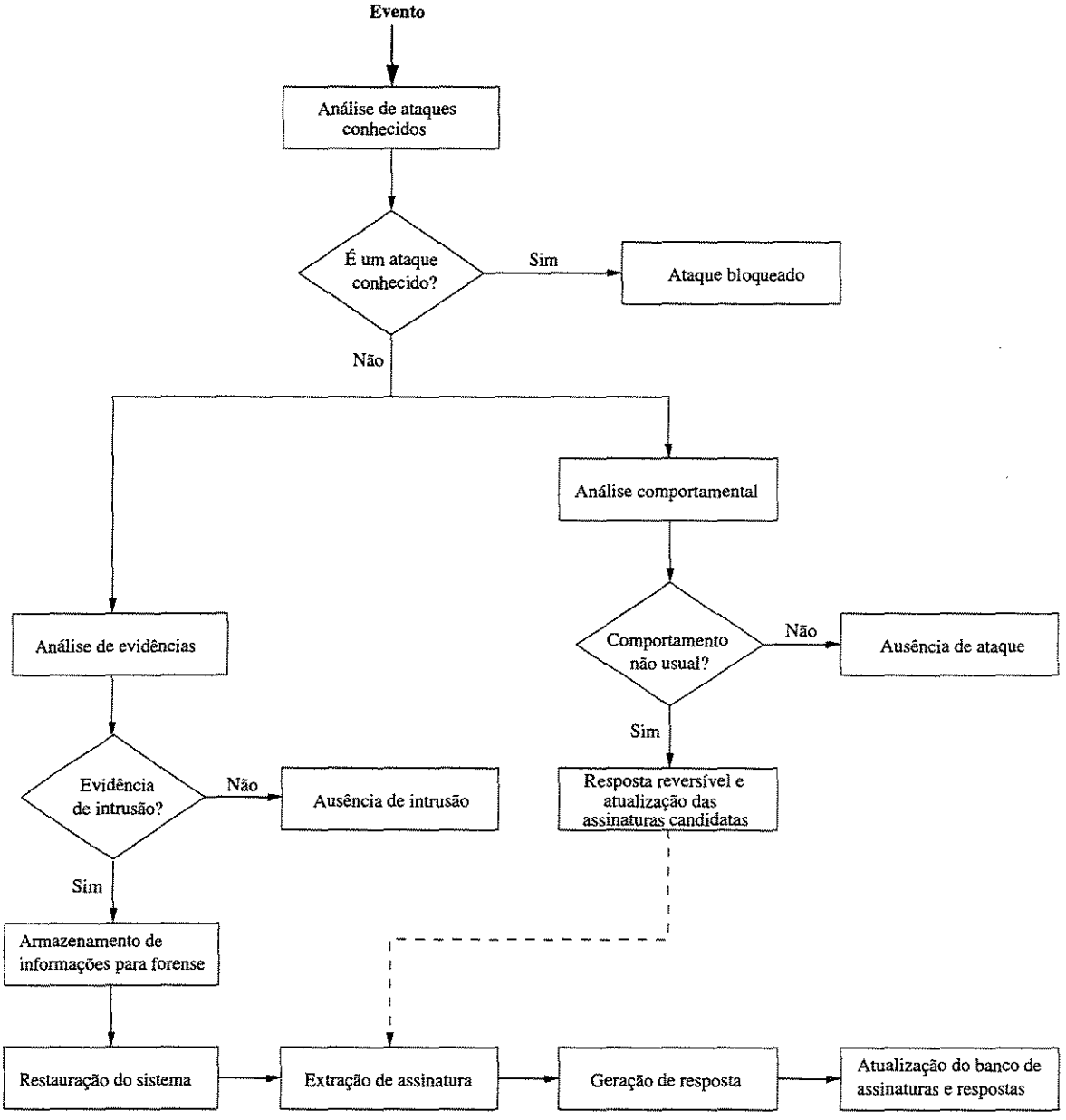

Figura 4.7: Fluxograma para o funcionamento geral da arquitetura proposta.

Caso tal evento contribua para a identificação de um comportamento não usual pelo Detector Baseado em Comportamento, então uma resposta reversível é realizada pelo Agente de Resposta Inata e esse evento passa a compor o conjunto de assinaturas candidatas. Caso contrário, nenhuma ação é iniciada pelo Detector Baseado em Comportamento. Se esse evento contribui para a identificação de evidências de intrusão pelo Detector Baseado em Evidências, então uma série de ações são desencadeadas (caso contrário, nenhuma ação é iniciada por esse detector):

(a) São coletadas informações relacionadas com a intrusão detectada. Essas informações são armazenadas pelo Repositório para Suporte Forense;

(b) O sistema computacional é restaurado pelo Agente de Resposta Inata, eliminando possíveis danos em arquivos e processos subvertidos. Essa etapa é bastante importante para restabelecer o funcionamento normal do sistema monitorado e eliminar possíveis *backdoors* e cavalos-de-tróia;

(c) O Extrator de Assinatura é ativado e, ao fim de sua análise, é apresentado um conjunto de assinaturas que identificam o ataque ocorrido. Esse componente pode utilizar como assinaturas candidatas os eventos não usuais provindos do Detector Baseado em Comportamento;

(d) O Gerador de Resposta elabora as medidas necessárias para bloquear o ataque em questão, considerando as assinaturas extraídas;

(e) O Banco de Dados de Assinaturas e Respostas do Detector Baseado em Conhecimento é atualizado após a geração da resposta. Sendo assim, o sistema computacional estará preparado para lidar com esse ataque em futuras exposições. O Repositório para Suporte Forense também poderá armazenar informações a respeito das assinaturas extraídas e da resposta elaborada.

Na ausência de ataques, idealmente nenhum componente de detecção irá realizar uma identificação, com exceção do Detector Baseado em Comportamento, que pode produzir falso-positivos isolados. Entretanto, esses falso-positivos poderão desencadear, no máximo, uma reação suave e reversível no sistema computacional, sem afetar o funcionamento básico desse sistema.

Apesar da descrição funcional ser apresentada em uma seqüência lógica, é necessário entender que os componentes do sistema trabalham concorrentemente. Por exemplo, enquanto uma assinatura está sendo extraída, um ataque conhecido pode ser identificado e bloqueado, ou um ataque desconhecido pode obter sucesso e, após a identificação de evidências, desencadear uma nova restauração e extração de assinatura.

Essencialmente, todos os componentes envolvidos em alguma detecção devem estar sempre ativos: Fonte de Informação, Detector Baseado em Conhecimento, Detector

Baseado em Comportamento e Detector Baseado em Evidências. Para prover informações de boa qualidade, o Repositório para Suporte Forense também deve permanecer ativo. Os demais componentes podem permanecer inativos, sendo ativados sob demanda quando alguma identificação é realizada. Desse modo é possível economizar recursos computacionais e viabilizar o funcionamento global da arquitetura.

4.4.1 Momento da análise

O componente mais crítico em relação ao momento da análise é o Detector Baseado em Conhecimento. Esse componente deve trabalhar em tempo real para cumprir com o compromisso de identificar ataques e possibilitar um bloqueio antes da ocorrência de intrusão e danos. Os demais componentes de detecção, o Detector Baseado em Comportamento e o Detector Baseado em Evidências, podem trabalhar em um ritmo mais conservador. Entretanto, deve-se observar que uma análise em modo *batch* infreqüente pode atrasar a ativação dos mecanismos de reserva de recursos e restauração do sistema providos pelo Agente de Resposta Inata, bem como o início do processo de extração de assinatura.

4.5 Analogias com o sistema imunológico

Embora o enfoque principal desta pesquisa não tenha compreendido um mapeamento aprofundado do sistema imunológico, esse sistema biológico foi bastante utilizado como uma fonte inspiradora. Em resumo, a meta era projetar um sistema de segurança que pudesse conter algumas das principais características do sistema imunológico humano, sem a preocupação de conduzir essa analogia até níveis celulares ou químicos.

Assim como o sistema biológico, a arquitetura proposta possui as seguintes virtudes:

- Identificação precisa de ataques[1] conhecidos através de um mecanismo de memória adquirida;
- Eliminação eficiente de ataques conhecidos;
- Capacidade de identificar de uma forma similar e não específica ataques desconhecidos;
- Provisão de resposta inespecífica, mas importante, para ataques desconhecidos;
- Restauração das partes afetadas em um ataque, eliminando os agentes invasores;

[1] ... ou agentes infecciosos, no caso do sistema imunológico.

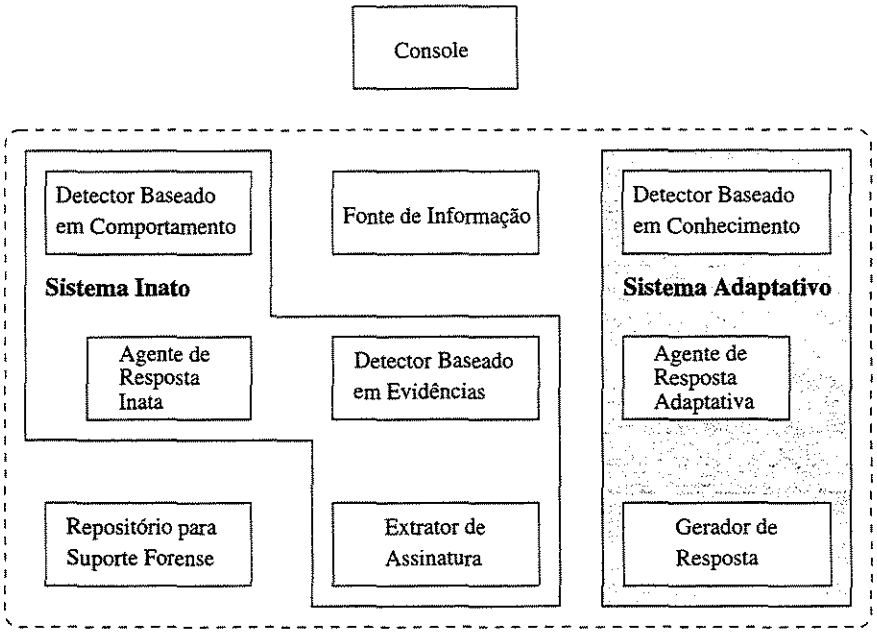

Figura 4.8: Abrangência dos sistemas inato e adaptativo.

- Análise de ataques desconhecidos de maneira a "memorizar" esse ataque, possibilitando identificações mais precisas em futuras exposições.

A arquitetura proposta também pode ser vista sob a perspectiva dos sistemas inato e adaptativo, como ilustra a Figura 4.8. Dessa forma, o sistema de segurança inato compreende os componentes que atuam de uma forma similar e inespecífica para diferentes tipos de ataque: Detector Baseado em Comportamento, Detector Baseado em Evidências, Agente de Resposta Inata e Extrator de Assinatura. Esse sistema também é responsável por ativar o sistema de segurança adaptativo durante um ataque desconhecido. Já o sistema de segurança adaptativo é capaz de tratar cada ataque especificamente, seja na identificação do ataque ou no desenvolvimento de uma resposta adaptativa. Os componentes que compreendem esse sistema são: Detector Baseado em Conhecimento, Agente de Resposta Adaptativa e Gerador de Resposta. A Fonte de Informação e o Console também podem assumir um papel relacionado com o sistema imunológico, embora não estejam inseridos no sistema inato ou adaptativo. O Repositório para Suporte Forense foi modelado para disponibilizar uma funcionalidade computacional desejada, sem buscar inspiração no sistema biológico.

A Tabela 4.3 mostra ainda uma analogia entre os componentes da arquitetura e o sistema imunológico humano, excetuando o componente Repositório para Suporte Forense.

Como pode ser notado, as analogias existentes entre a arquitetura proposta e o sistema imunológico ocorrem principalmente no nível arquitetural.

Componentes da arquitetura	**Sistema imunológico humano**
Console	Imunidade adquirida artificialmente
Fonte de Informação	Fonte de proteínas *self* e *nonself*
Detector Baseado em Comportamento	Detecção de anomalia do sistema inato
Detector Baseado em Evidências	Detecção de danos
Agente de Resposta Inata	Resposta inespecífica do sistema inato
Extrator de Assinatura	Digestão de antígenos e apresentação de seus fragmentos
Detector Baseado em Conhecimento	Detecção precisa através de células de memória de alta afinidade
Agente de Resposta Adaptativa	Resposta específica do sistema adaptativo
Gerador de Resposta	Produção de anticorpos específicos

Tabela 4.3: Analogia entre os componentes da arquitetura e o sistema imunológico humano.

Essa arquitetura aplica diversos princípios imunológicos, como apresentados na Seção 3.1.3: distribuição e paralelismo, multicamada, autonomia, adaptabilidade e memória, identificação por comportamento, detecção de anomalia, detecção de danos, e recrutamento dinâmico. Além desses princípios, outros também podem ser contemplados pela adição de novas características em uma implementação. Por exemplo, uma implementação pode considerar o casamento parcial de assinaturas de ataque e o monitoramento/restauração dos próprios componentes da arquitetura.

4.5.1 Arquitetura proposta e a Teoria do Perigo

Este trabalho também está amplamente relacionado com a Teoria do Perigo apresentada na Seção 3.2.10. O Detector Baseado em Evidências é responsável por identificar uma intrusão já ocorrida, que pode ser descoberta através da análise de danos no sistema. Esses danos ocorrem especialmente em arquivos de configuração, binários do sistema e processos em execução, indicando um perigo iminente para o funcionamento legítimo desse sistema. A arquitetura proposta utiliza uma zona de perigo temporal com as seguintes características:

- Termina no momento t_f em que a intrusão é identificada pelo Detector Baseado em Evidências;

- Inicia em um momento $t_i = t_f - \Delta$, onde Δ é um período de tempo pré-especificado, como, por exemplo, $\Delta = 24$ horas.

Sendo assim, durante o processo de extração de assinatura é assumido que a assinatura do ataque ocorre dentro da zona de perigo, isto é, entre os instantes t_i e t_f. Analisando os eventos produzidos dentro da zona de perigo são levantadas as assinaturas candidatas, provenientes do Detector Baseado em Comportamento ou da Fonte de Informação.

Após a fase de maturação das candidatas espera-se obter um conjunto de assinaturas que identifica a fonte do perigo — o ataque — de uma forma bastante eficiente. Essa estratégia parece ser muito mais adequada para encontrar boas assinaturas quando comparada com estratégias aleatórias para a discriminação entre *self/nonself* em computadores.

4.6 Alternativas de implementação

Antes de iniciar uma implementação baseada na arquitetura proposta é necessário verificar quais classes de ataque serão consideradas. Algumas classes de ataque que envolvem questões como confidencialidade e autenticidade podem ser abordadas eficientemente utilizando técnicas de criptografia [SG96]. Outras classes de ataque que envolvem a manipulação de cabeçalhos no nível de rede e transporte da pilha TCP/IP podem ser abordadas através de uma análise sistemática desses protocolos, como ocorre em diversos IDSs atuais. Por outro lado, aplicações que provêem algum tipo de serviço público têm sido o maior alvo de ataques nos últimos anos [Cen04a] e é justamente para esse universo de ataques que a arquitetura proposta pode ter o seu melhor desempenho.

Após considerar as classes de ataque, um projetista deve verificar quais funcionalidades são desejadas no sistema de segurança. Essa verificação deve ocorrer porque a arquitetura proposta apresenta um sistema completo de defesa que pode ser simplificado para atender necessidades particulares. Por exemplo, um sistema de segurança baseado nessa arquitetura funciona perfeitamente sem o componente Repositório para Suporte Forense, porém irá perder a funcionalidade de suportar uma análise forense manual. A funcionalidade associada a componente e a essencialidade desses componentes em um sistema de segurança são descritas como segue:

- Console: provê uma interface com o administrador e, embora a arquitetura seja automatizada, tende a ser um componente necessário em toda implementação;

- Fonte de Informação: provê toda informação a ser analisada, sendo fundamental em qualquer sistema;
- Detector Baseado em Evidências: todo aparato de restauração do sistema e extração de assinatura depende da detecção baseada em evidências de intrusão, tornando esse componente essencial;
- Extrator de Assinatura: oferece a funcionalidade de extrair assinaturas automaticamente, sendo necessário em todo sistema que objetiva esse tipo de análise. Essa funcionalidade é um dos principais atrativos da arquitetura proposta;
- Gerador de Resposta: elabora as medidas para as assinaturas extraídas e, portanto, esse componente só faz sentido se o Extrator de Assinatura também for utilizado. Entretanto, um sistema desenvolvido para apenas extrair a assinatura não necessita do Gerador de Resposta;
- Detector Baseado em Conhecimento: identifica ataques conhecidos, sendo mais interessante quando o componente Gerador de Resposta é utilizado. Não é essencial quando somente a extração da assinatura é requerida;
- Agente de Resposta Adaptativa: executa uma resposta para ataques conhecidos e depende do Detector Baseado em Conhecimento. Deve ser descartado quando esse detector não é utilizado e pode ser desconsiderado quando almeja-se apenas identificar a ocorrência de ataques conhecidos;
- Detector Baseado em Comportamento: identifica comportamento não usual, sendo bastante interessante para ativar uma resposta reversível e/ou encontrar assinaturas candidatas. Quando essas funcionalidades não são desejadas esse componente pode ser desconsiderado;
- Agente de Resposta Inata: provê as respostas reversíveis e a restauração do sistema. Pelo menos essa restauração é bastante importante na etapa de extração de assinatura, o que torna esse componente um pré-requisito para o Extrator de Assinatura;
- Repositório para Suporte Forense: deve ser utilizado somente quando uma análise forense manual pode ser considerada viável.

Essa descrição torna a arquitetura flexível, possibilitando desenvolver sistemas de segurança/análise que sejam bastante adequados para determinados ambientes computacionais. Uma implementação também pode considerar o uso de ferramentas já implementadas por terceiros e que cumprem a funcionalidade de alguns componentes especificados.

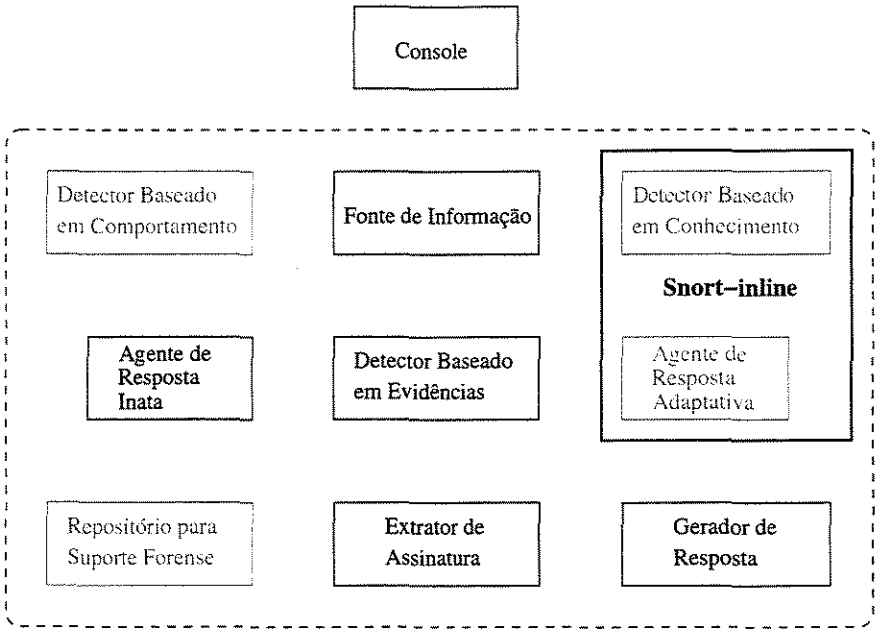

Figura 4.9: Possível arranjo de componentes em uma implementação simplificada.

A Figura 4.9 mostra um possível arranjo de componentes em uma implementação simplificada. Nesse caso são suprimidos os componentes Detector Baseado em Comportamento — o que obriga o Extrator de Assinatura buscar as candidatas na Fonte de Informação — e Repositório para Suporte Forense. Também, os componentes Detector Baseado em Conhecimento e Agente de Resposta Adaptativa são substituídos pela ferramenta Snort-inline [HM04].

Outra questão que deve ser abordada em uma implementação diz respeito à localização de cada componente no sistema computacional. Essa questão é apresentada como segue.

4.6.1 Localização dos componentes

Quanto à localização dos componentes, uma implementação pode ser totalmente distribuída ou parcialmente distribuída. Em uma implementação totalmente distribuída cada máquina em uma rede possui um sistema de segurança completo, contendo todos os componentes especificados. Em uma implementação parcialmente distribuída, parte dos componentes são encontrados em um nó central de uma rede, enquanto outros componentes são localizados nas estações de trabalho. A Figura 4.10 apresenta uma sugestão para a localização dos componentes em uma implementação parcialmente distribuída. De

acordo com essa sugestão:

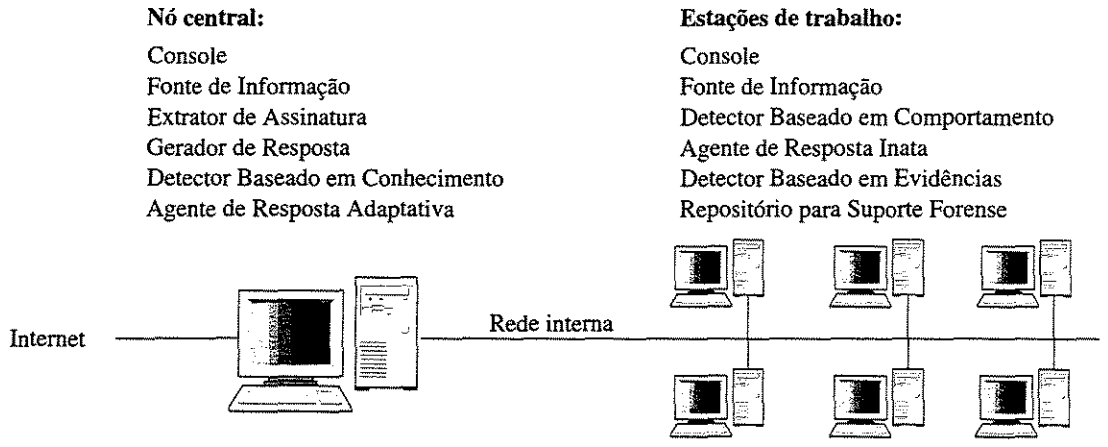

Figura 4.10: Uma sugestão para a localização dos componentes em uma implementação parcialmente distribuída.

- Cada estação de trabalho busca por evidências de intrusão localmente, efetua a resposta inespecífica, reversível e/ou restauradora, e pode levantar as assinaturas candidatas após uma intrusão, bem como armazenar informações para uma futura análise forense manual. Uma estação ainda é responsável por avisar o nó central da ocorrência de uma intrusão;

- O nó central é responsável por iniciar o processo de extração de assinatura e geração de resposta, baseado nas requisições das estações. Utilizando as assinaturas, esse nó protege as estações contra ataques provenientes da Internet.

É interessante notar que uma implementação totalmente distribuída não possui um ponto central de falhas e pode ser mais escalável. Por sua vez, uma implementação parcialmente distribuída é mais administrável e pode prover uma maior proteção para os componentes localizados no nó central. Porém, essa última deve utilizar uma comunicação segura entre o nó central e as estações de trabalho e, dependendo da localização do Detector Baseado em Conhecimento e do Agente de Resposta Adaptativa, os ataques internos não são bloqueados, como pode acontecer na sugestão apresentada.

Uma implementação totalmente centralizada é inviabilizada pela necessidade do Detector Baseado em Evidências atuar com informações coletadas no nível de máquina ou aplicação. Dessa maneira, pelo menos parte da Fonte de Informação deve residir em cada estação de trabalho.

4.7 Trabalhos relacionados

As principais pesquisas envolvendo técnicas para identificação de ataques foram apresentadas na Seção 2.4, compreendendo a detecção baseada em conhecimento, a detecção baseada em comportamento e métodos alternativos de detecção. As pesquisas mais relevantes na área da detecção de intrusão baseada no sistema imunológico, apresentadas na Seção 3.2, também concentram-se no emprego da detecção baseada em comportamento. Como já discutido anteriormente, tais métodos de detecção apresentam problemas salientes, falhando ao identificar ataques desconhecidos ou possuindo altas taxas de falso-positivos.

Os esforços aplicados na detecção de intrusão até o momento buscam desenvolver um sistema que seja livre de intrusões e com uma taxa aceitável de falso-positivos. O presente trabalho procura ser mais realista com a situação atual da detecção de intrusão, assumindo que ataques com sucesso são inevitáveis e estando preparado para lidar com tal situação.

Este trabalho também propõe um mecanismo automatizado para a extração de assinatura de ataques, visando transformar ataques desconhecidos em conhecidos. Pesquisas anteriores apenas buscaram gerar regras de detecção através de uma análise comportamental [LV92, TCL90, GD02]. Além dos problemas decorrentes da análise comportamental, essas regras compreendem uma seqüência de valores combinados e não uma assinatura de ataque compreendendo uma cadeia de caracteres qualquer. Nesse ponto é importante ressaltar que essas pesquisas "criam" regras que podem ser utilizadas na detecção. Já este trabalho "extrai" assinaturas, que são eventos que realmente ocorreram em momentos de intrusão.

Considerando tais aspectos, este trabalho possui uma relação forte somente com a linha de pesquisa de Kephart [Kep94, KA94, KSSW97]: um sistema para identificação de vírus. Embora essa relação exista, a arquitetura proposta neste trabalho possui algumas distinções essenciais:

1. Aborda a detecção de intrusão e não identificação de vírus;

2. Na pesquisa de Kephart uma detecção de anomalia é realizada antes de qualquer outra análise, com um comportamento não usual indicando o início do processo de análise e eliminação de vírus. Neste trabalho, uma detecção baseada em conhecimento pode ocorrer antes da detecção baseada em comportamento e a fase de análise ou extração de assinatura inicia com a detecção de evidências de intrusão;

3. Neste trabalho a detecção baseada em comportamento pode ser realizada para prover uma resposta inata e encontrar as assinaturas de ataque candidatas. No sistema de Kephart as assinaturas candidatas são capturadas analisando um conjunto de *decoys* tocados por vírus;

4. Como um ataque pode causar danos extensivos em um sistema monitorado, a arquitetura proposta provê um método genérico para restaurar esse sistema. O método de restauração proposto por Kephart é mais simples porque uma grande classe de vírus são não destrutivos [KSSW97].

A detecção baseada em evidências empregada possui um papel fundamental na arquitetura proposta, fato que também não foi explorado devidamente em outras pesquisas na área.

4.8 Conclusão

Este capítulo apresentou uma arquitetura de segurança inspirada no sistema imunológico humano. Embora essa arquitetura não tenha sido desenvolvida para contemplar detalhes do sistema biológico ela herda desse sistema as principais características de identificação de ataques conhecidos, manipulação de situações com ataques desconhecidos e mecanismos de restauração e resposta.

Um dos principais atrativos dessa arquitetura é especificar um mecanismo para extrair assinaturas de ataque automaticamente. Tal mecanismo emprega um algoritmo para extração de assinatura probabilístico, também proposto neste trabalho. Dessa forma é possível tornar ataques desconhecidos em ataques conhecidos, habilitando uma rápida detecção e resposta contra futuras exposições. Sendo assim, este trabalho traz uma contribuição saliente para a área de pesquisa da detecção de intrusão.

Foram discutidas, ainda, as principais analogias existentes entre o sistema imunológico e a arquitetura proposta, algumas alternativas de implementação e o relacionamento com outros trabalhos realizados.

Utilizando as idéias contidas neste capítulo este trabalho apresenta, na seqüência, um protótipo de sistema de segurança enfocando ataques a aplicações e extração de assinatura para ataques *buffer overflow*. Dessa maneira, tal protótipo vem validar as principais idéias propostas até aqui, considerando uma das mais importantes classes de ataque existentes.

Capítulo 5

O protótipo ADENOIDS

Este capítulo apresenta o protótipo de sistema de segurança ADENOIDS, que é uma aplicação da arquitetura proposta no Capítulo 4. ADENOIDS tem por objetivo validar características e funcionalidades propostas pela arquitetura desenvolvida, propiciando a execução de testes e a obtenção de resultados práticos. Como uma avaliação de todos os aspectos práticos e arquiteturais encontra-se além do escopo deste trabalho, essa validação será restrita às principais características e funcionalidades da arquitetura proposta.

ADENOIDS foi desenvolvido para monitorar um computador isolado, identificando evidências de intrusão expressas por aplicações em execução e extraindo assinaturas para ataques *buffer overflow* remotos. O mecanismo de extração de assinatura prototipado está concentrado inicialmente nos ataques *buffer overflow*. Entretanto, o mecanismo de identificação de evidências é capaz de detectar intrusões diversificadas, envolvendo, por exemplo, *buffer overflow*, *format string*, *race conditions*[1], dentre outras técnicas empregadas em ataques a aplicações. A partir deste momento, cada componente implementado da arquitetura será chamado de módulo.

A Seção 5.1 apresenta os módulos do protótipo ADENOIDS e seu relacionamento com a arquitetura proposta. As Seções 5.2 até 5.8 descrevem de uma forma mais detalhada cada módulo, incluindo particularidades da implementação realizada. Na Seção 5.9 é descrita uma ferramenta auxiliar para a restauração do sistema, que também foi implementada. A Seção 5.10 relata o mecanismo de auto-proteção utilizado em ADENOIDS e a Seção 5.11, ao final, exemplifica o funcionamento geral do sistema de segurança prototipado.

[1]Mais informações a respeito dessas técnicas de ataque podem ser encontradas em [NdG03].

5.1 Módulos de ADENOIDS

Nem todos os componentes especificados pela arquitetura são implementados em ADENOIDS. Os componentes implementados são destacados na Figura 5.1. O Detector Baseado em Conhecimento e o Agente de Resposta Adaptativa não foram implementados, mas as suas funcionalidades podem ser alcançadas através de uma ferramenta como Snort-inline [HM04]. O Gerador de Resposta também não foi implementado. Porém, o bloqueio de pacotes com *overflow* torna-se uma boa alternativa de resposta, que também pode ser oferecida pelo Snort-inline.

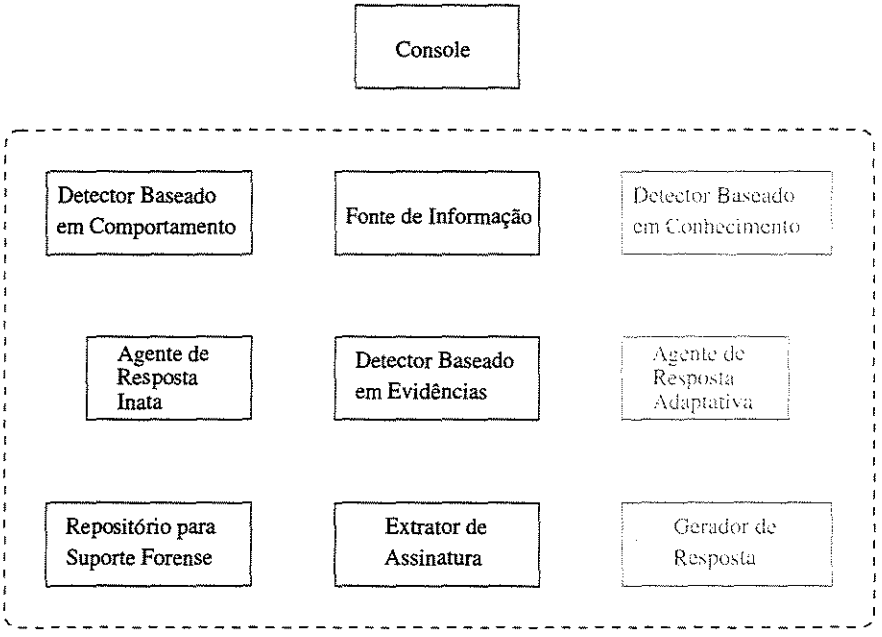

Figura 5.1: Componentes da arquitetura implementados em ADENOIDS.

Os módulos de ADENOIDS recebem um nome específico, como mostra a Tabela 5.1, que também identifica o relacionamento de cada módulo com a arquitetura proposta. Além desses módulos, também foi implementada a ferramenta UNDOFS que é utilizada pelos módulos ADIRA e ADFSR. UNDOFS possibilita restaurar o sistema de arquivos através de mecanismos de *undo* e *redo*.

A Figura 5.2 mostra novamente os componentes da arquitetura implementados em ADENOIDS. Entretanto, eles são ilustrados de uma forma rearranjada, exibindo os fluxos de comunicação — informação e controle — e referenciando os módulos de ADENOIDS. É importante observar que nem todo fluxo de comunicação proposto na arquitetura orig-

Módulos de ADENOIDS	Relacionamento com a arquitetura proposta
ADCON	Console
ADDS	Fonte de Informação
ADEID	Detector Baseado em Evidências
ADBID	Detector Baseado em Comportamento
ADIRA	Agente de Resposta Inata
ADSIG	Extrator de Assinatura
ADFSR	Repositório para Suporte Forense

Tabela 5.1: Módulos de ADENOIDS e relacionamento com a arquitetura proposta.

inal está implementado em ADENOIDS. Nesse protótipo, o módulo ADFSR implementa somente um acesso à ferramenta UNDOFS e, portanto, não necessita obter informação do módulo ADDS ou do módulo ADSIG. Entretanto ADFSR é acessível manualmente. Em outro caso, ADBID também não está empenhado na ativação de uma resposta reversível e, portanto, ADIRA é ativado somente por ADEID.

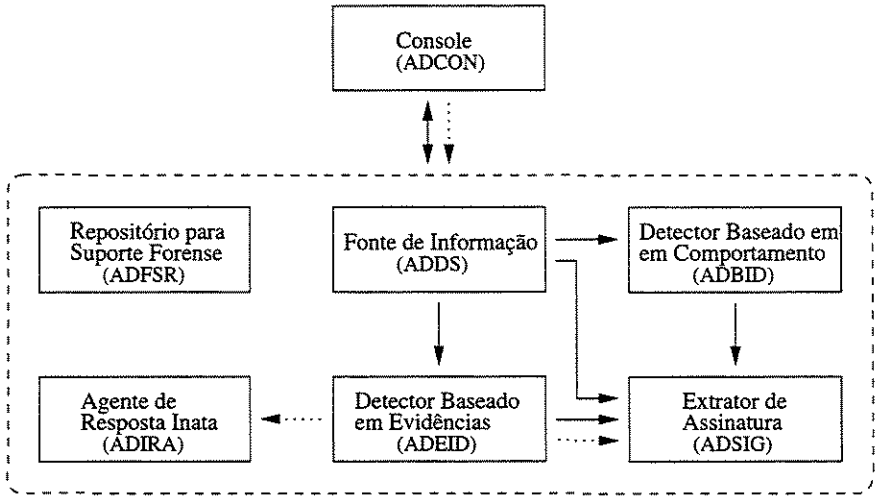

Figura 5.2: Componentes implementados mostrados de uma forma rearranjada, exibindo os fluxos de comunicação e referenciando os módulos de ADENOIDS.

A Figura 5.3 mostra um fluxograma que descreve o funcionamento geral de ADENOIDS. O módulo ADBID faz uma análise comportamental do tráfego de rede em busca de eventos relacionados com ataques *buffer overflow*. Caso um evento em questão seja identificado como não usual, então ele pode estar relacionado com um ataque e, por-

tanto, será classificado como uma assinatura candidata. Caso contrário, ADBID não realiza nenhuma ação. Se o evento analisado por ADEID contribui para a identificação de evidências de intrusão, então o sistema é restaurado por ADIRA e o processo de extração de assinatura é iniciado por ADSIG, utilizando como assinaturas candidatas os eventos não usuais identificados por ADBID. Caso ADEID não identifique uma intrusão, então nenhuma ação é disparada por esse detector. A etapa de coleta de evidências é realizada constantemente através da ferramenta UNDOFS e não é descrita nessa figura. Para acessar essas evidências é utilizado o módulo ADFSR.

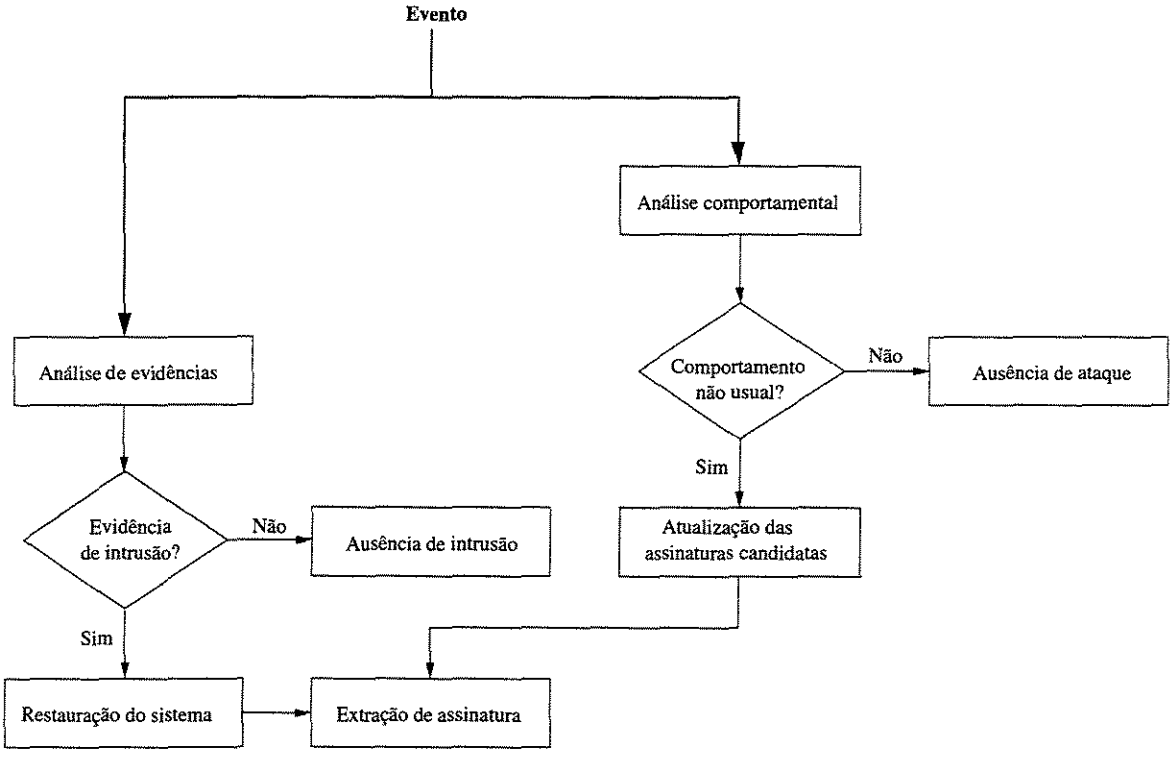

Figura 5.3: Fluxograma para o funcionamento geral de ADENOIDS.

Quando um ataque não utiliza a técnica *buffer overflow*, a intrusão ocasionada também poderá ser identificada por ADEID, levando a uma restauração do sistema. Entretanto, os eventos não usuais detectados por ADBID podem deixar de conter prováveis assinaturas para o ataque. Sendo assim, o processo de extração de ADSIG será prejudicado e poderá terminar sem assinaturas para o ataque, ou poderá resultar na extração de assinaturas que identificam eventos raros como, por exemplo, erros na comunicação. Quando ADENOIDS não está sob ataques ele se comporta exatamente como na arquitetura desenvolvida.

A implementação de todos os módulos é abordada a partir da Seção 5.2, enquanto a Seção 5.9 apresenta a ferramenta UNDOFS. Antes, porém, cabe manifestar qual foi o ambiente de desenvolvimento utilizado na implementação. Esse ambiente é descrito na seqüência.

5.1.1 Ambiente de desenvolvimento

Durante a implementação foi utilizado um computador Athlon XP 1900+ com 512MB DDR RAM, disco rígido de 80GB Ultra-DMA IDE 7200 RPM. Foi utilizada a distribuição Red Hat Linux 9.0 executando um *kernel* versão 2.4.19. Esse *kernel* foi recompilado após a aplicação do *patch* `host-skas3.patch` para suportar a execução da máquina virtual User-mode Linux [Dik01].

ADENOIDS foi implementado majoritariamente dentro do *kernel*. Considerando esse fato, foi utilizada uma máquina virtual User-mode Linux (*patch* `uml-patch-2.4.19-50`) como um ponto de partida para a implementação, facilitando todo o processo de desenvolvimento.

A partir do sistema *host* Red Hat Linux 9.0 foi possível introduzir as alterações necessárias no *kernel* do sistema *guest* User-mode Linux e executar esse sistema *guest* alterado como uma máquina virtual. Dessa forma, ADENOIDS foi desenvolvido para ser executado a partir de uma máquina virtual User-mode Linux. Apesar dessa restrição, a implementação não utiliza mecanismos específicos da máquina virtual, podendo ser portada para um sistema real.

Durante toda a etapa de implementação foi utilizada a linguagem de programação C e o compilador `gcc` versão 3.2.2.

5.2 O módulo ADCON: o Console

O módulo ADCON, que realiza a interface com o administrador, é provido através de um conjunto de arquivos de configuração, que permite interferir no funcionamento de ADENOIDS, e um conjunto de arquivos de *log*, que permite observar o estado atual do sistema. Os arquivos de configuração residem no diretório `/etc/adenoids` enquanto os arquivos de *log* concentram-se no diretório `/var/log/adenoids`. A Tabela 5.2 mostra os principais arquivos considerados e a natureza da informação utilizada em cada um desses arquivos.

O arquivo `policies.conf`, que contém as políticas de acesso para aplicações, deve ser editado para contemplar as aplicações que são potenciais alvos de ataque. Considerando as políticas de acesso especificadas nesse arquivo o módulo ADEID realiza a detecção baseada em evidências de intrusão. O arquivo `undofs.conf`, que contém os diretórios considerados pela ferramenta UNDOFS, deve ser ajustado para abrigar os diretórios que

Arquivo	Natureza da informação utilizada
`/etc/adenoids/adeid/policies.conf`	Políticas de acesso para aplicações
`/etc/adenoids/adira/enabled`	*Flag* para ativar/desativar ADIRA
`/etc/adenoids/adfsr/enabled`	*Flag* para ativar/desativar ADFSR
`/etc/adenoids/undofs/undofs.conf`	Diretórios considerados por UNDOFS
`/var/log/adenoids/adeid`	Resultados do monitoramento de ADEID
`/var/log/adenoids/adbid/SNAME`	Tráfego capturado do serviço `SNAME`
`/var/log/adenoids/adbid/*.abnormal`	Tráfego não usual de cada serviço
`/var/log/adenoids/adsig/messages`	Resultados das extrações de assinatura

Tabela 5.2: Principais arquivos de configuração/*log* e natureza da informação utilizada.

contêm arquivos essenciais para o bom funcionamento do sistema computacional. Todo o conteúdo desses diretórios pode ser restaurado em um determinado momento utilizando técnicas de *undo* e *redo*. Por sua vez, o arquivo `messages`, que contém os resultados das extrações de assinatura realizadas pelo módulo ADSIG, pode ser consultado para avaliar o estado de um processo de extração ou para verificar as assinaturas de ataque extraídas.

Alguns parâmetros de análise utilizados pelos módulos ADEID, ADBID e ADSIG não necessitam de ajustes regulares e, então, não estão associados a um arquivo de configuração no diretório `/etc/adenoids`. Tais parâmetros podem ser mudados alterando valores pré-definidos em arquivos de configuração constantes na codificação de módulo específico.

5.3 O módulo ADDS: a Fonte de Informação

Para facilitar a implementação e melhorar o desempenho do protótipo, o módulo ADDS foi desenvolvido através do uso de diversas partes separadas, cada uma delas acoplada diretamente ao módulo requisitante da informação. ADDS captura e fornece informação para os seguintes módulos:

- ADEID: chamadas ao sistema incluindo os argumentos. Esses dados são obtidos através de alterações nos procedimentos que implementam essas chamadas;
- ADBID: tráfego de rede para cada serviço monitorado. Tal tráfego é capturado através da biblioteca `pcap`;
- ADSIG: tráfego de rede para serviço que está envolvido em um processo de extração de assinatura. Esse tráfego também é capturado através da biblioteca `pcap`.

No caso do módulo ADEID a informação é descartada assim que é analisada. O módulo ADBID trabalha considerando o esquema de *buffer* proporcionado pela biblioteca pcap. Já o módulo ADSIG utiliza em sua análise informações capturadas através da biblioteca pcap e posteriormente armazenadas em arquivos.

ADDS não fornece informação para o módulo ADFSR uma vez que a implementação de ADFSR é restrita à ativação de funcionalidades da ferramenta UNDOFS.

5.4 O módulo ADEID: o Detector Baseado em Evidências

O módulo ADEID realiza a detecção baseada em evidências para identificar quaisquer intrusões que introduzam alterações em aplicações, envolvendo ou não um *buffer overflow*. Para tanto, ADEID monitora processos em busca de violações em políticas de acesso pré-especificadas no arquivo `policies.conf`. Cada política de acesso especifica o comportamento que determinados processos podem exibir durante a execução. Esse comportamento é verificado observando os seguintes eventos sobre os objetos analisados:

- Arquivos, diretórios e *links*: abertura, criação, remoção, truncamento e alteração de atributos (*pathname*, proprietário, grupo e permissões);
- Processos: criação e execução;
- Módulos de *kernel*: criação e remoção;
- Comunicação: envio de sinais para processos, estabelecimento e aceitação de conexões TCP, e envio e recebimento de tráfego UDP.

O formato para a especificação de cada política de acesso é o seguinte:

```
nome_da_política[/caminho/completo/do/programa]
{
    fs_acl
    {
        lista de caminhos com tipos de acesso
    }

    can_exec
    {
        lista de programas que podem ser executados
    }
```

```
    max_children = número máximo de processos filhos

    can_send_signal = yes | no

    can_manip_modules
    {
        lista de módulos que podem ser criados e posteriormente
        removidos
    }

    connect_using_tcp = yes | no

    send_using_udp = yes | no

    accept_conn_on_ports
    {
        lista de faixas de portas que podem ser usadas para aceitar
        conexões TCP ou aceitar tráfego UDP
    }
} # fim da política
```

Apesar de parecer custosa a especificação manual de políticas de acesso para processos, a prática tem mostrado o contrário. Por exemplo, a construção de uma política inicial para um servidor *web* Apache leva em torno de 30 minutos. Após essa especificação inicial, é necessário observar as mensagens de alerta emitidas no arquivo de *log* `adeid` e fazer os ajustes finais necessários. Em geral, devem ser observadas as mensagens emitidas em um período que envolva inclusive reinicializações da aplicação monitorada.

Para construir uma boa política é necessário conhecer a hierarquia de diretórios de um sistema Linux típico[2]. Essa hierarquia especifica regiões do sistema de arquivos que contém arquivos relacionados e, em muitos casos, arquivos que raramente são modificados. Utilizando essa idéia é possível identificar inúmeras ações de atacantes que procuram corromper arquivos binários ou de configuração.

O módulo ADEID é implementado como um *patch* para o *kernel*. Algumas funções que implementam chamadas ao sistema — aquelas relacionadas à detecção — foram modificadas para executar os procedimentos de detecção desenvolvidos.

As políticas de acesso são lidas do disco e carregadas para a memória durante a inicialização do sistema operacional. Para fins de teste também pode ser utilizada uma

[2]Mais informações sobre a hierarquia de diretórios podem ser obtidas através da *man page* `hier`.

nova chamada ao sistema criada para atualizar políticas após a inicialização.

Quando um novo programa é executado através da chamada ao sistema `sys_execve()`, é feita uma busca por uma política de acesso relativa a esse programa. Se existe uma política especificada no arquivo `policies.conf` para esse programa, tal política é anexada ao processo em questão. Para cada processo monitorado são analisadas todas as chamadas ao sistema relevantes para a detecção. Como ADEID realiza uma detecção baseada em evidências de intrusão, são analisadas somente as chamadas ao sistema que foram executadas com sucesso.

Todo processo-filho criado a partir de um processo-pai monitorado por ADEID também será monitorado. Nesse caso o monitoramento desse processo-filho irá ocorrer segundo:

- Uma política pré-especificada para o processo-filho, caso ela exista;
- A política do processo-pai, caso não exista uma política para o processo-filho.

Desse modo é possível evitar que atacantes venham burlar facilmente a detecção de ADEID através da criação de processos filhos. A resolução de *pathnames* absolutos e o tratamento de *links* simbólicos e *hard links* também ajuda a evitar o sucesso de técnicas de evasão aplicadas por atacantes.

Resultados preliminares indicam que o impacto no desempenho decorrente da detecção de ADEID é bastante pequeno. Utilizando diariamente esse módulo nota-se que esse impacto chega a ser imperceptível para um usuário doméstico. Também foi efetuado um conjunto de testes preliminares para verificar a penalidade média imposta por ADEID, cujos resultados são apresentados na Tabela 5.3.

Operação analisada	**Penalidade média (%)**
Inicialização e *halt* do sistema (máquina virtual)	0,52
Abertura e leitura de arquivos	4,32
Criação e escrita de arquivos	0,09
Acesso a arquivos via servidor *web* Apache 2.0.40	1,78

Tabela 5.3: Penalidade de desempenho média imposta por ADEID, considerando o tempo de execução total de processos. Os testes que acessam arquivos utilizaram um conjunto de 800 arquivos distribuídos uniformemente em 8 categorias de tamanho: 100, 1000, 5000, 10000, 50000, 100000, 500000 e 1000000 bytes.

O pior resultado obtido — abrindo e lendo arquivos — apresenta um custo médio de apenas 4,32%, em relação ao desempenho de um *kernel* sem ADEID[3]. As políticas

[3]ADEID foi o único módulo que possuiu uma versão para um *kernel* real, utilizado nos testes com acesso a arquivos. Os demais módulos foram desenvolvidos somente para um *kernel* de máquina virtual.

construídas também demonstram ser eficientes para identificar intrusões, possibilitando uma detecção livre de falso-negativos e falso-positivos no ambiente analisado.

Após a identificação de uma intrusão, ADEID ativa o módulo ADIRA chamando um procedimento dentro do *kernel* e, após isso, um sinal `SIGUSR1` é enviado para ADSIG. ADSIG também recebe algumas informações a respeito da intrusão, incluindo o processo relacionado e a política violada. Para propósitos de teste um usuário pode desabilitar esses mecanismos de ativação.

5.5 O módulo ADIRA: o Agente de Resposta Inata

O módulo ADIRA é implementado como um *patch* para o *kernel* e trabalha restaurando o sistema computacional após uma intrusão. Esse módulo não implementa as respostas reversíveis previstas pelo Agente de Resposta Inata e, conseqüentemente, é ativado somente por ADEID. Embora este trabalho não tenha contemplado outros mecanismos de resposta automática tal assunto foi amplamente discutido e verificado em um trabalho paralelo [Fer03].

ADIRA restaura o sistema computacional através dos seguintes passos:

1. Bloqueie todos os processos de usuário;
2. Restaure o sistema de arquivos até um ponto de checagem utilizando a ferramenta UNDOFS;
3. Reinicie as aplicações monitoradas;
4. Termine o processo relacionado com a intrusão e seus descendentes;
5. Desbloqueie todos os processos bloqueados.

O Passo 1 é realizado para impedir que algum outro processo execute concorrentemente com o processo de restauração. O Passo 2 invoca o sistema de restauração de arquivos de forma a eliminar principalmente as modificações em binários e arquivos de configuração introduzidas durante a intrusão. Durante o Passo 3 é realizada a reinicialização de todas as aplicações que estão sendo monitoradas por ADENOIDS. Os processos envolvidos na intrusão são eliminados no Passo 4 e, ao final, todos os demais processos de usuário são desbloqueados.

A restauração do sistema de arquivos pode afetar a continuidade da execução de alguns programas, embora tal fato não tenha sido notado durante os testes. Entretanto, a restauração do sistema de arquivos pode contemplar somente parte da árvore de diretórios. Nesse caso, alguns dados relevantes para aplicações de usuário podem não ser incluídas na restauração, eliminando a interferência na execução dessas aplicações.

5.6 O módulo ADBID: o Detector Baseado em Comportamento

O módulo ADBID realiza uma detecção baseada em comportamento, analisando o tráfego de rede entrante em busca de requisições[4] que indiquem um comportamento não usual. Esse módulo analisa o tráfego relacionado com todas as aplicações monitoradas pelo módulo ADEID. ADBID não realiza a ativação de uma resposta reversível, sendo utilizado exclusivamente para a obtenção de assinaturas de ataque candidatas.

As assinaturas candidatas encontradas por ADBID são exatamente as requisições não usuais identificadas por esse módulo em um período de tempo requisitado. Sendo assim, o comportamento não usual identificado por ADBID deve conter requisições que são mais prováveis de ocorrerem em ataques *buffer overflow* e menos prováveis de serem encontradas na ausência desses ataques. Como visto na Seção 2.5.1, um ataque *buffer overflow* consiste em exceder o limite de um *buffer* e poderá obter sucesso somente quando a informação manipulada possui um tamanho além do esperado. Considerando esses fatos ADBID identifica como não usual aquelas requisições cujo tamanho (número de bytes) está acima do tamanho observado até o momento.

ADBID realiza uma estatística individual a respeito do tamanho das requisições capturadas para cada protocolo de aplicação analisado. De acordo com a configuração padrão, a cada 10000 requisições capturadas é recalculada a média μ e o desvio-padrão δ do tamanho das requisições. Os novos valores para μ e δ são obtidos através de uma média ponderada entre os valores antigos e os valores calculados para as últimas 10000 requisições.

Após decodificar uma requisição ADBID verifica se essa requisição é usual ou não comparando o seu tamanho l com os parâmetros μ e δ. De acordo com a configuração padrão, essa requisição é considerada não usual quando $l > \mu + 2\delta$. Essa classificação tende a descartar uma grande quantidade de requisições que não constituem um *overflow*[5], identificando os verdadeiro-positivos e alguns falso-positivos.

Esse módulo escreve seqüencialmente em um arquivo todas as requisições não usuais, organizadas de acordo com a estrutura `adbid_trequest` mostrada a seguir.

```
struct adbid_trequest
{
    struct timeval time;
    int dst_port;
```

[4]O termo "requisição" foi adotado para referenciar os dados recebidos no nível dos protocolos de aplicação. Esses dados são obtidos a partir da captura e decodificação do tráfego TCP/IP.

[5]Por exemplo, se o tamanho das requisições está uniformemente distribuído em um intervalo, essa classificação descarta mais de 97% das requisições analisadas [GD79].

```
    char* data;
    int len;
};
```

Nessa estrutura o campo `time` indica o momento em que a requisição foi identificada como não usual, o campo `dst_port` contém a porta de destino utilizada pela requisição, o campo `data` referencia o conjunto de bytes que foi capturado e o campo `len` armazena a quantidade de bytes dessa requisição.

O arquivo escrito por ADBID é consultado pelo módulo ADSIG para buscar as assinaturas candidatas dentro de um intervalo de tempo em questão.

5.7 O módulo ADSIG: o Extrator de Assinatura

O módulo ADSIG é responsável por extrair assinaturas de ataque analisando as assinaturas candidatas de ADBID que precedem o momento da identificação da intrusão por ADEID.

ADSIG trabalha exatamente como proposto na Seção 4.3.8. Uma vez ativado por ADEID, ADSIG lê as informações a respeito da intrusão — política violada e processo relacionado — e insere essas informações em uma fila FIFO. Quando o processo de extração de assinatura atual termina, caso haja algum, ADSIG lê as próximas informações na fila e inicia um novo processo de extração. O Algoritmo 2 mostra em detalhes como o processo de extração de assinatura ocorre. Esse algoritmo é uma adaptação direta da abordagem genérica proposta pelo Algoritmo 1 que é descrito na Seção 4.3.8.

O Algoritmo 2 começa a funcionar somente após a restauração do sistema computacional realizada por ADIRA. De acordo com esse algoritmo as assinaturas candidatas utilizadas por ADSIG correspondem às requisições não usuais ocorridas nas últimas 24 horas (Passo 1). Essas requisições estão ainda relacionadas com o processo que violou alguma política de ADEID. Por exemplo, se aplicação servidora `wu-ftpd` gera evidências de intrusão, então somente o tráfego não usual de comandos FTP será considerado. Cada requisição entrante capturada por ADDS após a restauração do sistema (Passo 4) tem o seu tamanho comparado com o tamanho das assinaturas candidatas, sendo descartadas aquelas candidatas menores que essa requisição entrante (Passo 6). Essa eliminação funciona bem para ataques *buffer overflow* porque se uma requisição é legítima — e provavelmente não ultrapassa os limites de um *buffer* — uma candidata cujo tamanho é no máximo igual ao tamanho dessa requisição provavelmente não irá ultrapassar esse *buffer* também[6]. Dessa forma, essa assinatura candidata pode ser considerada como um ato legítimo no

[6]Uma análise mais completa poderia considerar um tamanho diferente de *buffer* para cada tipo de requisição de um determinado protocolo de aplicação.

Algoritmo 2: Extração de assinaturas de ataque do módulo ADSIG.
Entrada: Um número real $p \in]0; 1]$, um conjunto E de todos os eventos gerados nas últimas 24 horas que precedem a detecção de ADEID, um conjunto N de eventos gerados no sistema computacional após a restauração de ADIRA, e $iinfo$, que contém as informações a respeito da intrusão identificada.
Saída: Um conjunto $C \subseteq E$ de eventos, que são as assinaturas de ataque extraídas. Essas assinaturas possuem probabilidade estimada inferior a p de que haverá falso-positivos utilizando-as conjuntamente para a identificação de ataques.
EXTRAIASSINATURAS(E, N, p, $iinfo$)
(1) Construa o conjunto $C \subseteq E$ coletando as requisições não usuais identificadas por ADBID nas últimas 24 horas para o tráfego de rede relacionado com $iinfo$;
(2) $progresso \leftarrow 0$;
(3) **enquanto** $progresso < \left\lceil \frac{|C|}{p} \right\rceil$ **faça**
(4) Leia uma nova requisição entrante $n \in N$ capturada por ADDS;
(5) **para cada** $c_i \in C$ **faça**
(6) **se** tamanho(c_i) < tamanho(n) **então** $C \leftarrow C \setminus \{c_i\}$;
(7) $progresso \leftarrow progresso + 1$;
(8) **retorne** cada requisição em C ou nulo se $|C| = 0$;

sistema e, portanto, deve ser descartada. Ao final do processo de extração de assinatura, ADSIG grava as assinaturas restantes em um arquivo de saída. Essas assinaturas não estão formatadas de acordo com algum IDS em particular mas essa etapa de tradução pode ser realizada facilmente por um procedimento auxiliar.

A implementação corrente desse módulo não explora a vantagem da ocorrência de ataques subseqüentes e o processo de extração de assinatura é reiniciado quando novas evidências de intrusão são encontradas.

O valor padrão para o parâmetro p é $0,0001$. Isso significa que a utilização dessas assinaturas juntas na detecção irá gerar provavelmente menos que um falso-positivo a cada dez mil requisições analisadas no futuro. É claro que falso-positivos irão ocorrer somente quando ADSIG extrair uma requisição válida como uma assinatura de ataque maturada.

5.8 O módulo ADFSR: o Repositório para Suporte Forense

O módulo ADFSR implementa uma interface para prover *redo* passo-a-passo no sistema de arquivos, invocando a ferramenta UNDOFS após uma reinicialização do sistema. Desse modo é possível verificar manualmente ou através de ferramentas automatizadas as alterações ocorridas no sistema de arquivos. Uma análise forense como essa pode ser de especial interesse para um administrador que deseja observar ações de atacantes ou conferir o funcionamento exibido por ADENOIDS.

Arquivos de *log* de outros módulos também podem auxiliar numa tarefa forense, incluindo:

- Resultados do monitoramento de ADEID;
- Requisições não usuais identificadas por ADBID;
- Assinaturas de ataque extraídas por ADSIG.

O correlacionamento dessas informações pode possibilitar inclusive a realização de ajustes na configuração de ADENOIDS visando uma melhor atuação no ambiente considerado.

5.9 A ferramenta UNDOFS

A ferramenta UNDOFS implementa um mecanismo genérico para prover restauração no sistema de arquivos aplicando técnicas de *undo*. Embora essa ferramenta não seja considerada um módulo de ADENOIDS, ela é essencial na execução das tarefas dos módulos ADIRA e ADFSR, merecendo também um destaque.

UNDOFS também é desenvolvido como um *patch* para o *kernel* e incorpora as funcionalidades de *undo* e *redo* em qualquer sistema de arquivos que possa suportar tanto a leitura quanto a escrita de dados.

Esse mecanismo é ativado antes que as seguintes operações sobre arquivos, diretórios e *links* possam ser realizadas: criação, remoção, escrita, truncamento e mudança de atributos (*pathname*, proprietário, grupo e permissões). Para cada operação completada com sucesso um *log* de *undo* é criado. Esse *log* mantém a informação necessária a respeito de cada operação, de modo que tais operações possam ser desfeitas no futuro. *Logs* de *redo* são criados por operações realizadas durante um processo de *undo*.

Um procedimento dentro do *kernel* pode ser chamado para requisitar a realização de um processo de *undo* ou *redo* até um ponto de checagem inserido na seqüência de *logs* armazenados. Os pontos de checagem são inseridos automaticamente durante a inicialização

do sistema operacional. Dessa forma um *undo* até o ponto de checagem irá restaurar o sistema de arquivos para o estado em que se encontrava no momento da última inicialização.

O arquivo de configuração `undofs.conf` especifica quais diretórios são cobertos por esse mecanismo. Isso significa que somente os arquivos que estão contidos sob os diretórios especificados implicam na geração de *logs* e, portanto, podem ser restaurados. A configuração padrão de UNDOFS inclui os diretórios `/bin`, `/boot`, `/dev`, `/etc`, `/initrd`, `/lib`, `/sbin`, `/usr` e `/var/named`. Esses diretórios contêm os mais importantes binários e arquivos de configuração necessários para um bom funcionamento do sistema computacional.

O desempenho da ferramenta UNDOFS depende diretamente da operação que está sendo realizada. Escrever dados no final de um arquivo irá adicionar apenas um *log* de tamanho fixo. O truncamento de um arquivo requer a leitura dos bytes a serem truncados e a escrita deles no arquivo de *log*. A remoção de arquivos e a operação de escrita sobre outros dados também são bastante caras. A mudança de atributos gera apenas um *log* de tamanho fixo que contém os atributos antigos.

Enquanto o uso de UNDOFS para contemplar todo o sistema de arquivos pode ser custoso, sua aplicação em diretórios vitais é viável. Tendo em vista que os principais binários e arquivos de configuração de um sistema GNU/Linux raramente são modificados, então raramente serão gerados *logs* para esses arquivos. Dessa forma o custo imposto é devido a verificações simples e tende a ser bastante aceitável.

5.10 Auto-proteção

ADENOIDS implementa dentro do *kernel* um mecanismo simples de auto-proteção. Esse mecanismo consiste em negar o acesso aos módulos de ADENOIDS, à ferramenta UNDOFS e aos arquivos de configuração de ADENOIDS e UNDOFS a partir dos processos que estão sendo monitorados. Dessa forma, o projeto do protótipo considera que todo possível alvo de ataque — usualmente aplicações servidoras — deve ser monitorado.

As operações controladas pelo mecanismo de auto-proteção incluem o acesso a arquivos, o envio de sinais a processos relativos aos módulos e a execução de chamadas ao sistema exclusivas de ADENOIDS.

5.11 Exemplo de funcionamento

Esta seção mostra o funcionamento geral do protótipo ADENOIDS através de um exemplo prático que inclui um ataque à aplicação monitorada `named`.

Durante a inicialização do sistema operacional as políticas de acesso para o módulo ADEID são lidas com sucesso e uma mensagem de sucesso é impressa na tela:

```
adeid: Loading policies... Ok
```

Em especial, a política de monitoramento para processos que executam o programa `/usr/sbin/named` é a seguinte:

```
named[/usr/sbin/named]
{
    fs_acl
    {
        /                   n,
        /etc/ld.so.cache    r,
        /etc/localtime      r,
        /etc/named.conf     r,
        /etc/protocols      r,
        /etc/resolv.conf    r,
        /etc/services       r,
        /lib/libc-2.1.3.so  r,
        /var/named          r,
        /var/run/named.pid  w,
        /var/run/ndc        w,
        /dev/null           w
    }

    can_exec{}
    max_children = 1
    can_send_signal = no
    can_manip_modules{}
    connect_using_tcp = no
    send_using_udp = yes
    accept_conn_on_ports{0, 53}
}
```

Durante a inicialização do sistema dois programas executando `/usr/sbin/named` são disparados, sendo que o primeiro processo cria um processo-filho e termina a execução em seguida. De acordo com registros no arquivo de *log* de ADEID o processo-pai tem identificação 335 e o processo-filho tem identificação 336:

```
---New log session opened---
 Monitoring process 335 [/usr/sbin/named] according to policy named
 Monitoring process 336 [/usr/sbin/named] according to policy named
 End monitoring process 335 [/usr/sbin/named] according to policy named
```

O processo de identificação 336 será monitorado durante toda a sua existência. Esse processo recebe 10000 requisições para resolução de nome e, dentre essas requisições, ADBID identifica 23 requisições não usuais distintas. Após serem processadas as 10000 requisições um atacante executa um *exploit* que contém um ataque *buffer overflow* contra a aplicação `/usr/sbin/named`:

```
[root@timo named]# ./linx86_bind 10.1.1.52 -e
copyright LAST STAGE OF DELIRIUM feb 2001 poland  //lsd-pl.net/
bind 8.2 8.2.1 8.2.2 8.2.2PX for slackware 4.0/redhat 6.2 x86

stack dump:
42 24 08 08 02 00 91 f3 0a 01 01 32 00 00 00 00
00 00 00 00 78 fc ff bf d6 58 08 08 90 3f 0d 08
d0 54 11 40 16 00 00 00 01 00 00 00 90 3f 0d 08
05 00 00 00 e0 e7 0b 08 16 00 00 00 01 00 00 00
a0 e0 05 08 d0 54 11 40 94 fc ff bf 60 e9 0c 08
00 00 00 00 98 fd ff bf 98 fd ff bf 61 d6 05 08
90 3f 0d 08 ec 26 11 40 ec b1 10 40 60 ae 00 40
e4 fd ff bf ec 26 11 40

frame ptr=0xbffffc00 adr=bffffa2e ofs=134 port=8684 connected! sent!
```

ADEID identifica a intrusão e ativa os mecanismos de restauração do sistema através de ADIRA. Uma verificada no arquivo de *log* de ADEID revela o seguinte:

```
*Policy named violated by the process 336 [/usr/sbin/named]* --->
 executing program /bin/bash
*** *** *** >>>
*Policy /usr/sbin/named had exceeded acceptable violation level (10)* --->
 Attack in progress!
<<< *** *** ***
```

Nesse caso, o *exploit* usado pelo atacante carrega o programa `/bin/bash` no processo executando `/usr/sbin/named`, causando a identificação da intrusão por ADEID. O sistema é restaurado por ADIRA e o atacante perde a conexão. ADEID ativa o módulo ADSIG para iniciar o processo de extração de assinatura. ADSIG começa o processo de

extração com 24 assinaturas candidatas. Essas assinaturas compreendem as 23 requisições não usuais identificadas anteriormente por ADBID junto com a requisição contendo o *overflow*[7], também identificada como não usual:

```
<Begin of signature generation>
Number of candidates: 24
  0:[ 200-215-114-006 fnsce7001 dsl brasiltelecom net br]
  1:[ 200-180-188-189 paemt7005 dsl brasiltelecom net br]
  2:[ 206 196 158 200 ipwhois rfc-ignorant org]
  3:[ 157 161 136 200 ipwhois rfc-ignorant org]
  4:[ excalibur las ic unicamp br las ic unicamp br]
  5:[ quasar las ic unicamp br las ic unicamp br]
  6:[ 233 248 171 200 ipwhois rfc-ignorant org]
  7:[ 200-181-081-134 bsace7002 dsl brasiltelecom net br]
  8:[ 200-180-174-036 paemt7005 dsl brasiltelecom net br]
  9:[ 140 247 180 200 ipwhois rfc-ignorant org]
 10:[ 211 131 136 216 ipwhois rfc-ignorant org]
 11:[ 181 161 136 200 ipwhois rfc-ignorant org]
 12:[ ip-170-180-134 xdsl-dinamico ctbcnetsuper com br]
 13:[ 205 131 136 216 ipwhois rfc-ignorant org]
 14:[ 150 158 158 200 ipwhois rfc-ignorant org]
 15:[ 114 246 171 200 ipwhois rfc-ignorant org]
 16:[ 214 145 174 200 ipwhois rfc-ignorant org]
 17:[ 200-203-067-227 smace7002 dsl brasiltelecom net br]
 18:[ sunsdlc1-19-vhost2 mtvwca1-dc1 genuity net]
 19:[ sunsdlc1-23-vhost1 cartnj1-dc1 genuity net]
 20:[ 215 226 136 216 ipwhois rfc-ignorant org]
 21:[ 200-181-082-038 bsace7002 dsl brasiltelecom net br]
 22:[ ip-170-181-083 xdsl-dinamico ctbcnetsuper com br]
 23:[?....;1._..|.w..w..O..O.....1.....Q1..f......Y1.9.u.f...f9^.t...?
     1...1..?I..A....1.[.K....C..C.1.........../bin/sh..............@...@..
     ............@.......@.................?...@...A......?.............
     ..........@.....r...?......]
```

É importante observar que na requisição contendo o *overflow* — a última — consta o comando `/bin/sh` mas na detecção de ADEID consta a execução de `/bin/bash`. Porém, no sistema atacado verifica-se que o arquivo `/bin/sh` é um *link* simbólico para `/bin/bash`.

Durante o processo de extração de assinatura são utilizadas outras 10000 requisições normais para maturar as candidatas. O resultado desse processo é o seguinte:

[7]Cada caractere não imprimível dessa requisição é substituído por um caractere '.' a seguir.

```
  Signature generation state: progress = 0, number of candidates = 24
  Signature generation state: progress = 147, number of candidates = 13
  Signature generation state: progress = 227, number of candidates = 1
  Signature generation state: progress = 1000, number of candidates = 1
  Signature generation state: progress = 2000, number of candidates = 1
  Signature generation state: progress = 3000, number of candidates = 1
  Signature generation state: progress = 4000, number of candidates = 1
  Signature generation state: progress = 5000, number of candidates = 1
  Signature generation state: progress = 6000, number of candidates = 1
  Signature generation state: progress = 7000, number of candidates = 1
  Signature generation state: progress = 8000, number of candidates = 1
  Signature generation state: progress = 9000, number of candidates = 1
  Signature generation state: progress = 10000, number of candidates = 1
Number of candidates: 1
  0:[?....;1._..|.w..w..O..O.....1.....Q1..f......Y1.9.u.f...f9^.t...?
     1...1..?I..A....1.[.K....C..C.1.........../bin/sh..............@...@..
     ............@.......@.................?...@...A......?.............
     ..........@.....r...?......]
<End of signature generation>
```

Quando são analisadas as primeiras 147 requisições normais o número de candidatas cai para 13. Quando o número de requisições normais analisadas alcança 227 a quantidade de candidatas cai para 1. Essa assinatura é maturada até serem analisadas 10000 requisições devido à escolha do parâmetro $p = 0,0001$. Ao final do processo de extração resta somente uma assinatura candidata, que é a requisição contendo o *overflow*. Essa assinatura é bastante específica, uma vez que é composta por mais de uma centena de bytes. A base de regras do IDS Snort possui uma assinatura que contempla os ataques identificados pela assinatura extraída por ADENOIDS. Entretanto, a assinatura especificada no IDS Snort é mais genérica, contendo apenas 17 bytes.

5.11.1 Momento da análise

O módulo ADEID executa sua análise verificando chamadas ao sistema em tempo real. Dessa forma é possível evitar grandes danos no sistema computacional porque a detecção ocorre logo após a ocorrência de uma evidência clara. Já o módulo ADBID realiza a detecção de acordo com os esquemas de captura e *buffers* encontrados na biblioteca `pcap`.

5.12 Conclusão

Este capítulo fez uma descrição sucinta de um protótipo construído com base na arquitetura de segurança proposta. Esse protótipo, ADENOIDS, não implementa todos os componentes e funcionalidades encontrados nessa arquitetura mas procura fazer uma validação das principais idéias propostas.

A implementação de cada módulo é mostrada sem fazer um aprofundamento nas técnicas de implementação utilizadas. Dessa forma, espera-se contribuir para o entendimento geral de ADENOIDS sem esbarrar na colocação de detalhes de codificação. Após a descrição desses módulos é apresentado um exemplo prático do funcionamento do protótipo enfocando as etapas de detecção e extração de assinatura.

O próximo capítulo mostra uma seqüência de testes e resultados obtidos com o funcionamento de ADENOIDS sob ataques. Esses testes atentam principalmente para a capacidade de ADENOIDS identificar intrusões e extrair assinaturas de ataque. Analisando os resultados obtidos é possível verificar as várias virtudes da arquitetura proposta, as quais foram discutidas no Capítulo 4.

Capítulo 6

Testes e resultados experimentais

Foram realizados alguns testes com o protótipo ADENOIDS visando a obtenção de resultados experimentais. ADENOIDS foi implementado majoritariamente sobre um ambiente de máquina virtual, o que inviabilizou uma análise aprofundada do seu desempenho. Os testes realizados tiveram por objetivo principal verificar a capacidade da detecção baseada em evidências e do mecanismo de extração de assinaturas. Analisando os resultados coletados, foi possível notar a precisão da detecção baseada em evidências e o sucesso obtido nas etapas de extração de assinatura.

Os testes foram realizados utilizando dois conjuntos de dados. O primeiro conjunto foi extraído do 1999 DARPA Intrusion Detection Evaluation. O segundo conjunto foi coletado no Laboratório de Administração e Segurança de Sistemas (LAS) do IC/UNICAMP. Esses dados foram, em seguida, submetidos ao sistema-alvo executando ADENOIDS. Esse sistema-alvo corresponde ao mesmo ambiente de máquina virtual utilizado no desenvolvimento do protótipo, conforme descrito na Seção 5.1.1.

Este capítulo apresenta os testes e resultados experimentais como segue. A Seção 6.1 faz algumas observações gerais sobre os testes realizados. A Seção 6.2 apresenta os testes e resultados envolvendo a detecção baseada em evidências. Ao final, a Seção 6.3 apresenta os testes e resultados experimentais envolvendo o processo de extração de assinaturas de ataques.

6.1 Observações gerais

O sistema-alvo utilizado nos testes é baseado na instalação do Red Hat Linux 6.2 executando o *kernel* virtual 2.4.19. Esse sistema foi ajustado para prover as seguintes aplicações vulneráveis a ataques *buffer overflow*:

- `named`: servidor de nomes (DNS);

- `wu-ftpd`: servidor de arquivos (FTP);
- `imapd`: servidor de acesso remoto a *e-mail* (IMAP);
- `amd`: *daemon* para montagem automática de sistema de arquivos.

Cada uma dessas aplicações pode ser atacada com sucesso através da utilização de *exploits* disponíveis na Internet, o que possibilitou a simulação da ocorrência de intrusões no sistema-alvo.

Os módulos ADEID e ADFSR e, conseqüentemente, a ferramenta UNDOFS, foram testados durante um período superior a dois meses, considerando o uso de cada uma das aplicações vulneráveis. Já os módulos ADIRA, ADBID e ADSIG foram testados intensivamente durante um período de duas semanas, considerando as aplicações `named` e `wu-ftpd`[1]. Essas duas semanas de teste utilizaram dados coletados por mais de dois meses.

6.2 Detecção baseada em evidências

Cada política de monitoramento do módulo ADEID foi construída seguindo dois passos e observando as violações reportadas:

1. Estabelecimento inicial da política: esse passo ocupou aproximadamente 30 minutos de trabalho intensivo. Após essa etapa, a política já foi submetida aos principais eventos da aplicação monitorada;
2. Refinamento da política: esse passo gastou cerca de dois dias de trabalho bastante esparso. Essa etapa busca contemplar eventos mais raros — por exemplo, inicialização e finalização da aplicação e situações de erro — que não foram observados durante a criação da política inicial.

ADEID relaciona cada violação encontrada com a respectiva política em uso. Utilizando essa facilidade é possível construir diversas políticas de acesso em paralelo.

6.2.1 Resultados obtidos

Após a elaboração das políticas de monitoramento, o módulo ADEID demonstrou ser bastante eficiente para identificar ataques. Ele mostrou ser livre de falso-positivos e falso-negativos durante os testes. Em outras palavras, ADEID identificou positivamente todos os ataques contra as aplicações monitoradas e não realizou a identificação de ações

[1] Foi analisada apenas a conexão de controle do tráfego FTP.

legítimas. Devido a essa precisão na detecção, ADEID pôde ativar os módulos ADIRA e ADSIG nos momentos oportunos.

A ferramenta UNDOFS cumpriu corretamente as tarefas de restauração, mesmo quando foi utilizada para reverter situações onde foram executadas ações bastante destrutivas como `rm -rf /etc`. Fazendo uso da ferramenta UNDOFS, o módulo ADFSR também pôde refazer as ações praticadas durante as intrusões.

Durante os testes, o módulo ADIRA foi capaz de realizar sua tarefa de restauração invocando UNDOFS. Dessa forma, após cada intrusão o sistema de arquivos foi revertido para o estado verificado durante a última inicialização do sistema. O término de processos envolvidos na intrusão realizado por ADIRA também fez com que os atacantes perdessem a comunicação com a máquina invadida.

Os resultados experimentais envolvendo os módulos ADBID e ADSIG são detalhados separadamente na Seção 6.3.

6.3 Extração de assinatura

Esta seção apresenta, a seguir, os testes e resultados experimentais obtidos através da utilização dos conjuntos de dados DARPA e LAS, respectivamente.

6.3.1 Conjunto de dados DARPA

O conjunto de dados 1999 DARPA Intrusion Detection Evaluation é composto por informações incluindo tráfego de rede e *log* de eventos [Lab99]. Inicialmente ADENOIDS foi testado somente contra ataques *buffer overflow* à aplicação `named`, uma vez que não havia dados de treinamento para a aplicação `imapd` e o *daemon* vulnerável para `sendmail` não estava disponível. Para compensar isso, alguns ataques *buffer overflow* foram inseridos no tráfego para a aplicação `wu-ftpd`. Como ADENOIDS foi desenvolvido para analisar eventos produzidos somente por um computador, os testes foram feitos considerando o tráfego de rede destinado para cada máquina separadamente.

6.3.2 Resultados obtidos

A Tabela 6.1 sumariza os resultados obtidos para esse conjunto de dados. A primeira coluna descreve a aplicação monitorada e a segunda coluna mostra a média de requisições recebidas por dia para a máquina em questão. A terceira coluna apresenta o número de requisições utilizadas para efetuar o treinamento do módulo ADBID de forma a obter valores para μ e δ. A quarta coluna mostra o número de requisições que foram recebidas antes do ataque, capturadas nas últimas 24 horas. Dentre essas requisições serão

escolhidas as candidatas. Cada teste foi realizado considerando um conjunto exclusivo de requisições anteriores ao ataque. A quinta coluna mostra o número de assinaturas candidatas, isto é, o número de requisições não usuais identificadas por ADBID nas últimas 24 horas. A sexta coluna apresenta o número de requisições legítimas exigido para completar o processo de extração de assinatura. Em alguns casos o resultado final de ADSIG pode ser conhecido utilizando menos de 1000 requisições legítimas. Entretanto, para satisfazer o valor do parâmetro p indicado entre parênteses, o processo de extração deve ser continuado. A sétima coluna indica o número de assinaturas extraídas por ADSIG ao final do processo de maturação. Alguns ataques podem apresentar mais de uma assinatura. A oitava coluna indica se a requisição contendo uma parte significativa do *overflow* foi encontrada por ADSIG. A última coluna mostra a quantidade de falso-positivos ocorridos após o processo de extração de assinatura, isto é, quantas assinaturas de ataque extraídas podem identificar ações legítimas.

Analisando os resultados obtidos é possível concluir que ADENOIDS foi bastante eficiente para descartar a ocorrência de falso-positivos. Para a aplicação `named` ADBID identificou 6, 11 e 8 requisições não usuais em cada dia, respectivamente. Dentre essas requisições sempre estava a requisição contendo o ataque. Se um IDS baseado em comportamento utilizasse essa técnica de detecção ele estaria gerando 5, 10 e 7 falso-positivos em cada dia[2]. Nesse caso, ADENOIDS foi capaz de extrair as assinaturas dos ataques sem a ocorrência de falso-positivos. Para a aplicação `wu-ftpd` ADBID foi capaz de identificar como candidatas todas as requisições voltadas ao *overflow* e algumas requisições legítimas. Após a etapa de maturação de ADSIG também foi possível eliminar a ocorrência de falso-positivos sem, entretanto, descartar qualquer requisição voltada ao *overflow*. Em 2 dentre 6 testes realizados com a aplicação `wu-ftpd` houve a extração de uma assinatura que não estava ligada aos ataques realizados.

As Figuras 6.1 e 6.2 mostram o progresso da extração de assinatura *versus* o número de assinaturas candidatas para os testes envolvendo as aplicações `named` e `wu-ftpd`, respectivamente.

Para a aplicação `named` as assinaturas finais foram conhecidas antes mesmo que o progresso atingisse 5000 requisições legítimas. Porém, para satisfazer p foram exigidas 10000 requisições durante o processo completo de extração. Uma análise do conjunto de dados DARPA revela, em uma média de quatro semanas, um tráfego de 174559 requisições DNS por dia. Fazendo uma comparação proporcional, as 10000 requisições exigidas em cada processo de extração poderiam ser capturadas em menos de uma 1 hora e 30 minutos.

Para a aplicação `wu-ftpd` as assinaturas finais foram encontradas antes que o progresso atingisse 1000 requisições legítimas, embora o processo completo tenha exigido de 3340

[2]Esses valores são obtidos ao verificar que o ataque feito à aplicação `named` utiliza somente uma requisição maliciosa.

Aplicação	Média de reqs/dia	# Reqs treino	# Reqs (24 h)	# Candidatas	# Reqs legítimas exigido	# Reqs extraídas	Encontrou?	# Falso-positivos
named	174559	50000	58942	6	10000 ($p = 0,0001$)	1	sim	0
named	174559	50000	267336	11	10000 ($p = 0,0001$)	1	sim	0
named	174559	50000	266995	8	10000 ($p = 0,0001$)	1	sim	0
wu-ftpd	1575	2000	1873	29	4342 ($p = 0,003$)	13	sim	1
wu-ftpd	1575	2000	1603	36	4008 ($p = 0,003$)	12	sim	0
wu-ftpd	827	2000	918	22	3340 ($p = 0,003$)	10	sim	0
wu-ftpd	827	2000	795	22	3674 ($p = 0,003$)	11	sim	1
wu-ftpd	761	2000	670	24	4008 ($p = 0,003$)	12	sim	0
wu-ftpd	761	2000	1097	38	4008 ($p = 0,003$)	12	sim	0

Tabela 6.1: Resultados experimentais para o conjunto de dados DARPA.

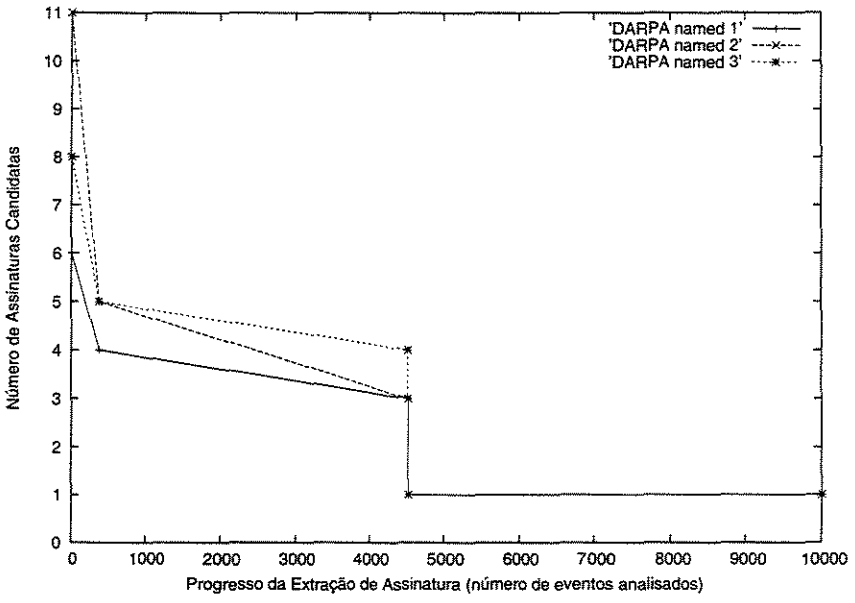

Figura 6.1: Progresso *versus* número de candidatas para named DARPA.

a 4342 requisições. Uma análise do conjunto de dados DARPA revela, em média, um tráfego de 1054 requisições de controle FTP por dia. Comparando proporcionalmente, as requisições exigidas durante cada processo de extração seriam capturadas em um período aproximado de 4 dias.

6.3.3 Conjunto de dados LAS

O conjunto de dados LAS é composto por *queries* DNS coletadas no servidor DNS externo do Laboratório de Administração e Segurança de Sistemas durante 43 dias. Esse conjunto de dados foi escolhido por dois fatores:

- A aplicação named é muito utilizada e freqüentemente vulnerável;
- *Queries* DNS podem ser reenviadas de uma maneira facilitada quando elas são carregadas pelo protocolo UDP.

Algumas informações adicionais a respeito do conjunto de dados LAS são encontradas na Tabela 6.2. Esses dados foram inicialmente analisados e foi constatada a ausência de

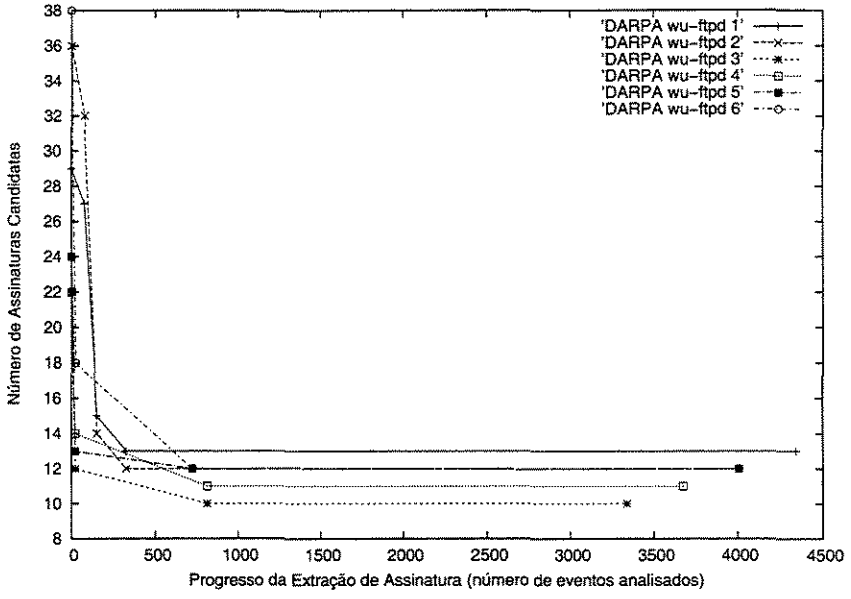

Figura 6.2: Progresso *versus* número de candidatas para `wu-ftpd` DARPA.

ataques.

Total de requisições capturadas	369397
Total de requisições distintas	33928
Média de requisições por dia	8590
Média de requisições distintas por dia	789

Tabela 6.2: Informações adicionais sobre o conjunto de dados LAS.

6.3.4 Resultados obtidos

Tendo como base esse conjunto de dados foram realizados 12 testes inserindo ataques entre as requisições. A Tabela 6.3 sumariza os resultados obtidos. Essa tabela está organizada de acordo com o mesmo formato apresentado pela Tabela 6.1. A única alteração nesses testes foi a utilização de um número fixo de 10000 requisições anteriores ao ataque, o que

excede a quantidade média de requisições capturadas por dia.

Analisando os resultados obtidos é possível concluir novamente que ADENOIDS foi bastante eficiente para descartar a ocorrência de falso-positivos. ADBID identificou de 7 a 25 requisições não usuais em cada porção de 10000 requisições considerada. Dentre essas requisições não usuais sempre estava a requisição contendo o ataque. Após a etapa de maturação de ADSIG foi possível eliminar a ocorrência de falso-positivos, sem descartar as requisições relacionadas ao ataque. Em 2 dentre 12 testes realizados houve a extração de uma assinatura que não estava ligada aos ataques.

As Figuras 6.3 e 6.4 mostram o progresso da extração de assinatura *versus* o número de assinaturas candidatas para os seguintes testes, respectivamente:

- Um conjunto de 8 testes onde o processo de extração de assinaturas consome 10000 requisições legítimas;
- Um conjunto de 4 testes onde são consumidas mais de 10000 requisições durante o processo de extração.

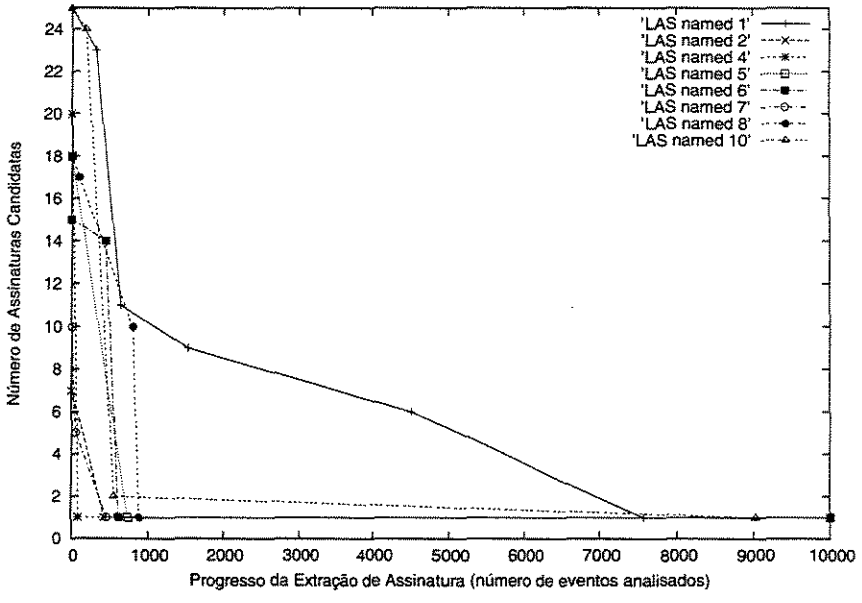

Figura 6.3: Progresso *versus* número de candidatas para 8 testes LAS.

Aplicação	Média de reqs/dia	# Reqs treino	# Reqs (24 h)	# Candidatas	# Reqs legítimas exigido	# Reqs extraídas	Encontrou?	# Falso-positivos
named	8590	40922	10000	25	10000 ($p = 0,0001$)	1	sim	0
named	8590	40922	10000	7	10000 ($p = 0,0001$)	1	sim	0
named	8590	40922	10000	14	21099 ($p = 0,0001$)	2	sim	1
named	8590	40922	10000	20	10000 ($p = 0,0001$)	1	sim	0
named	8590	40922	10000	18	10000 ($p = 0,0001$)	1	sim	0
named	8590	40922	10000	15	10000 ($p = 0,0001$)	1	sim	0
named	8590	40922	10000	10	10000 ($p = 0,0001$)	1	sim	0
named	8590	40922	10000	18	10000 ($p = 0,0001$)	1	sim	0
named	8590	40922	10000	20	20000 ($p = 0,0001$)	2	sim	1
named	8590	40922	10000	25	10000 ($p = 0,0001$)	1	sim	0
named	8590	40922	10000	20	11640 ($p = 0,0001$)	1	sim	0
named	8590	40922	10000	23	30955 ($p = 0,0001$)	1	sim	0

Tabela 6.3: Resultados experimentais para o conjunto de dados LAS.

Como pode ser notado na Figura 6.3, em 6 dentre os 8 testes do primeiro conjunto as assinaturas finais foram conhecidas antes que o progresso atingisse 1000 requisições legítimas. Para satisfazer p foi necessário processar 10000 requisições, o que excede moderadamente a média de 8590 requisições capturadas por dia. Dessa forma, cada processo de extração consumiria aproximadamente 1 dia e 4 horas para capturar as requisições legítimas.

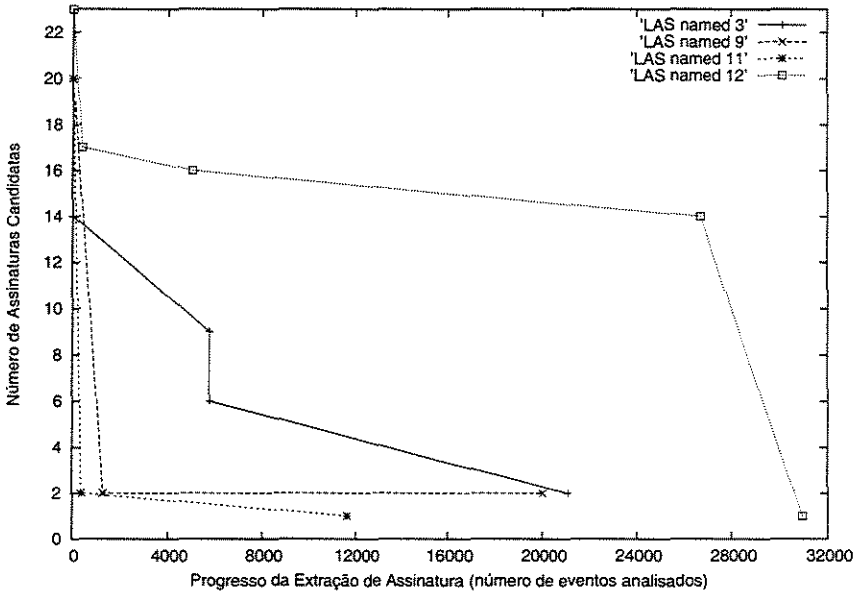

Figura 6.4: Progresso *versus* número de candidatas para 4 testes LAS.

Como pode ser observado na Figura 6.4, todos os 4 testes do segundo conjunto utilizam mais de 10000 requisições para completar o processo de extração de assinatura. Em 2 dentre esses 4 testes foram extraídas duas assinaturas finais, sendo que uma delas não estava relacionada ao ataque realizado. Como foram exigidas de 11640 a 30955 requisições para completar o processo de extração, tal processo poderia ter consumido de 1 dia e 9 horas a 3 dias e 15 horas, considerando a média de requisições por dia.

6.4 Conclusão

Este capítulo descreveu como ADENOIDS foi testado e quais resultados experimentais foram obtidos. Através dessa descrição é possível notar que alguns resultados importantes foram verificados, os quais incluem especialmente:

- A factibilidade e a precisão da detecção baseada em evidências de intrusão;
- A possibilidade de aproveitar tal precisão na análise automatizada de intrusões;
- A precisão do algoritmo proposto para encontrar as assinaturas de ataque.

Foram utilizados dois conjuntos de dados para teste. O primeiro conjunto, 1999 DARPA Intrusion Detection Evaluation, tem sido mundialmente utilizado para verificar metodologias para a identificação de intrusão. O segundo conjunto foi capturado durante o pleno funcionamento de um servidor em um laboratório de pesquisa que, justamente pelo ambiente de uso, está sujeito a um tráfego bastante diversificado.

O mecanismo de detecção baseada em evidências apresentou taxas de falso-positivos e falso-negativos iguais a zero nos testes realizados. Esse mecanismo foi desenvolvido para ataques a aplicações em geral, incluindo ataques *buffer overflow*. A restauração do sistema foi capaz de retornar o sistema computacional ao seu funcionamento normal, possibilitando eliminar possíveis *backdoors* e cavalos-de-tróia. Após essa restauração, ADENOIDS foi capaz de identificar precisa e automaticamente as assinaturas dos ataques consumados. Entretanto, o tempo necessário para completar um processo de extração pode ser da ordem de dias. Nesse caso, a arquitetura desenvolvida propõe também a utilização de eventos legítimos previamente coletados, o que pode contribuir na eliminação de um longo tempo de espera.

Os resultados aqui descritos também foram bem recebidos pela comunidade científica, tanto na área de computação evolutiva quanto na área de segurança em redes [dPdCdG04, dPdG04].

Capítulo 7

Conclusão

Este trabalho realizou um estudo sobre sistemas de detecção de intrusão, levantando as características e limitações existentes em cada técnica empregada. Também foi estudado o sistema imunológico humano de maneira a buscar inspiração para o desenvolvimento de uma arquitetura de segurança computacional. Ao invés de utilizar esse sistema biológico unicamente para o desenvolvimento de algoritmos de detecção, esta pesquisa buscou identificar nesse sistema algumas características relevantes e bastante úteis para a segurança computacional, tais como:

- Ocorrência de ataques com sucesso;
- Identificação por meio de danos;
- Detecção baseada em conhecimento prévio;
- Detecção baseada em comportamento;
- Respostas inespecífica e específica;
- Análise automatizada de agentes invasores e memória;
- Precisão em exposições futuras ao mesmo agente.

Utilizando tais características foi projetada uma arquitetura de segurança computacional com funcionalidades que vão além das encontradas em um IDS convencional, permitindo, inclusive, realizar o estudo de ataques e a extração de assinaturas de detecção.

Foi construído um protótipo baseado nessa arquitetura, ADENOIDS, que implementa as principais idéias apresentadas. Esse protótipo foi submetido a testes utilizando dois conjuntos de dados. Os resultados experimentais obtidos permitiram verificar que, considerando a ocorrência de ataques com sucesso, é possível identificar ataques desconhecidos

através da análise de evidências de intrusão. Uma vez identificado um ataque desconhecido, é possível extrair automaticamente uma assinatura para tal ataque, tornando-o conhecido. Com base nesses fatos, foi possível verificar as principais idéias apresentadas na arquitetura proposta e validar a hipótese deste trabalho.

7.1 Contribuições

Esta pesquisa traz consigo algumas contribuições, que são identificadas como segue:

- Descrição abrangente sobre sistemas de detecção de intrusão, apresentando as vantagens e desvantagens das abordagens existentes e incluindo uma revisão das pesquisas mais relevantes;
- Levantamento dos aspectos principais do sistema imunológico humano, de forma a identificar características relevantes no desenvolvimento de sistemas de segurança computacional. Embora tais características sejam conhecidas de longa data, as pesquisas anteriores concentravam-se em detalhes relevantes para a detecção baseada em comportamento, sem explorar com grande amplitude o sistema biológico;
- Uma definição para a detecção baseada em evidências de intrusão. Essa forma de identificação de ataques analisa somente as ações realizadas com sucesso. Por considerar cenários típicos de intrusão essa detecção facilita o processo de identificação. Sendo assim, essa abordagem pode ser bastante precisa, como mostram os resultados experimentais;
- Uma exploração da oportunidade trazida pela detecção baseada em evidências, possibilitando a alocação dinâmica de recursos para restauração do sistema e extração de assinatura;
- Especificação de um algoritmo para extração de assinaturas de ataque que permite estudar um ataque desconhecido e torná-lo conhecido. Esse algoritmo é aplicável em situações gerais e faz uso da localidade dos eventos do ataque em relação ao momento da identificação da intrusão. Resultados demonstram que esse algoritmo é bastante eficiente para eliminar falso-positivos;
- Desenvolvimento de uma arquitetura de segurança computacional utilizando as funcionalidades e princípios extraídos do sistema imunológico. Essa arquitetura permite que cada máquina mantenha uma base de conhecimento particular, sem armazenar informações irrelevantes e guardando somente as informações que dizem respeito a ataques com sucesso. Também são feitas analogias com o sistema imunológico,

algumas considerações práticas e um relacionamento com trabalhos anteriores. A arquitetura desenvolvida foi verificada através da construção de ADENOIDS, da realização de testes e da análise dos resultados obtidos.

Durante o desenvolvimento deste trabalho também foram sugeridas algumas linhas de pesquisa, resultando na publicação de um artigo em conferência internacional [CdG04].

7.2 Trabalhos futuros

Este trabalho apresentou uma arquitetura de segurança genérica e implementou ADENOIDS, que, apesar de possibilitar a verificação das características principais, não explora todas as possibilidades dessa arquitetura. Dessa forma, esta pesquisa abre um grande leque para a realização de trabalhos futuros, como os citados a seguir.

7.2.1 Generalização de ADENOIDS

O protótipo ADENOIDS foi implementado para efetuar a detecção baseada em evidências para ataques no nível de aplicação e extrair assinatura para ataques *buffer overflow*. Uma extensão deste trabalho consiste em implementar os mecanismos de detecção baseada em comportamento, extração de assinatura e geração de resposta para ataques no nível de aplicação em geral. Um estudo como esse deve trabalhar com os seguintes pontos essenciais:

- Detector Baseado em Comportamento: deve ser capaz de identificar comportamento não usual relacionado com uma variedade de ataques a fim de prover uma resposta reversível;

- Assinaturas candidatas: se forem obtidas através do Detector Baseado em Comportamento, elas exigirão uma identificação mais precisa desse componente, no sentido de que apenas uma taxa bastante baixa de falso-negativos poderá ocorrer. Entretanto, a ocorrência de falso-positivos nesse momento não irá causar um grande impacto no sistema de segurança. Em uma alternativa simplificada, as assinaturas candidatas também podem ser compostas por todos os eventos gerados antes da detecção de evidências, considerando um intervalo de tempo pré-especificado;

- Extrator de Assinatura: ADENOIDS utiliza uma comparação simples para descartar assinaturas que não compreendem um ataque *buffer overflow*. Essa comparação leva em conta o tamanho das requisições em questão e pode deixar de refletir outras classes de ataque. Uma extração de assinatura para ataques em geral deve utilizar

um mecanismo de comparação mais genérico como, por exemplo, um casamento aproximado de padrões. Outra estratégia poderia considerar vários critérios de análise de assinaturas, onde cada critério diz respeito a uma classe de ataques particular. Nesse caso, por exemplo, tamanho das requisições poderia estar relacionado a ataques *buffer overflow*, a ocorrência da *string* "%n" poderia estar relacionada com ataques *format string*, outros caracteres especiais como '|', ';', '&' poderiam estar relacionados a outros tipos de ataque envolvendo falhas na validação de entrada, etc.;

- Gerador de Resposta: deve ser considerada a existência de mais de um tipo de resposta para contemplar diferentes classes de ataque.

Para resolver cada um desses problemas também pode ser utilizada novamente a inspiração proveniente de sistemas biológicos, como a do sistema imunológico humano.

7.2.2 Levantamento de eventos legítimos

O algoritmo de extração proposto na Seção 4.3.8 faz uso de um conjunto de eventos legítimos gerados no sistema. Com base nesses eventos, tal algoritmo irá promover a eliminação de falso-positivos. Como já foi apresentado, esse conjunto de eventos pode ser obtido utilizando um repositório contendo eventos coletados previamente, coletando eventos após a restauração do sistema, ou através de uma combinação desses dois esquemas anteriores.

A utilização de um repositório tem a vantagem de possibilitar a continuidade do processo de extração de assinatura mesmo sob a presença de outras intrusões. Também, não é necessário esperar pela geração de eventos no sistema. Entretanto, é necessário especificar como esse repositório será construído e atualizado, bem como garantir que a quantidade de eventos armazenados é suficiente para concluir os processos de extração futuros. Sem dúvida, a construção de um conjunto contendo um alto nível de diversidade irá melhorar a capacidade de eliminação de falso-positivos.

A utilização de eventos coletados após a restauração do sistema não requer exigências como no caso anterior. Porém, é necessário esperar pela geração de eventos no sistema computacional e a ocorrência de ataques força uma reinicialização do processo de extração.

A combinação dessas duas estratégias também pode ser proveitosa, trazendo uma extração de assinatura rápida quando p e $|C|$ são adequados. Alguns processos de extração poderão necessitar de um conjunto complementar de eventos que pode ser obtido através da segunda estratégia.

Em todos os casos é possível, ainda, tomar vantagem da ocorrência de ataques subseqüentes, correlacionando seus eventos e até mesmo restringindo a quantidade de assinaturas candidatas.

Um estudo envolvendo essas alternativas sem dúvida trará resultados relevantes a respeito da extração automática de assinaturas de ataque. A consideração de ataques em geral pode ser vista basicamente como um pré-requisito para a obtenção de melhores resultados nesse estudo.

7.2.3 Generalização de assinaturas de ataque

Na fase de maturação de afinidade o sistema imunológico é capaz de criar uma espécie de memória contra o agente invasor. Dessa forma, no futuro ele identifica de um modo bastante preciso esse agente e outros agentes estruturalmente relacionados. Pensando no problema de detecção de intrusão esse processo compreenderia a extração de assinatura para:

- O ataque que deu origem à intrusão;
- Ataques que utilizam a mesma técnica ou que podem ser representados por uma assinatura bastante parecida.

Para o caso específico de ataques *buffer overflow* é possível encontrar um método para obter essa funcionalidade: quando a base de conhecimento passa a armazenar n assinaturas de ataque relacionadas a requisições de tamanho maior que l, então pode ser criada uma regra de detecção que identifica como ataque todas as requisições com tamanho superior a l.

Para ataques em geral, essa generalização pode ser bem mais complicada. Uma idéia consiste em verificar a possibilidade de utilizar uma detecção baseada em conhecimento com um determinado grau de aproximação, considerando o casamento parcial de assinaturas de ataque.

7.2.4 Ambiente para teste de sistemas de segurança

Sem dúvida a realização de testes com sistemas de segurança é uma etapa que exige bastante trabalho e consome uma grande parcela de tempo. Pensando nesse fato, um estudo voltado para a criação de um ambiente de testes facilitaria bastante o processo de desenvolvimento de sistemas de detecção e segurança em geral. Aspectos a serem trabalhados são:

- Construção de um sistema computacional vulnerável: busca propiciar um ambiente combinando os principais serviços, diferentes sistemas operacionais e vulnerabilidades. Tal ambiente deve considerar preferencialmente a utilização de uma estratégia baseada em máquinas virtuais;

- Geração de registros de eventos: deve incluir as principais ações monitoradas por sistemas de segurança, como chamadas ao sistema e tráfego de rede;
- Padronização na geração de registros de eventos: dessa forma o desenvolvedor pode trabalhar livremente, independente da arquitetura-alvo, com base em um formato pré-estabelecido;
- Coleta e disponibilização de conjuntos de dados: deve prover registros a respeito do funcionamento legítimo do sistema e registros contendo ataques misturados ao funcionamento legítimo.

A construção de um ambiente de testes também deve ser conduzida de maneira a levar a um ambiente imparcial. Caso contrário, sua utilização fica prejudicada.

7.2.5 Automatização de *honeypots*

Honeypots são sistemas que implementam armadilhas para atrair potenciais atacantes. Esses sistemas são projetados para [BM01]:

- Desviar o atacante de um acesso a sistemas críticos;
- Coletar informação a respeito da atividade de atacantes;
- Encorajar o atacante a permanecer no sistema por um tempo suficiente até que os administradores possam efetuar uma resposta.

Em geral, a informação coletada em um *honeypot* é analisada de uma forma manual, requerendo conhecimento e supervisão humanos. Uma análise automatizada dessas informações pode habilitar um rápido entendimento das ações dos atacantes e possibilitar uma resposta bastante eficiente, antes mesmo que outros sistemas vulneráveis sejam comprometidos. Essa automatização de *honeypots* têm sido recentemente explorada em uma pesquisa sugerida através deste trabalho. Tal pesquisa, que analisa alterações no sistema de arquivos e pode possibilitar a identificação de ataques similares em outras máquinas, tem sido bem recebida pela comunidade científica [CdG04].

Considerando as características da arquitetura proposta neste trabalho é possível, ainda, dotar um *honeypot* de um mecanismo automático para extração de assinaturas de ataque no nível de rede. Dessa forma, é possível elaborar uma rede de *honeypots* — *honeynet* — contendo uma diversidade de *hardware* e *software* utilizado em um ambiente computacional. Quando uma atividade de ataque é identificada, é iniciado o processo de extração de assinatura e, tão logo a assinatura seja extraída, os sistemas vulneráveis são

protegidos. Utilizando idéias como essa é possível melhorar o tratamento das informações coletadas em *honeypots* e, idealmente, impedir a ocorrência de ataques em larga escala.

O processo de extração de assinatura pode ser visto, ainda, como uma espécie de análise forense automatizada. Outras aplicações poderiam ser obtidas através do estabelecimento de métodos e padrões para a captura automática de outros tipos de evidências, de forma que tais evidências possam ser aceitas durante um processo judicial.

7.3 Considerações finais

Foi desenvolvida uma arquitetura contemplando diversas características do sistema imunológico desejáveis a sistemas de segurança, especialmente sistemas de detecção de intrusão. A visualização do sistema imunológico como uma fonte inspiradora no nível arquitetural foi essencial para a modelagem das funcionalidades alcançadas. Ao invés de outras abordagem, esta pesquisa considera que ataques com sucesso são inevitáveis e, portanto, esta situação deve ser aproveitada para estudar esses ataques. O protótipo desenvolvido veio verificar as principais características dessa arquitetura e validar a hipótese desta tese.

Esta pesquisa também teve uma boa aceitação por parte da comunidade científica, com 3 artigos publicados em congressos internacionais e 2 artigos publicados em congressos nacionais. Outros trabalhos derivados desta linha de pesquisa também obtiveram sucesso na publicação de artigos em congressos nacionais e internacionais.

Embora as idéias propostas durante este trabalho possam não ser aplicadas integralmente em produtos comerciais, elas apresentam bastante potencial para automatizar a resposta a incidentes ou até mesmo para auxiliar a realização de análises manuais.

Referências bibliográficas

[ABC+03] Uwe Aickelin, Peter Bentley, Steve Cayzer, Jungwon Kim e Julie McLeod. Danger Theory: The link between AIS and IDS? Em *Proceedings of the Second International Conference on Artificial Immune Systems*, páginas 147–155, 2003.

[AC02] Uwe Aickelin e Steve Cayzer. The Danger Theory and its application to artificial immune systems. Em *Proceedings of the First International Conference on Artificial Immune Systems*, páginas 141–148, 2002.

[AK98] Ross Anderson e Abida Khattak. The use of information retrieval techniques for intrusion detection. Em *First Internation Workshop on Recent Advances in Intrusion Detection*, 1998.

[Alt04] AltaVista. Altavista. Disponível através da URL http://www.altavista.com, 2004.

[Amo99] Edward Amoroso. *Intrusion Detection: an Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response.* Intrusion.net Books, 1ª edição, 1999.

[And72] James Anderson. Computer security technology planning study 2. Relatório técnico, Electronic Systems Division, Air Force Systems Command, Hanscom Field, 1972.

[And80] James Anderson. Computer security threat monitoring and surveillance. Relatório técnico, James P. Anderson Co., 1980.

[APL97] Abul Abbas, Jordan Pober e Andrew Lichtman. *Cellular and Molecular Immunology.* W.B. Saunders, 3ª edição, 1997.

[Axe00] Stefan Axelson. Intrusion detection systems: A survey and taxonomy. Relatório técnico, Department of Computer Engineering, Chalmers University of Technology, 2000.

[Bac00] Rebecca Bace. *Intrusion Detection.* Macmillan Technical Publishing, 1ª edição, 2000.

[BAF+03] Elena Barrantes, David Ackley, Stephanie Forrest et al. Randomized instruction set emulation to disrupt binary code injection attacks. Em *Proceedings of the Tenth ACM Conference on Computer and Communications Security*, páginas 281–289, 2003.

[BEFG02] Justin Balthrop, Fernando Esponda, Stephanie Forrest e Matthew Glickman. Coverage and generalization in an artificial immune system. Em *Proceedings of the Genetic and Evolutionary Computation Conference*, páginas 3–10, 2002.

[BFG02] Justin Balthrop, Stephanie Forrest e Matthew Glickman. Revisiting LISYS: Parameters and normal behavior. Em *Proceedings of the 2002 Congress on Evolutionary Computation*, páginas 1045–1050, 2002.

[BGFI+98] Jai Balasubramaniyan, Jose Garcia-Fernandez, David Isaco, Eugene Spafford e Diego Zamboni. An architecture for intrusion detection using autonomous agents. Relatório técnico, CERIAS, Purdue University, 1998.

[BM01] Rebecca Bace e Peter Mell. Intrusion detection systems. NIST Special Publication on Intrusion Detection Systems. Disponível *on line* em fevereiro de 2004 na URL http://csrc.nist.gov/publications/nistpubs/800-31/sp800-31.pdf, 2001.

[Can97] Adriano Cansian. *Desenvolvimento de um Sistema Adaptativo de Detecção de Intrusos em Redes de Computadores.* Tese de doutorado, Instituto de Física, Universidade Estadual de São Paulo, 1997.

[Cap04] Computer Incident Advisory Capability. Red hat mremap() function vulnerability. CIAC, Disponível *on line* em março de 2004 na URL http://www.ciac.org/ciac/bulletins/o-045.shtml, 2004.

[CdG04] Martim Carbone e Paulo de Geus. A mechanism for automatic digital evidence collection on high-interaction honeypots. Em *Proceedings of the Fifth Annual IEEE Information Assurance Workshop*, páginas 1–8, 2004.

[Cen03] CERT Coordination Center. CERT/CC overview incident and vulnerability trends. Carnegie Mellon University, Disponível *on line* em março de 2004 na URL http://www.cert.org/present/cert-overview-trends, 2003.

[Cen04a] CERT Coordination Center. CERT/CC advisories 1988-2004. Carnegie Mellon University, Disponível *on line* em março de 2004 na URL http://www.cert.org/advisories, 1988-2004.

[Cen04b] CERT Coordination Center. CERT/CC advisories. Disponível *on line* em maio de 2004 na URL http://www.cert.org/advisories/#2003, 2004.

[Coh87] Fred Cohen. Computer viruses: Theory and experiments. *Computers & Security*, volume 6, páginas 22–35, 1987.

[CWP+00] Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie e Jonathan Walpole. Buffer overflows: Attacks and defenses for the vulnerability of the decade. Em *DARPA Information Survivability Conference and Exposition*, páginas 119–129, 2000.

[DAO97] Dipankar Dasgupta e Nii Attoh-Okine. Immunity-based systems: A survey. Em *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, páginas 369–374, 1997.

[Das99a] Dipankar Dasgupta. *Artificial Immune Systems and their Applications.* Springer-Verlag, 1ª edição, 1999.

[Das99b] Dipankar Dasgupta. Immunity-based intrusion detection systems: A general framework. Em *Proceedings of the Twenty Second National Information Systems Security Conference*, páginas 147–160, 1999.

[DB01] Dipankar Dasgupta e Hal Brian. Mobile security agents for network traffic analysis. Em *Proceedings of the Second DARPA Information Survivability Conference and Exposition II*, páginas 332–340, 2001.

[DBS92] Herver Debar, Monique Becker e Didier Siboni. A neural network component for an intrusion detection system. Em *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, páginas 478–483, 1992.

[dC01] Leandro de Castro. *Engenharia Imunológica: Desenvolvimento e Aplicação de Ferramentas Computacionais Inspiradas em Sistemas Imunológicos Artificiais.* Tese de doutorado, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 2001.

[dCT02] Leandro de Castro e Jonathan Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach.* Springer-Verlag, 1ª edição, 2002.

[dCZ04] Leandro de Castro e Fernando Von Zuben. *Recent Developments in Biologically Inspired Computing.* Idea Group Publishing, 1ª edição, 2004.

[DDW99] Hervé Debar, Marc Dacier e Andreas Wespi. A revised taxonomy for intrusion-detection systems. Relatório técnico, IBM Research, Zurich Research Laboratory, 1999.

[Den86] Dorothy Denning. An intrusion detection model. Em *Proceedings of the Seventh IEEE Symposium on Security and Privacy*, páginas 119–131, 1986.

[Des02] Solar Designer. Non-executable user stack. Disponível *on line* em novembro de 2002 na URL http://www.angelfire.com/sk/stackshield, 2002.

[DFH96] Patrik D'haeseleer, Stephanie Forrest e Paul Helman. An immunological approach to change detection: algorithms, analysis and implications. Em *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*, páginas 110–119, 1996.

[DG01] Dipankar Dasgupta e Fabio Gonzalez. An intelligent decision support system for intrusion detection and response. Em *Proceedings of the International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security*, páginas 1–14, 2001.

[DG02] Dipankar Dasgupta e Fabio González. An immunity-based technique to characterize intrusions in computer networks. *IEEE Transactions on Evolutionary Computation*, volume 6, páginas 281–291, 2002.

[D'h96] Patrik D'haeseleer. An immunological approach to change detection: Theoretical results. Em *Proceedings of the Ninth IEEE Computer Security Foundations Workshop*, páginas 110–119, 1996.

[Dik01] Jeff Dike. The User-mode Linux kernel home page. Disponível *on line* em maio de 2004 na URL http://user-mode-linux.sourceforge.net/index.html, 2001.

[dPdCdG04] Fabrício de Paula, Leandro de Castro e Paulo de Geus. An intrusion detection system using ideas from the immune system. Em *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, 2004. Em publicação.

[dPdG04] Fabrício de Paula e Paulo de Geus. Attack evidence detection, recovery and signature extraction with ADenoIdS. Em José de Souza e Petre Dini, editors, *Lecture Notes in Computer Science, como Proceedings of*

the Eleventh International Conference on Telecommunications. Springer-Verlag, 2004. Em publicação.

[dPdRFdG02] Fabrício de Paula, Marcelo dos Reis, Diego Fernandes e Paulo de Geus. ADenoIdS: A hybrid IDS based on the immune system. Em *Proceedings of the Ninth International Conference on Neural Information Processing*, páginas 479–484, 2002.

[dR03] Marcelo dos Reis. Forense computacional e sua aplicação em segurança imunológica. Dissertação de mestrado, Instituto de Computação, Universidade Estadual de Campinas, 2003.

[dRdPFdG02] Marcelo dos Reis, Fabrício de Paula, Diego Fernandes e Paulo de Geus. A hybrid IDS architecture based on the immune system. Em *Anais do Segundo Workshop em Segurança de Sistemas Computacionais*, páginas 33–40, 2002.

[dRFdPdG01] Marcelo dos Reis, Diego Fernandes, Fabrício de Paula e Paulo de Geus. Modelagem de um sistema de seguranca imunológico. Em *Anais do Terceiro Simpósio Seguranca em Informática*, páginas 91–100, 2001.

[EFH04] Fernando Esponda, Stephanie Forrest e Paul Helman. A formal framework for positive and negative detection schemes. *IEEE Transactions on Systems, Man and Cybernetics*, páginas 357–373, 2004.

[Esc99] Laurent Eschenauer. Imsafe ids. Disponível *on line* em março de 2004 na URL http://imsafe.sourceforge.net, 1999.

[FAPC94] Stephanie Forrest, Lawrence Allen, Alan Perelson e Rajesh Cherukuri. Self-nonself discrimination in a computer. Em *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, páginas 202–212, 1994.

[FBGA02] Stephanie Forrest, Justin Balthrop, Matthew Glickman e David Ackley. Computation in the wild. Em Kihong Park e Walter Williger, editors, *The Internet as a Large-Scale Complex System.* Oxford University Press, 2002.

[Fer03] Diego Fernandes. Resposta automática em um sistema de segurança imunológico computacional. Dissertação de mestrado, Instituto de Computação, Universidade Estadual de Campinas, 2003.

[FH99] Stephanie Forrest e Steven Hofmeyr. John Holland's invisible hand: An artificial immune system. Presented at the Festschrift held in honor of John Holland, 1999.

[FH00] Stephanie Forrest e Steven Hofmeyr. Architecture for an artificial immune system. *Evolutionary Computation Journal*, volume 7, páginas 1289–1296, 2000.

[FHS97] Stephanie Forrest, Steven Hofmeyr e Anil Somayaji. Computer immunology. *Communications of the ACM*, volume 40, páginas 88–96, 1997.

[FHSL96] Stephanie Forrest, Steven Hofmeyr, Anil Somayaji e Thomas Longstaff. A sense of self for unix processes. Em *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, páginas 120–128, 1996.

[FS01] Mike Frantzen e Mike Shuey. Stackghost: Hardware facilitated stack protection. Em *Proceedings of the 2001 USENIX Security Symposium*, páginas 55–66, 2001.

[GD79] Mauri Guerra e Denis Donaire. *Estatística Indutiva: Teoria e Aplicações*. Livraria Ciência e Tecnologia Editora, 1ª edição, 1979.

[GD02] Fabio González e Dipankar Dasgupta. An immunogenetic technique to detect anomalies in network traffic. Em *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002.

[GGD03] Jonatan Gómez, Fabio González e Dipankar Dasgupta. An immuno-fuzzy approach to anomaly detection. Em *Proceedings of the IEEE International Conference on Fuzzy Systems*, páginas 1219–1224, 2003.

[Gia92] Joseph Giarratano. CLIPS version 5.1 user's guide. NASA, Lyndon B. Johnson Space Center, 1992.

[Gre04] Eric Grevstad. CPU-based security: The NX bit. Disponível *on line* em julho de 2004 na URL http://hardware.earthweb.com/chips/article.php/3358421, 2004.

[H+93] Judith Hochberg et al. NADIR: An automated system for detecting network intrusion and misuse. *Computers & Security*, volume 12, páginas 235–248, 1993.

[HF99] Steven Hofmeyr e Stephanie Forrest. Immunity by design: An artificial immune system. Em *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, páginas 1289–1296. Morgan Kaufmann, 1999.

[HFD98] Steven Hofmeyr, Stephanie Forrest e Patrik D'haeseleer. An immunological approach to distributed network intrusion detection. Em *First International Workshop on the Recent Advances in Intrusion Detection*, 1998.

[HFS98] Steven Hofmeyr, Stephanie Forrest e Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, volume 6, páginas 151–180, 1998.

[HM04] Jed Haile e Rob McMillen. Snort-inline tool. Disponível *on line* em abril de 2004 na URL http://project.honeynet.org/papers/honeynet/tools, 2004.

[Hof99] Steven Hofmeyr. *An Immunological Model of Distributed Detection and its Application to Computer Security.* Tese de doutorado, Department of Computer Sciences, University of New Mexico, 1999.

[Hua00] Xie Huagang. Build a secure system with LIDS. Disponível *on line* em abril de 2004 na URL http://www.lids.org/document/build_lids-0.2.html, 2000.

[IH02] Warren Kruse II e Jay Heiser. *Computer Forensics: Incident Response Essentials.* Addison Wesley, 1ª edição, 2002.

[JV91] Harold Javitz e Alfonso Valdes. The SRI IDES statistical anomaly detector. Em *Proceedings of 1991 IEEE Symposium on Security and Privacy*, páginas 316–326, 1991.

[KA94] Jeffrey Kephart e William Arnold. Automatic extraction of computer virus signatures. Em *In Proceedings of the Fourth Virus Bulletin International Conference*, páginas 178–184, 1994.

[KB99a] Jungwon Kim e Peter Bentley. An artificial immune model for network intrusion detection. Em *Proceedings of the Seventh European Congress on Intelligent Techniques and Soft Computing*, 1999.

[KB99b] Jungwon Kim e Peter Bentley. The human immune system and network intrusion detection. Em *Proceedings of the Seventh European Congress on Intelligent Techniques and Soft Computing*, 1999.

[KB99c] Jungwon Kim e Peter Bentley. Negative selection and niching by an artificial immune system for network intrusion detection. Em *Proceedings of the Genetic and Evolutionary Computation Conference*, páginas 149–158, 1999.

[KB01a] Jungwon Kim e Peter Bentley. Evaluating negative selection in an artificial immune system for network intrusion detection. Em *Proceedings of the Genetic and Evolutionary Computation Conference*, páginas 1330–1337, 2001.

[KB01b] Jungwon Kim e Peter Bentley. Towards an artificial immune system for network intrusion detection: An investigation of clonal selection with a negative selection operator. Em *Proceedings of the 2001 Congress on Evolutionary Computation*, páginas 1244–1252, 2001.

[KB02] Jungwon Kim e Peter Bentley. Towards an artificial immune system for network intrusion detection: An investigation of dynamic clonal selection. Em *Proceedings of the 2002 Congress on Evolutionary Computation*, páginas 1015–1020, 2002.

[KBA02] Vladimir Kiriansky, Derek Bruening e Saman Amarasinghe. Secure execution via program shepherding. Em *Proceedings of the 2002 USENIX Security Symposium*, páginas 191–206, 2002.

[KCW93] Jeffrey Kephart, David Chess e Steve White. Computers and epidemiology. *IEEE SPECTRUM*, páginas 20–26, 1993.

[Kep94] Jeffrey Kephart. A biologically inspired immune system for computers. Em *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, páginas 130–139, 1994.

[KFBK00] Calvin Ko, Timothy Fraser, Lee Badger e Douglas Kilpatrick. Detecting and countering system intrusions using software wrappers. Em *Proceedings of the 2000 USENIX Security Symposium*, páginas 145–156, 2000.

[KFL94] Calvin Ko, George Fink e Karl Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. Em *Proceedings of the Tenth Annual Computer Security Applications Conference*, páginas 134–144, 1994.

[Kim02] Jungwon Kim. *Integrating Artificial Immune Algorithms for Intrusion Detection.* Tese de doutorado, Department of Computer Science, University College London, 2002.

[KS94a] Gene Kim e Eugene Spafford. The design and implementation of Tripwire: A file system integrity checker. Em *Proceedings of the Second ACM*

Conference on Computer and Communications Security, páginas 18–29, 1994.

[KS94b] Sandeep Kumar e Eugene Spafford. A pattern matching model for intrusion detection. Em *Proceedings of the Seventeenth National Computer Security Conference*, páginas 11–21, 1994.

[KSSW97] Jeffrey Kephart, Gregory Sorkin, Morton Swimmer e Steve White. Blueprint for a computer immune system. Em *Virus Bulletin International Conference in San Francisco*, 1997.

[KTK02] Christopher Kruegel, Thomas Toth e Engin Kirda. Service specific anomaly detection for network intrusion detection. Em *ACM Symposium on Applied Computing*, páginas 201–208, 2002.

[L+90] Teresa Lunt et al. IDES: A progress report. Em *Proceedings of the Sixth Annual Computer Security Applications Conference*, páginas 273–285, 1990.

[Lab99] MIT Lincoln Laboratory. 1999 DARPA intrusion detection evaluation data set. Disponível *on line* em maio de 2004 na URL http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html, 1999.

[LB91] 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*, 1991.

[LE01] David Larochelle e David Evans. Statically detecting likely buffer overflow vulnerabilities. Em *Proceedings of the 2001 USENIX Security Symposium*, páginas 177–190, 2001.

[LS00] Wenke Lee e Salvatore Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, volume 3, páginas 227–261, 2000.

[LSM99] Wenke Lee, Salvatore Stolfo e Kui Mok. A data mining framework for building intrusion detection models. Em *Proceedings of the Twentieth IEEE Symposium on Security and Privacy*, páginas 120–132, 1999.

[LV92] Liepins e Vaccaro. Intrusion detection: Its role and validation. *Computers & Security*, volume 11, páginas 347–355, 1992.

[Mé98] Ludovic Mé. GASSATA, a genetic algorithm as an alternative tool for security audit trails analysis. Em *First International Workshop on Recent Advances in Intrusion Detection*, 1998.

[Mat96] Polly Matzinger. The real function of the immune system or tolerance and the four d's (danger, death, destruction and distress). Disponível *on line* em março de 2004 na URL http://cmmg.biosci.wayne.edu/asg/polly.html, 1996.

[NCFF01] Stephen Northcutt, Mark Cooper, Matt Fearnow e Karen Frederick. *Intrusion Signatures and Analysis.* New Riders Publishing, 1ª edição, 2001.

[NdG03] Emilio Nakamura e Paulo de Geus. *Segurança de redes em ambientes cooperativos.* Editora Futura, 3ª edição, 2003.

[NP99] Peter Neumann e Phillip Porras. Experience with EMERALD to date. Em *Proceedings of the USENIX Workshop on Intrusion Detection and Network Monitoring*, páginas 73–80, 1999.

[One96] Aleph One. Smashing the stack for fun and profit. Phrack Magazine, volume 49, 1996.

[Pet00] Richard Pethia. Computer security. CERT Coordination Center. Disponível *on line* em maio de 2004 na URL http://www.cert.org/congressional_testimony/Pethia_testimony_Mar9.html, 2000.

[Por92] Phillip Porras. STAT, a state transition analysis tool for intrusion detection. Dissertação de mestrado, Computer Science, University of California, 1992.

[Pro01] Paul Proctor. *The Practical Intrusion Detection Handbook.* Prentice Hall PTR, 1ª edição, 2001.

[Pro03] Niels Provos. Improving host security with system call policies. Em *Proceedings of the Twelfth USENIX Security Symposium*, páginas 257–272, 2003.

[Qui98] Helen Quill. *Report of the NIAID Task Force on Immunology.* National institutes of Health, 1998.

[Raf03] Jason Rafail. Vulnerability note VU#301156: Linux kernel do_brk() function contains integer overflow. US-CERT, Disponível *on line* em março de 2004 na URL https://www.kb.cert.org/CERT_WEB/services/vulnotes.nsf/id/301156, 2003.

[RBM96] Ivan Roitt, Jonathan Brostoff e David Male. *Immunology.* Mosby, 4ª edição, 1996.

[SBS99] Sekar, Bowen e Segal. On preventing intrusions by process behavior monitoring. Em *Proceedings of the 1999 USENIX Security Symposium*, páginas 29–40, 1999.

[SCC+96] Stuart Staniford-Chen, Steven Cheung et al. GrIDS—a graph-based intrusion detection system for large networks. Em *The Nineteenth National Information Systems Security Conference*, 1996.

[Sec04] NFR Security. Network flight recorder. Disponível através da URL http://www.nfr.net, 2004.

[SF00] Anil Somayaji e Stephanie Forrest. Automated response using system-call delays. Em *Proceedings of the 2000 USENIX Security Symposium*, páginas 185–198, 2000.

[SFA97] Anil Somayaji, Stephanie Forrest e David Ackley. Building diverse computer sytems. Em *Proceedings of Sixth Workshop on Hot Topics in Operating Systems*, páginas 67–72, 1997.

[SG96] Gene Spafford e Simson Garfinkel. *Practical Unix & Internet Security.* O Reilly and Associates, 2ª edição, 1996.

[SHF97] Anil Somayaji, Steven Hofmeyr e Stephanie Forrest. Principles of a computer immune system. Em *Proceedings of the 1997 New Security Paradigms Workshop*, páginas 75–82, 1997.

[SLD02] Edward Suh, Jaewook Lee e Srinivas Devadas. Secure program execution via dynamic information flow tracking. Relatório técnico, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2002.

[Sma88] Stephen Smaha. Haystack: An intrusion detection system. Em *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, páginas 37–44, 1988.

[Som02] Anil Somayaji. *Operating System Stability and Security through Process Homeostasis.* Tese de doutorado, Department of Computer Sciences, University of New Mexico, 2002.

[Spa88] Eugene Spafford. The internet worm program: An analysis. Relatório técnico, Department of Computer Sciences, Purdue University, 1988.

[Spa89] Eugene Spafford. The internet worm: Crisis and aftermath. *Communications of the ACM*, volume 32, páginas 678–687, 1989.

[Ste00a] Peter Stephenson. *Investigating Computer-Related Crime.* CRC Press, 1ª edição, 2000.

[Ste00b] Richard Stevens. *TCP/IP Illustrated*, volume 1. Addison Wesley, 1ª edição, 2000.

[SZ00] Eugene Spafford e Diego Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, volume 34, páginas 547–570, 2000.

[T+01] Andreas Tscharner et al. Towards a taxonomy of intrusion detection systems and attacks. MAFTIA deliverable D3 Version 1.01. Disponível *on line* em fevereiro de 2004 na URL http://www.newcastle.research.ec.org/maftia/deliverables/D3.pdf, 2001.

[TCL90] Henry Teng, Kaihu Chen e Stephen Lu. Adaptive real-time anomaly detection using inductively generated sequential patterns. Em *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, páginas 278–284, 1990.

[Tea02] PaX Team. Non executable data pages. Disponível *on line* em março de 2004 na URL http://pax.grsecurity.net, 2002.

[TK02] Thomas Toth e Christopher Kruegel. Accurate buffer overflow detection via abstract payload execution. Em *Fifth International Symposium on Recent Advances in Intrusion Detection*, páginas 274–291, 2002.

[Twy04] Jamie Twycross. Immune systems, Danger Theory and intrusion detection. Em *2004 Symposium on The Immune System and Cognition*, 2004.

[UC04] US-CERT. US-CERT current activity. Disponível *on line* em maio de 2004 na URL http://www.us-cert.gov/current/current_activity.html#sasser, 2004.

[Uto03] Nelson Uto. Segurança de sistemas de agentes móveis. Dissertação de mestrado, Instituto de Computação, Universidade Estadual de Campinas, 2003.

[Ven00] Vendicator. Stackshield: A "stack smashing" technique protection tool for linux. Disponível *on line* em março de 2004 na URL http://www.angelfire.com/sk/stackshield, 2000.

[W+88] Alan Whitehurst et al. Expert systems in intrusion detection: A case study. Em *Proceedings of the Eleventh National Computer Security Conference*, páginas 74–81, 1988.

[WFBA00] David Wagner, Jeffrey Foster, Eric Brewer e Alexander Aiken. A first step towards automated detection of buffer overrun vulnerabilities. Em *Network and Distributed System Security Symposium*, páginas 3–17, 2000.

[WFP99] Christina Warrender, Stephanie Forrest e Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. Em *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, páginas 133–145, 1999.

[XKPI02] Jun Xu, Zbigniew Kalbarczyk, Sanjay Patel e Ravishankar Iyer. Architecture support for defending against buffer overflow attacks. Em *Proceedings of the Second Workshop on Evaluating and Architecting System Dependability*, 2002.