Resposta Automática em um Sistema de Segurança Imunológico Computacional

Diego de Assis Monteiro Fernandes

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Resposta Automática em um Sistema de Segurança Imunológico Computacional

Diego de Assis Monteiro Fernandes

Fevereiro de 2003

Banca Examinadora:

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

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Fernandes, Diego de Assis Monteiro

F391r Resposta automática em um sistema de segurança imunológico computacional / Diego de Assis Monteiro Fernandes -- Campinas, [S.P. :s.n.], 2003.

Orientador : Paulo Lício de Geus

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Computadores - Medidas de segurança. 2. Sistemas de consultas e respostas. 3. Internet (Redes de computação). 4. Redes de computação – Medidas de segurança. I. Geus, Paulo Lício de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

TERMO DE APROVAÇÃO

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

Prof. Dr. Carlos Alberto Maziero
PUC - PR

Prof. Dr. Ricardo Dahab
IC - UNICAMP

Prof. Dr. Paulo Lício de Geus
IC – UNICAMP

Resposta Automática em um Sistema de Segurança Imunológico Computacional

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Diego de Assis Monteiro Fernandes e aprovada pela Banca Examinadora.

Campinas, 26 de fevereiro de 2003.

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

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

Agradecimentos

A Deus pela minha existência e por todas as oportunidades proporcionadas.

Aos meus pais, minhas eternas inspirações, por todo o cuidado, carinho e incentivo incondicionais em toda a minha vida.

À Tatinha, pelo carinho, apoio, companheirismo e paciência.

À Dona Idalina, minha grande avó, que me ensinou a sempre sorrir para os acontecimentos da vida.

À minha irmã Maria Gabriela, por toda sua ternura.

Ao Paulo, meu irmão e grande amigo.

À Tita, vô Toni, vó Daysi, tio Sérgio e Beth que, apesar da distância, sempre me deram apoio e incentivo.

Ao meu orientador, Paulo Lício de Geus, pela oportunidade concedida e pela orientação.

Ao Fabrício e Marcelo, meus companheiros desde a graduação, por todos os momentos compartilhados, e pela amizade conquistada.

A todos os integrantes e ex-integrantes do LAS: Alessandro, Cleymone, Edmar, Flávio, Jansen e João por todas as experiências e conhecimentos compartilhados.

Ao Capitán Bado, por todas as músicas ouvidas e tocadas, e pelas situações sempre muito divertidas.

À CAPES e FAPESP, pelo apoio financeiro

Aos colegas do IC, por todas as conversas e cafés degustados.

Aos professores e funcionários do Instituto de Computação.

Aos meus pais e à minha Tatinha.

Resumo

O sistema imunológico humano, por ser capaz de garantir a sobrevivência de um indivíduo durante cerca de 70 anos, mesmo que ele se depare, a cada dia, com bactérias e vírus potencialmente mortais, provê uma rica fonte de inspiração para a segurança de redes de computadores. Além de representar um modelo bastante próximo das condições em que a maioria das redes de computadores se encontra, o sistema imunológico humano engloba uma série de características desejáveis a um sistema de segurança.

Baseado nisso, este trabalho apresenta um mecanismo automático de resposta a intrusões que integra um sistema de segurança projetado para detectar e responder a potenciais ameaças, mesmo que previamente não conhecidas. Com as respostas providas pelo mecanismo almeja-se conter o poder de uma possível intrusão, evitando que atividades ilícitas possam aumentar o comprometimento da confidencialidade, integridade e disponibilidade de um sistema.

É fato que estratégias inadequadas de resposta ou respostas disparadas sob a detecção de falsos positivos podem prejudicar usuários e processos legítimos. Considerando essa possibilidade, o mecanismo dispõe de funcionalidades que procuram suavizar os efeitos de tal reação imprópria.

Abstract

The human immune system, since it is able to guarantee the survival of an individual for almost 70 years, even though he/she encounters potentially deadly parasites, bacteria and viruses on a daily basis, provides a rich source of inspiration for computer network security. 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 presents an automatic intrusion response mechanism as part of a security system developed to detect and respond to harmful activities, including those not previously encountered. It is expected, with the response provided by the mechanism, to limit the harmful effects of a intrusion, thus avoiding illict activities to futher compromise confidenciality, integrity and availability of the system.

It is well known that inadequate response strategies or responses triggered by false positive detection can harm legitimate user and process activities. Given this, the mechanism has features that seek to minimize the effects of inadequate reaction.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A sociedade tem se tornado cada vez mais dependente do acesso a redes de computadores, que, dentre seus benefícios, proporciona um compartilhamento intenso de informações. Esse compartilhamento, entre vários de seus benefícios, torna acessível detalhes de implementação, configuração e mesmo falhas de grande parte das aplicações comumente utilizadas nos computadores. Tais informações viabilizam a descoberta e exploração das falhas encontradas nas aplicações para, por exemplo, obter acesso não autorizado, levando assim ao comprometimento da segurança.

Qualquer falha ou fraqueza encontrada no sistema que viabilize atividades ilícitas são denominadas vulnerabilidades e dividem-se em vulnerabilidades técnicas ou de gerenciamento. Vulnerabilidades técnicas são fraquezas que ocorrem no projeto ou implementação do *hardware* ou *software*. Já as vulnerabilidades de gerenciamento são fraquezas que ocorrem por falhas na elaboração da política de segurança, configuração ou outras áreas de gerenciamento do sistema. Independente da vulnerabilidade, sua exploração constitui uma etapa essencial para o comprometimento da segurança de um sistema.

Uma definição interessante que caracteriza a segurança de um sistema, apresentada em [41], é a de que um sistema está seguro quando ele se comporta como o esperado. Para constatação dessa condição, é importante que se tenha bem esclarecido qual é o comportamento esperado do sistema. Um modo interessante de formalizá-lo é através do estabelecimento de uma política de segurança. Uma política de segurança pode ser definida como o conjunto de leis, regras e práticas que formalizam como uma organização deve gerenciar, proteger e distribuir seus recursos[62].

Além da definição de segurança mencionada, uma definição mais formal pode ser elaborada. Essa definição afirma que um sistema é seguro quando três condições sobre os seus recursos são garantidas: confidencialidade, integridade e disponibilidade[90]. A confidencialidade requer que os recursos estejam acessíveis somente a usuários autorizados. A integridade requer que os recursos permaneçam inalterados por acidentes ou tentativas

maliciosas. E, por fim, a disponibilidade requer que o sistema permaneça em funcionamento sem degradação de acesso e atenda requisições de uso dos recursos para usuários autorizados.

Baseado nas definições de segurança apresentadas, uma intrusão pode ser definida tanto como uma violação da política de segurança de um sistema[62] ou como o comprometimento da sua confidencialidade, integridade ou disponibilidade. Para evitar uma intrusão, diversos sistemas computacionais estão providos de mecanismos estáticos de segurança, como mecanismos de autenticação e controle de acesso. Entretanto, é comum que tais mecanismos também sejam explorados para o comprometimento da segurança, justificando a necessidade de adotar mecanismos adicionais para aprimorá-la[15].

Os sistemas de detecção de intrusão desempenham essa função e geralmente formam a última linha de defesa no esquema geral de segurança de um sistema[62]. A detecção de intrusão é útil não somente por detectar falhas na segurança, mas também por monitorar tentativas de intrusão, provendo assim informações importantes para efetuar respostas. Dispõe-se hoje de uma grande variedade de sistemas de detecção de intrusão, adotando diferentes abordagens e técnicas[69].

Entretanto, a maioria dos esforços nessa área volta-se especificamente para a descoberta de uma intrusão. Já o desenvolvimento de mecanismos de resposta ainda é pouco abordado, cabendo ao administrador executar as medidas necessárias para conter o poder do ataque e restaurar os danos provocados. Esse esforço para automatizar a resposta, apesar de recente, tem contado com um número crescente de pesquisas. Um mecanismo de resposta automático é importante pois, além dos ataques tornarem-se mais numerosos e complexos[87], não se pode contar com a ação do administrador imediata à detecção. Além disso, quanto mais rápido for contido o ataque, menor é a probabilidade da ocorrência de danos ao sistema, pois o atraso existente entre a detecção e a resposta provê uma oportunidade de exploração para o atacante.

O projeto de um mecanismo de resposta automático deve considerar algumas questões para elaborar respostas eficientes e diminuir a chance de prejudicar eventos legítimos do sistema. Um mecanismo projetado de forma insatisfatória pode transformar uma ferramenta de contenção de ataques em um artifício para o atacante. A exploração de tal mecanismo, para execução de respostas inapropriadas, é capaz de provocar uma negação de serviço a usuários e processos legítimos.

De um modo geral, as questões envolvendo a detecção e resposta a intrusão são debatidas de modo a obter um nível melhor de segurança. Esse nível de segurança pode ser atingido adotando-se recursos adicionais e melhores modelos, que representem de maneira mais próxima as condições em que a maioria das redes de computadores se encontra—um ambiente hostil e sujeito a falhas. 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, por ser capaz de garantir a sobrevivência de um indivíduo durante cerca de 70 anos, mesmo que ele se depare, a cada dia, com bactérias e vírus potencialmente mortais, apresenta um paralelo bastante forte com a segurança de redes de computadores.

Esta dissertação apresenta um mecanismo de resposta automático para um sistema de detecção de intrusão e resposta baseado no sistema imunológico humano. O mecanismo de resposta conta com um conjunto de ações que tem como objetivo bloquear o poder do ataque, restaurando o sistema a um estado seguro. Nele, é considerado que cada ataque conhecido deve contar com um tipo específico de resposta, propondo adicionalmente outro esquema para conter ataques ainda não conhecidos.

Além disso, questões pertinentes a respostas disparadas na detecção de falsos positivos também foram abordadas. Nesses casos, espera-se, através de algumas ações especiais do mecanismo, suavizar os efeitos prejudiciais provocados em usuários e processos legítimos. Mais ainda, o mecanismo proposto é capaz de organizar e executar uma resposta distribuída, através de um esquema de comunicação que utiliza a tecnologia de agentes móveis. O uso da tecnologia é interessante, pois adota um paradigma que apresenta vários benefícios, quando comparado aos paradigmas tradicionais de computação em rede[64].

As estratégias de resposta descritas podem ser definidas e configuradas através de uma política de resposta.

1.1 Organização do trabalho

A organização do trabalho é apresentada como segue. O Capítulo 2 aborda os sistemas de detecção de intrusão e resposta, apresentando as abordagens e técnicas utilizadas. O Capítulo 3 descreve a estrutura e o funcionamento do sistema imunológico humano, bem como a aplicação de seus conceitos na área de segurança de computadores. A tecnologia de agentes móveis, empregada no mecanismo, é tratada no Capítulo 4 onde são mencionados os conceitos básicos e sua aplicação na detecção de intrusão e resposta. O Capítulo 5 descreve o sistema de segurança baseado no sistema imunológico humano onde o mecanismo de resposta proposto está inserido. A modelagem do mecanismo de resposta é apresentada no Capítulo 6. Sua implementação e os testes realizados são abordados no Capítulo 7. Por fim, a conclusão e os trabalhos futuros são expostos no Capítulo 8.

Capítulo 2

Sistemas de Detecção de Intrusão e Resposta

A detecção de intrusão pode ser definida como o processo de monitoramento e análise de eventos em um sistema computacional a procura de sinais que indiquem a ocorrência de problemas de segurança[11]. Sistemas de detecção de intrusão, também conhecidos pelo acrônimo IDS[1], são sistemas que automatizam o processo de monitoramento e análise dos eventos de um sistema ou de uma rede em busca de problemas de segurança, ou seja, automatizam o processo de detecção de intrusão. Um sistema de detecção sozinho não é capaz de garantir a segurança de uma organização. Ao invés disso, é parte de uma infraestrutura composta por outros mecanismos e técnicas de segurança, tais como: mecanismos de controle de acesso, criptografia, *firewall*[105] e mecanismos de identificação e autenticação. Apesar de ser apenas uma parte dessa infraestrutura, um sistema de detecção de intrusão é fundamental para tornar a segurança de um sistema computacional mais robusta.

Em geral, os sistemas de detecção de intrusão são utilizados com os seguintes objetivos:

- **Atribuição de responsabilidades**: É a capacidade de relacionar determinadas atividades ou eventos ao indivíduo responsável. A atribuição de responsabilidade torna-se essencial quando deseja-se tomar medidas jurídicas contra o atacante. Essa tarefa, dependendo do cenário do ataque, pode tornar-se bastante complexa[10].

- **Resposta**: É a capacidade de caracterizar uma certa atividade ou evento como um ataque e então executar ações com o intuito de bloqueá-lo ou de prevenir uma nova ocorrência. 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,

[1] O acrônimo em inglês foi escolhido por ser de uso corrente tanto na literatura brasileira como na estrangeira.

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

Além dos principais objetivos descritos, os sistemas de detecção de intrusão e resposta provêem outros benefícios que contribuem para a segurança de um sistema. Primeiro, a utilização de um IDS aumenta a possibilidade de descoberta de um ataque, inibindo a atividade de atacantes. Mais ainda, são capazes de detectar ataques internos. Segundo, respostas automáticas são rápidas e diminuem a possibilidade de danos ao sistema[16]. Terceiro, monitorar o restante da infraestrutura de segurança permite aos administradores encontrarem falhas na administração e no projeto de segurança, além de também viabilizar a descoberta de possíveis ameaças para uma organização que não são prevenidas pelos outros mecanismos de segurança. Quarto, provê informações úteis sobre tentativas de intrusões ou intrusões efetuadas, capacitando a elaboração de um diagnóstico mais aprimorado para a recuperação e correção das causas.

Esse capítulo tem como objetivo fazer uma revisão sobre os sistemas de detecção de intrusão, apresentando seu funcionamento, suas características e as técnicas adotadas. A revisão baseia-se principalmente nos trabalhos de Rebecca Bace[11, 10, 9] que classificam os trabalhos na área de um modo interessante. Sua organização é mostrada como segue. A Seção 2.1 descreve a origem da detecção de intrusão. A Seção 2.2 apresenta um modelo genérico de um sistema de detecção adotado como base para a descrição de componentes. Esse modelo divide um sistema de detecção em três componentes básicos: fonte de informação, análise e resposta. Cada um dos componentes são descritos, respectivamente, nas Seções 2.3, 2.4 e 2.6, enquanto que a Seção 2.5 apresenta as técnicas utilizadas no componente de análise. Por fim, a Seção 2.7 apresenta algumas ferramentas com funções similares e complementares aos sistemas de detecção de intrusão, como por exemplo os sistemas verificadores de integridade.

2.1 Origem da detecção de intrusão

A detecção de intrusão teve origem no sistema de auditoria financeiro, onde um auditor especializado inspecionava os registros das transações de uma empresa, procurando por fraudes e erros. Esse sistema, inicialmente efetuado por um especialista, a partir de 1950 começou a ser executado pelo computador, fixando-se definitivamente em meados da década de 60. Com o desenvolvimento dos computadores e com sua propagação na década de 70, a demanda por segurança tornou-se uma necessidade óbvia[70, 11]. Essa demanda mobilizou os especialistas nos sistemas de auditoria financeira para desenvolvimento de técnicas que permitissem a detecção de atividades ilícitas, dando início às pesquisas na área da detecção de intrusão.

James P. Anderson foi o primeiro a sugerir a necessidade de automatização de um sistema de auditoria voltado para a segurança. Além disso, é seu o crédito da criação do primeiro esquema automatizado de detecção de intrusão, que adotava um método de detecção de mau uso em *mainframes*. Seu artigo publicado em 1972[3] atentava ao surgimento de problemas de segurança em computadores. Mais tarde, em 1980, James P. Anderson publicou um estudo sugerindo melhorias para a auditoria de segurança[4]. Esse estudo apresentava uma taxonomia de ameaças e intrusos, ressaltando a necessidade de compreensão das possíveis ameaças e ataques para facilitar seu reconhecimento nos dados da auditoria.

Entre 1984 e 1986, Dorothy Denning e Peter Neumann desenvolveram o primeiro modelo de um IDS em tempo real[27]. Esse modelo assume que as atividades intrusivas diferem das atividades normais. Com isso, sua tarefa é encontrar atividades que não se enquadram em representações do comportamento normal. Seu protótipo chamava-se *Intrusion Detection Expert System* (IDES)[8], inicialmente um sistema especialista treinado para detectar atividades maliciosas conhecidas. Posteriormente foi refinado e estendido, sendo renomeado para *Next-Generation Intrusion Detection Expert System* (NIDES)[2]. Os relatórios publicados por James P. Anderson e o trabalho de Denning e Neumann foram o começo de muitas das pesquisas em detecção de intrusão na década de 80 e ainda servem como base para muitos dos trabalhos atuais.

2.2 Modelo genérico de um processo de detecção de intrusão e resposta

Um sistema de detecção de intrusão e resposta conta com várias técnicas possíveis de serem utilizadas para desempenhar suas funções. Essas técnicas, como por exemplo os métodos de monitoramento ou de análise de eventos, podem servir como base para uma possível classificação de IDSs. O modelo genérico de um IDS apresentado nessa seção, organiza as classificações e técnicas em termos de três componentes fundamentais. O primeiro deles é a fonte de informação, que provê o fluxo de entrada dos eventos a serem analisados. O segundo é o componente de análise que busca no fluxo de entrada indícios de uma intrusão. Por fim, o componente de resposta que é responsável por gerar um conjunto de ações baseadas na saída produzida pela análise.

- **Fonte de informação**: Esse componente é responsável pelo processo de coleta dos eventos posteriormente analisados. Existem diferentes tipos de fontes de informação monitoradas para determinar a ocorrência de uma intrusão, sendo os monitoramentos mais comuns os de rede, máquina e aplicação.

- **Análise**: É o componente de análise que organiza e seleciona os eventos relevantes derivados da fonte de informação. A partir dessa análise é possível deduzir que um determinado conjunto de eventos caracteriza uma intrusão. As abordagens mais comuns utilizadas no processo de análise são a detecção de anomalia e a detecção de mau uso.

- **Resposta**: A resposta é formada pelo conjunto de ações tomado pelo IDS sempre que uma intrusão é detectada. Em geral, a resposta é classificada em dois tipos: passivas e ativas. Respostas ativas compreendem as ações que atuam com intuito de bloquear o ataque ou restaurar o sistema. Esse tipo de resposta pode ser efetuada pelo administrador ou automaticamente pelo sistema de detecção. As respostas passivas englobam as ações que simplesmente informam a ocorrência de uma intrusão.

As seções seguintes detalham cada um dos componentes do modelo, descrevendo as abordagens e técnicas utilizadas.

2.3 Fonte de informação

O primeiro componente do modelo é a fonte de informação. A fonte de informação provê o fluxo de dados que é analisado a procura de evidências de uma intrusão. Um esquema de classificação de IDS pode ser elaborado levando em conta o tipo de informação coletada para a análise. Enquanto alguns IDSs utilizam o tráfego da rede como fonte de informação, outros analisam os eventos gerados pelo sistema operacional ou por aplicações. Basicamente, levando em conta a fonte de informação, um IDS pode ser classificado em três tipos. Os baseados em máquina coletam dados internos de um computador, geralmente no nível do sistema operacional; os baseados em rede coletam informações da rede; e os baseados em aplicação coletam dados de aplicações específicas executando em um sistema.

2.3.1 Baseado em máquina

Os IDSs baseados em máquina operam com informações coletadas de um único computador. Esse tipo de coleta permite que IDSs analisem atividades com grande precisão e confiabilidade, determinando exatamente quais processos e usuários estão envolvidos em um possível ataque. Os sistemas baseados em máquina geralmente utilizam dois tipos de fonte de informação: registros de auditoria do sistema operacional e arquivos de registros de eventos, ou simplesmente arquivos de *log*.

Registros do sistema operacional são normalmente gerados pelo seu núcleo e, em geral, considera-se a melhor fonte de informação para esse tipo de IDS. A principal justificativa é que tais registros possuem um grande nível de detalhes, permitindo a detecção de certos

padrões que não seriam visíveis a um nível menor de detalhes. Além disso, o sistema operacional geralmente está estruturado para prover proteção aos registros gerados. Similarmente, arquivos de *log* são um conjunto de arquivos contendo registros de várias atividades do sistema. Um exemplo tradicional de uma aplicação que provê um esquema de registro de eventos é a aplicação *syslog* do Unix. A segurança desses registros é considerada mais fraca quando comparada aos registros de auditoria do sistema operacional. Entretanto, por não conter um nível tão grande de detalhes, são mais simples de serem analisados.

As vantagens dos IDSs baseados em máquina são descritas a seguir:

- Devido a habilidade de monitorar detalhadamente eventos em uma única máquina, esses IDSs são capazes de detectar ataques que não podem ser reconhecidos por um IDS baseado em rede.
- Os IDSs baseados em máquina podem, geralmente, operar em um ambiente onde o tráfego de rede é cifrado, uma vez que a informação manipulada é capturada antes da cifragem. Além disso, não é afetado pela topologia de rede.
- Quando os registros de auditoria do sistema operacional são utilizados como fonte de informação, a detecção de cavalos de tróia ou outros ataques que comprometam a integridade de aplicações é facilitada. Isso acontece pois esses comprometimentos constantemente resultam em inconsistências na execução dos processos.

Apesar disso, esse esquema também apresenta a seguintes desvantagens:

- Quando uma rede é envolvida na detecção, os IDSs baseados em máquina são mais difíceis de administrar, uma vez que a informação deve ser configurada e gerenciada em toda máquina monitorada.
- É comum que a informação seja extraída de uma máquina que é potencialmente um alvo de ataques, tornando o IDS também um possível alvo e comprometendo a detecção.
- Esse tipo de IDS, por só fornecer informações restritas a uma máquina, não é adequado para prover informações que permitam a detecção de sondagens e tentativas de ataques que envolvam um conjunto de máquinas, tal como uma varredura da rede.
- Nessa abordagem, o desempenho da máquina que abriga o IDS pode ser afetado, pois a quantidade de informações geradas pode ser muito grande, exigindo grandes capacidades de armazenamento e processamento.

2.3.2 Baseado em rede

A maioria dos sistemas de detecção de intrusão, com relação a sua fonte de informação, são baseados em rede[10]. Normalmente, esses IDS capturam e analisam o tráfego de rede envolvendo um conjunto de máquinas na detecção. Além disso, é comum também que esse tipo de IDS seja formado por um conjunto de sensores ou máquinas posicionadas estrategicamente na rede. Cada uma dessas unidades monitoram o tráfego da rede, realizam uma análise local e enviam a informação para um gerenciador central. As vantagens de um IDS baseado em rede são relacionadas a seguir:

- Um IDS baseado em rede bem posicionado é capaz de monitorar uma rede grande.
- A utilização desses IDSs resultam em um pequeno impacto na rede, pois geralmente a captura é realizada por dispositivos passivos que interceptam o tráfego sem interferir nas conexões.
- Um IDS baseado em rede pode ser projetado para ser seguro contra ataques, chegando a ser invisível para vários tipos de atacantes.

Suas principais desvantagens são:

- Tais IDSs podem apresentar dificuldades em capturar e processar todos os pacotes quando ocorre uma sobrecarga na rede, levando a perda de informações e, com isso, comprometendo a detecção.
- Esse esquema de detecção é diretamente dependente da topologia da rede. Podem, por exemplo, ser limitados em redes que utilizam *switches*.
- A análise pode ser prejudicada quando o tráfego da rede é cifrado.
- Muitos desses IDSs são capazes de relatar somente a ocorrência de uma tentativa de ataque e não sua efetivação. Nesses casos, cabe aos administradores a tarefa de investigar as atividades do possível ataque mais profundamente.
- Alguns IDSs baseados em rede apresentam problemas em lidar com ataques que envolvam a fragmentação de pacotes.

2.3.3 Baseado em aplicação

Os IDSs baseados em aplicação compõem um conjunto especial de sistemas de detecção baseados em máquina que analisam os eventos gerados por aplicações específicas em execução

no sistema. A fonte de informação mais comum utilizada por esse tipo de IDS são os arquivos de *log* gerados pelas aplicações. Em geral, os IDSs dessa categoria são restritos a um único tipo de aplicação, devido a especificidade das informações produzidas. Suas vantagens são:

- Devido ao nível de registro das atividades, tais IDSs são capazes de monitorar a interação entre um usuário e a aplicação, permitindo com mais facilidade a detecção de atividades não autorizadas.
- Podem executar em ambientes onde a informação é cifrada, uma vez que a interação acontece diretamente com as aplicações.

A maior desvantagem apresentada nesse esquema é que a fonte de informação é mais vulnerável a ataques quando comparado as fontes dos IDSs baseados em máquina, uma vez que os arquivos de *log* não possuem o nível de proteção proporcionado aos registros de auditoria do sistema operacional.

2.4 Análise

Em um IDS, depois do fluxo de eventos ser coletado na fonte de informação, eles são analisados a procura de atividades maliciosas. A análise, no contexto da detecção de intrusão, é a organização e caracterização dos dados e atividades do sistema com o intuito de identificar atividades suspeitas. Existem duas abordagens principais para análise dos eventos no processo de detecção: mau uso e anomalia. Na detecção de mau uso, a análise compara a informação de entrada com padrões estabelecidos de ataques conhecidos, também chamados de assinaturas. Essa técnica é utilizada pela maioria dos IDS comerciais. Já a detecção de anomalia estabelece perfis do que constitui o comportamento normal do sistema e caracteriza como ataque os padrões encontrados que não se enquadram aos padrões estabelecidos.

2.4.1 Detecção de mau uso

A detecção de mau uso analisa a atividade do sistema, procurando por eventos ou conjuntos de eventos que se igualam a padrões que caracterizam um ataque conhecido. Como esses padrões são comumente denominados de *assinaturas*, a detecção de mau uso também é conhecida como detecção de assinaturas. Em geral, essa abordagem gera um número menor de falsos positivos quando comparadas à detecção de anomalia. Além disso, por ser baseada em assinaturas, a detecção é mais rápida e específica, o que possibilita a especificação da técnica ou ferramenta utilizada, auxiliando a tomada das medidas preventivas.

Entretanto, esse tipo de detecção não é capaz de detectar ataques novos ou desconhecidos e, dependendo da especificidade das assinaturas, também apresenta dificuldade na detecção de variantes de ataques.

2.4.2 Detecção de anomalia

A detecção de anomalia caracteriza como ataque os padrões de comportamento considerados incomuns em um sistema ou rede. Para caracterizar um comportamento como anormal, são construídos perfis com base em dados coletados durante um período de operação normal. O IDS então utiliza um conjunto de métricas para determinar quando a informação monitorada difere dos perfis estabelecidos. A detecção de anomalia, por ser baseada no comportamento normal do sistema e não nas características específicas do ataque, como na detecção de mau uso, é capaz detectar ataques desconhecidos. Além disso, em alguns casos, as informações produzidas pela detecção de um ataque pode ser utilizada como assinatura na detecção por mau uso. Entretanto, a maior desvantagem dessa abordagem é que geralmente a análise produz um grande número de falsos positivos. Isso acontece, porque o comportamento normal de um sistema pode variar de acordo com o tempo através da adição de novos usuários e da utilização de novas aplicações.

2.4.3 Momento da análise

Outra modo de classificar os sistemas de detecção de intrusão pode ser realizado considerando o período transcorrido entre o acontecimento dos eventos capturados e a análise. Esse esquema, divide os IDSs em dois grupos: detecção em tempo real, e detecção em modo *batch*. Na detecção em modo *batch*, o fluxo de informação para o componente de análise não é contínuo. Ao invés disso, as informações são armazenadas e capturadas periodicamente pelo mecanismo de análise. Já nos IDSs que efetuam a análise em tempo real, o mecanismo é alimentado continuamente pela fonte de informação. Com isso, a detecção pode ser mais rápida, permitindo que as ações de resposta sejam executadas com o ataque ainda em andamento.

2.4.4 Um modelo do componente de análise

Rebecca Bace define em [11] um modelo de construção e execução para o componente de análise, generalizando os métodos utilizados na busca de evidências. Esse modelo divide o processo de análise em três fases: a construção do analisador, a análise propriamente dita e a realimentação ou refinamento do analisador. As fases são descritas a seguir.

Construção do analisador

O primeiro passo da construção é a coleta de informação.

- Detecção de mau uso: essa parte do processo envolve a obtenção de informações gerais sobre intrusões, incluindo informações sobre vulnerabilidades, ataques, ameaças e ferramentas de ataques.
- Detecção de anomalia: a informação é coletada do próprio sistema ou de um sistema similar. Essa informação é necessária para construção dos perfis que indicam o comportamento dos usuários.

Depois que a informação é obtida, ocorre o seu pré-processamento.

- Detecção de mau uso: o pré-processamento dos dados geralmente envolve a transformação dos eventos coletados em uma forma canônica de modo que permita a comparação com as informações providas pela fonte.
- Detecção de anomalia: os dados obtidos são transformados em vetores ou tabelas numéricas. De certa forma, como na detecção por mau uso, as informações são convertidas em uma forma canônica.

Com as informações transformadas em uma forma canônica, o analisador é construído, bem como sua base para a classificação. Isso é realizado através da seguinte forma:

- Detecção de mau uso: o analisador é construído em termos de regras de classificação, que utilizam como base para comparação as assinaturas dos ataques. Pode-se tomar como exemplo de um analisador de mau uso, um sistema especialista. Sistemas especialistas são compostos de uma base de conhecimento que contém descrições do comportamento suspeito baseado em intrusões conhecidas.
- Detecção de anomalia: o analisador é composto pelos perfis de comportamento, que são calculados a partir dos dados obtidos anteriormente. Com os perfis calculados, o analisador pode optar por várias técnicas para identificar as anomalias.

Realização da análise

Quando o analisador já está construído e preparado para a análise, pode ser dado início a sua realização, obtendo as informações fornecidas pela fonte de informação. O primeiro passo dessa etapa consistem em pré-processar os dados obtidos da fonte de informação.

- Detecção de mau uso: os dados são convertidos em uma forma canônica, correspondente a estrutura das assinaturas dos ataques.

- Detecção de anomalia: os dados são transformados em estruturas similares as estruturas dos perfis de comportamento.

Com os eventos transformados em um modo canônico, a comparação com a base de conhecimento é realizada:

- Detecção de mau uso: compara os eventos com a base de assinaturas.
- Detecção de anomalia: compara os eventos com os perfis de comportamento.

Se o analisador encontra alguma relação nos eventos com a base de conhecimento, então um ataque é detectado e uma resposta é disparada.

Refinamento do analisador

Alguns analisadores contém uma terceira fase que pode executar em paralelo com a análise. Essa fase trata da manutenção e aperfeiçoamento do analisador.

- Detecção de mau uso: inclusão de novas assinaturas na base de conhecimento.
- Detecção de anomalia: atualização dos perfis de comportamento.

2.5 Técnicas de análise

Diversas técnicas podem ser utilizadas para detectar uma intrusão. Em geral, essas técnicas são empregadas para construir um analisador com base nas duas abordagens principais de análise: a detecção de mau uso e a detecção de anomalia. Essa seção descreve as técnicas mais conhecidas e utilizadas na detecção de intrusão, que estão divididas em três classes: técnicas de detecção de mau uso, técnicas de detecção de anomalia e esquemas alternativos de detecção.

2.5.1 Técnicas de detecção de mau uso

A detecção de mau uso codifica eventos relativos a uma intrusão conhecida e então monitora um sistema em funcionamento procurando por ocorrências de tais eventos. Essa abordagem de análise garante uma maior confiabilidade, mas restringe-se somente a detecção de ataques conhecidos[62]. A seguir estão descritas as principais técnicas utilizadas na detecção de mau uso.

2.5.1.1 Sistemas de produção/especialistas

Uma das primeiras técnicas aplicadas na detecção de mau uso foram os sistemas de produção, sendo empregado em vários IDSs, tais como[8]: MIDAS, IDES, NIDES e DIDS. Nesse esquema, a assinatura de um ataque é construída utilizando regras do tipo `se-então`. As condições necessárias para a ocorrência de uma intrusão são especificadas na parte referente ao `se`. Quando essas condições são satisfeitas, as ações especificadas na parte do `então` são executadas.

A vantagem dessa técnica é a de que o funcionamento interno do sistema de produção é independente das declarações das regras de análise[62]. Essa característica permite que usuários introduzam regras referentes aos ataques sem o conhecimento do funcionamento interno do sistema de produção. Com as regras introduzidas, o sistema é capaz de analisar os eventos fornecidos pela fonte de informação.

Existem, entretanto, alguns problemas associados com a utilização de sistemas de produção na detecção de intrusão. Tais sistemas não suportam a manipulação de grande quantidade de dados. Além disso, não provêem um esquema para fixar uma relação de ordem nos eventos estabelecidos nas regras. Mais ainda, sua capacidade de detecção depende do conhecimento do usuário que introduz as regras. Por fim, a manutenção e adição das regras pode ser problemática, caso existam relações de dependência entre as regras.

2.5.1.2 Transição de estados

A técnica de transição de estados utiliza um conjunto de estados e transições para detectar intrusões. A detecção é iniciada em um estado que representa o funcionamento normal do sistema. As transições são estimuladas por ações que alteram o estado do sistema, e são definidas em função de usuários, processos, e dados presentes em um determinado instante. Depedendo das transições realizadas, um estado que determina a ocorrência de uma intrusão é alcançado. Diversas abordagens são utilizadas para empregar a transição de estados na detecção de intrusão, tendo duas delas um maior destaque. Na primeira, análise de transição de estados, os estados que compõem a intrusão formam uma cadeia que deve ser percorrida do início ao fim. Na segunda abordagem, os estados são caracterizados por uma estrutura mais genérica denominada rede de Petri.

Análise de transição de estados

A análise de transição de estados utiliza autômatos finitos para modelar intrusões. Nela, uma intrusão é composta de uma seqüência de ações que levam de um estado inicial a um estado de intrusão. O estado inicial representa uma instância do sistema antes da ocorrência dos eventos da intrusão e o estado final representa a instância do sistema

quando o ataque é efetivado. Uma instância do sistema é descrita em termos de atributos ou privilégios de usuários, sendo que suas ações estimulam as transições. Esse esquema foi explorado primeiramente no sistema STAT e em sua extensão USTAT[8] para o ambiente Unix, ambos desenvolvidos pela Universidade da Califórnia.

Esse esquema prove uma representação intuitiva e de alto nível. Além disso, utiliza, sempre que possível, o menor subconjunto de ações para caracterizar um ataque e com isso são capazes de detectar suas variações. Esse esquema de detecção possibilita a detecção de ataques coordenados e realizados de forma lenta. Apesar disso, é difícil modelar ataques mais sofisticados e, dependendo de seu tipo, essa abordagem não é eficaz.

Redes de Petri

Outra abordagem da transição de estados é a utilização de redes de Petri. Essa abordagem é implementada no sistema IDIOT[18], que utiliza uma variação dessa rede para representar padrões de intrusão. Nesse modelo, uma intrusão é representada como uma rede de Petri onde a cor dos *tokens* de cada estado modela o contexto de um evento. Do mesmo modo que no diagrama de transição de estados, a detecção parte de um estado inicial até um final. Essa abordagem é mais rápida se comparada a detecção utilizando o diagrama de transição de estados e permite a representação de restrições parciais na ordem dos eventos que caracterizam uma intrusão.

2.5.1.3 Recuperação de informação para análise em modo *batch*

Embora muitos dos IDSs utilizem o esquema de análise em tempo real, a análise em modo *batch* também é usada para procurar evidências em registros e arquivos. Essa abordagem é interessante para investigadores e equipes de respostas a incidentes que desejam conhecer as particularidades de uma intrusão. Anderson e Khattak da Universidade de Cambridge propõem uma distinção entre sistemas que detectam novas ocorrências de ataque e sistemas que permitem ao administrador encontrar evidências de instâncias de um ataque depois de sua identificação. Para resolver essa procura de evidências é proposta a utilização de um esquema de busca de informações[5], que procura por palavras chave ou combinação de palavras em arquivos. A abordagem proposta é bem simples e eficiente.

2.5.2 Técnicas de detecção de anomalia

A detecção de anomalia estabelece um conjunto de perfis com base no comportamento normal do sistema e compara os eventos fornecidos pela fonte de informação com os perfis definidos. Em geral, esses perfis são definidos como um conjunto de métricas, que são representações numéricas de comportamentos particulares de usuários ou do sistema. Esse esquema de análise assume que um usuário possui um comportamento previsível e uso

consistente do sistema. A seguir estão descritas as técnicas mais importantes utilizadas na detecção de anomalia.

2.5.2.1 Modelo de Dorothy Denning

Dorothy Denning descreve em [27], um dos artigos embrionários na área da detecção de intrusão, o modelo de um IDS. São propostos, para esse modelo, quatro esquemas estatísticos diferentes de detecção de anomalia, onde cada esquema considera um tipo diferente de métrica do sistema. Os modelos são descritos a seguir.

Modelo operacional

O modelo operacional adota algumas métricas para a detecção, como por exemplo o número de *logins* sem sucesso em um determinado período. As métricas são comparadas com um conjunto de limiares e uma anomalia é detectada quando algum limiar é excedido. Esse modelo corresponde a detecção por limiar (*threshold*), descrita posteriormente nessa seção.

Modelo de média e desvio padrão

O segundo modelo estatístico propõe a média e o desvio padrão para a classificação dos dados. Para a realização da análise, assume-se que o analisador conhece a média e o desvio padrão das métricas de comportamento estabelecidas. Um comportamento analisado é considerado anormal quando não se enquadra dentro do intervalo de confiança descrito pela média e o desvio padrão. Denning supõe que essa caracterização é aplicável a contadores, registros de intervalos e medidas de uso de recursos.

Modelo multivariado

Esse modelo é uma extensão do modelo anterior, porém ao invés de uma única métrica são adotadas duas ou mais métricas para a análise. Com isso, a análise realiza uma correlação entre as métricas para detectar uma intrusão.

Modelo do processo de Markov

Esse modelo é o mais complexo e tem sua aplicação limitada a contadores. Nele, para construir o perfil de comportamento normal, um detector considera cada evento como um estado variável. Além disso, para contabilizar as freqüências entre as variações, utiliza uma matriz de transição de estados. Uma observação é definida como anômala se sua probabilidade, determinada pelo estado anterior e pelo valor na matriz, é muito baixa.

Esse modelo permite ao analisador, ao invés de concentrar-se em um único, estabelecer uma relação entre eventos.

2.5.2.2 Análise quantitativa

A abordagem mais comum na detecção de anomalia é a análise quantitativa, onde regras de detecção são expressas em formas numéricas. Esse conjunto de técnicas geralmente envolve alguma computação, que pode variar desde uma simples soma até cálculos criptográficos complexos. O resultado dessas técnicas podem ser usado tanto para a detecção de mau uso ou como para a detecção de anomalia. A seguir estão descritos os métodos mais comuns.

Detecção de limiar

Provavelmente a forma mais comum de análise quantitativa é a detecção de limiar (*threshold*), onde certos atributos do usuário ou comportamentos do sistema são caracterizados em termos de contadores. Uma intrusão é encontrada quando esses contadores excedem um limiar estabelecido dentro de um intervalo de tempo. O exemplo clássico de um limiar é a quantidade máxima permitida de *logins* sem sucesso.

Detecção de limiar heurística

Essa forma é similar a descrita anteriormente, mas nesse caso os limiares são adaptados de acordo com observações de comportamento. Esse processo aumenta a exatidão da detecção, especialmente quando é realizada em um grande conjunto de usuários ou em um ambiente complexo.

Verificadores de integridade

Outra forma de análise quantitativa é a verificação de integridade, que procura por alterações não previstas em objetos do sistema. O exemplo mais comum de verificação de integridade é a utilização de uma função para calcular o *checksum* de um objeto do sistema. O Tripwire[57] realiza esse tipo de verificação.

Análise quantitativa e redução de dados

Os primeiros IDSs utilizavam esquemas de análise quantitativa para realizar a redução dos dados a serem analisados. A redução de dados é o processo de eliminação de informações supérfluas ou redundantes. Esse esquema reduz o consumo de recursos para armazenamento e otimiza o processo de detecção.

2.5.2.3 Medidas estatísticas

Os primeiros sistemas de detecção de anomalia eram baseados em medidas estatísticas. Essa técnica inclui as abordagens utilizadas nos sistemas de detecção IDES (e seu aperfeiçoamento, o NIDES)[2] e Haystack[93]. Os sistemas IDES/NIDES[2], desenvolvidos pela SRI International, foram os mais proeminentes dos primeiros sistemas de detecção. Eram sistemas híbridos, incluindo tanto a detecção de mau uso quanto a detecção de anomalia. As técnicas estatísticas empregadas nos sistemas criam perfis de comportamento para cada usuário ou componente do sistema. Esses perfis são atualizados periodicamente, para refletir mudanças de comportamento ao longo do tempo. Cada perfil expressa o comportamento normal para um usuário em particular em termos de um conjunto de métricas. Uma vez ao dia os perfis são atualizados baseados nas atividades dos usuários. Haystack[93] emprega duas abordagens estatísticas de detecção de anomalia. A primeira medida procura determinar o grau de similaridade entre uma sessão de um usuário e um tipo específico de intrusão. Na segunda, métodos estatísticos complementares detectam desvios nas atividades de uma sessão de um usuário, comparando com o perfil de comportamento normal.

A maior vantagem da técnica é que, quando bem projetada, não demanda uma atualização constante dos perfis. Por outro lado, a análise estatística também apresenta algumas desvantagens. Primeiro, muitas dessas abordagens são projetadas para realizar a análise em modo *batch*, eliminando a possibilidade de responder automatica e rapidamente com o intuito de bloquear o ataque. Segundo, esse esquema de análise não leva em conta as relações entre eventos na detecção. Terceiro, a taxa de falsos alarmes, falsos positivos ou negativos, para sistemas de análise estatística costuma ser alta.

2.5.2.4 Medidas estatísticas não paramétricas

As primeiras detecções estatísticas utilizavam abordagens paramétricas para caracterizar os padrões de comportamento. Nessas abordagens paramétricas eram realizadas suposições a respeito da distribuição dos dados analisados. O problema de tal suposições é que, quando eram formuladas incorretamente, as taxas de erros na detecção tornavam-se altas. Técnicas de análise estatísticas não paramétricas superam esse problema. Nela, os perfis de comportamento dos usuários são construídos com menos suposições e permitem ao analisador considerar métricas do sistema que não são facilmente acomodadas por esquemas paramétricos. Linda Lankewicz e Mark Benard foram os pioneiros na utilização dessa técnica[65], propondo o esquema de análise por *cluster*. O esquema coleta uma grande quantidade de dados e organiza em *clusters* de acordo com critérios de avaliação. Um algoritmo de clusterização é utilizado para agrupar as informações, convertidas em vetores, em classes de comportamento, procurando agrupá-las de modo que os membros

de cada classe sejam o mais semelhante possível. As atividades dos usuários são limitadas a duas classes distintas: anômala ou normal

2.5.2.5 Baseadas em regras

Outra variação da detecção de anomalia é a detecção baseada em regras. As suposições dessa abordagem são similares às suposições da análise estatística. A principal diferença é que nesse tipo de deteçcão são aplicadas regras para representar e armazenar os padrões de comportamento. O IDS *Wisdom and Sense*, descrito a seguir, utiliza essa técnica. O *Wisdom and Sense*[67], não projetado primeiramente para a detecção de intrusão, teve seu desenvolvimento iniciado em 1984, com a primeira publicação em 1989. Esse sistema possui duas funcionalidades principais. A primeira, é a parte do aprendizado (a parte *wisdom*) que realiza uma análise do histórico de dados e produz um conjunto de regras descrevendo o comportamento normal. A outra parte, da percepção (*sense*), utiliza as regras geradas pela primeira parte em um sistema especialista que procura por comportamentos anômalos, caracterizados como violações das regras.

2.5.2.6 Redes neurais

Sistemas de detecção baseados em redes neurais[15] utilizam técnicas de aprendizado para caracterizar o comportamento anômalo. Essa abordagem consiste na realização de um treinamento em um histórico de registros de eventos livre de sinais de intrusão. Uma rede neural é formada de um conjunto de elementos simples de processamento, chamados unidades, que interagem por meio de conexões ponderadas. O conhecimento de uma rede neural é codificado em termos das conexões ponderadas entre as unidades e o processo de aprendizagem é realizado através da mudança dos pesos das conexões ou através da adição e remoção de conexões. Um evento é então detectado como anormal através de alterações do estado das unidades, alterações do peso das conexões ou pela adição e remoção de conexões. Entre os problemas associados com a utilização dessa técnica estão a tendência à instabilidade na tentativa de aprender determinados tipos de comportamento e a falta de informações providas na detecção.

2.5.3 Esquemas alternativos de detecção

Bace ainda classifica alguns esquemas de detecção como alternativos, pois apresentam abordagens diferentes da detecção de mau uso e anomalia.

2.5.3.1 Abordagens do sistema imuno

Pesquisadores da Universidade do Novo México encararam a questão da segurança de computadores de um modo inovador, desenvolvendo um projeto que aplica os princípios

de defesa do sistema imunológico humano a detecção de intrusão[38]. A essência dos projetos conduzidos pelos pesquisadores está na capacidade de discernimento entre o *self* e o *nonself*, ou seja, discernir o que normal e o que é estranho no sistema. Para fazer essa distinção, o grupo utiliza como fonte de informação as chamadas ao sistema produzidas pelas aplicações em execuções[48, 102]. A detecção de intrusão baseada no sistema imunológico é descrita em mais detalhes no Capítulo 3.

2.5.3.2 Algoritmos Genéticos

Outra abordagem sofisticada é a utilização de algoritmos genéticos para a análise dos eventos. Um algoritmo genético é uma instância de uma classe de algoritmos denominada algoritmos evolucionários, que incorporam conceitos da seleção natural de Darwin. Os algoritmos genéticos utilizam formas codificadas, denominadas cromossomos, com métodos que permitem sua combinação ou mutação para formação de novos indivíduos. A classe é reconhecida por sua capacidade de tratar com problemas de otimização.

Algumas pesquisas aplicam algoritmos genéticos na detecção de intrusão. Para isso, definem vetores de hipótese para os eventos, onde cada vetor pode indicar uma intrusão ou um comportamento normal. Todas as hipóteses são testadas e as válidas são selecionadas. Em seguida, um novo conjunto de hipóteses é gerado com base nos resultados do teste. Esse processo se repete até uma solução ser encontrada.

2.5.3.3 Detecção baseada em agentes

Agentes são aplicações autônomas que realizam tarefas designadas por usuários[99]. Esse tipo de aplicação possui características interessantes para o projeto de sistemas distribuídos. Baseado nisso, algumas pesquisas tem adotado a tecnologia no projeto de sistemas de detecção. Os trabalhos que relacionam agentes móveis e detecção de intrusão são cobertos em mais detalhes no Capítulo 4.

2.5.3.4 Data Mining

Algumas pesquisas aplicam a técnica de *data mining* para a detecção de intrusão[66]. *Data mining* refere-se ao processo de elaboração de modelos a partir de um grande conjunto de dados para evidenciar fatos não aparentes em outros tipos de inspeção. O objetivo da aplicação dessa abordagem na detecção de intrusão é descobrir padrões consistentes a partir de um histórico de dados que possa ser utilizado para descrever o comportamento de usuários e aplicações de um sistema. Esses padrões são utilizados como base para a classificação de comportamentos anômalos.

Classificação	Quantidade de sistemas
Notificação	31
Resposta Manual	8
Resposta Automática	17
Total	56

Tabela 2.1: Classificação dos sistemas de respostas a intrusão (Março de 2000).

2.6 Resposta

O próximo passo após a detecção de um IDS é a resposta. Resposta a intrusão pode ser definida como a execução de ações contra os efeitos de uma intrusão[82]. Uma resposta, além de recuperar o danos do sistema, pode ser executada com o objetivo de conter o ataque. Por isso, a definição apresentada não leva em conta somente ações executadas após o encerramento do ataque, mas também ações tomadas quando o ataque ainda está em andamento. Uma resposta pode variar desde um simples alarme para o administrador até um conjunto de ações para bloquear um ataque em andamento.

Em geral uma resposta é disparada com os seguintes objetivos[40]:

- **Proteger os recursos do sistema**: A curto prazo, esse item inclui as ações de contenção da intrusão, além de ações de recuperação e restauração do sistema para um estado seguro. A longo prazo, procura-se aprender a respeito da intrusão e utilizar o conhecimento adquirido para remover as vulnerabilidades exploradas e estender as capacidades de detecção e resposta.

- **Identificar o responsável pela intrusão.**

Embora a maioria dos esforços de pesquisa sejam no processo de detecção, a funcionalidade de resposta também tem sido incorporada aos IDSs. Ragsdale classifica em [87] os sistemas de resposta em sistemas de notificação, respostas manuais e respostas automáticas, como mostrado na Tabela 2.1. Como pode ser constatado na tabela, a maioria dos sistemas são os de notificação, onde a resposta é realizada como um alarme ou um relatório. Os sistemas de resposta manuais provêem aos administradores a capacidade de iniciar uma reação partindo de um conjunto pré-estabelecido de ações. Respostas automáticas atuam no sistema imediatamente após a detecção, diminuindo o tempo de poder do intruso.

Dentre os sistemas de resposta automática relacionados na Tabela 2.1, dois sistemas de detecção possuem um esquema interessante de resposta automática: CSM[103, 8] (*Cooperating Security Managers*) e EMERALD[85, 74] (*Event Monitoring Enabling Responses to*

Anomalous Live Disturbances). Esses sistemas provêem um certo grau de adaptação nas respostas em tempo real e utilizam um sistema especialista para determinar a resposta apropriada, que é escolhida de acordo com o grau de suspeita do ataque. Cada grau de suspeita está associado com um tipo de resposta diferente.

Outra forma de classificação dos sistemas de resposta, apresentada em [11], divide as respostas a intrusão em duas classes não mutuamente exclusivas: respostas ativas ou resposta passivas. Nas respostas passivas, o sistema simplesmente notifica e reporta a ocorrência de um incidente, cabendo ao administrador tomar as medidas de contenção e recuperação necessárias. As respostas ativas englobam as ações tomadas de modo a bloquear e conter os efeitos do ataque. Esse tipo de resposta pode incluir, por exemplo, a coleta de informações a respeito do incidente ou o bloqueio do tráfego de determinados nós da rede.

2.6.1 Resposta ativa

Para iniciar um processo de resposta ativa é preciso observar um conjunto de fatores, que coletivamente identificam o contexto do incidente. A elaboração da resposta com base nesses fatores aumentam a chance da contenção do ataque. A consideração desses fatores é importante pois, se o alarme disparado pelo IDS for um falso positivo, a resposta pode afetar usuários legítimos causando uma negação de serviço. Como mostrado na Figura 2.1, o incidente é o evento principal para o início de um processo de resposta. Entretanto, outros fatores também devem ser considerados:

- **convicção**: quantas e quais são as evidências no sistema que sugerem a ocorrência de uma intrusão.
- **estado de alerta**: qual o estado do sistema, do processo e do responsável com relação aos eventos que originaram a intrusão.
- **severidade do incidente**: qual o impacto já causado pelo incidente com respeito a confidencialidade, integridade e disponibilidade do sistema e dos dados. Além disso, qual a necessidade de uma resposta nesse estágio.
- **impacto da resposta**: qual o impacto causado no sistema se uma resposta em particular for iniciada e como essa resposta pode afetar usuários legítimos caso seja um falso alarme.
- **alvo**: qual sistema, recursos ou dados aparentam ser o foco do ataque.
- **usuário envolvido**: se o ataque está sendo conduzido através de alguma conta de usuário, quais os privilégios relacionados a esse usuário.

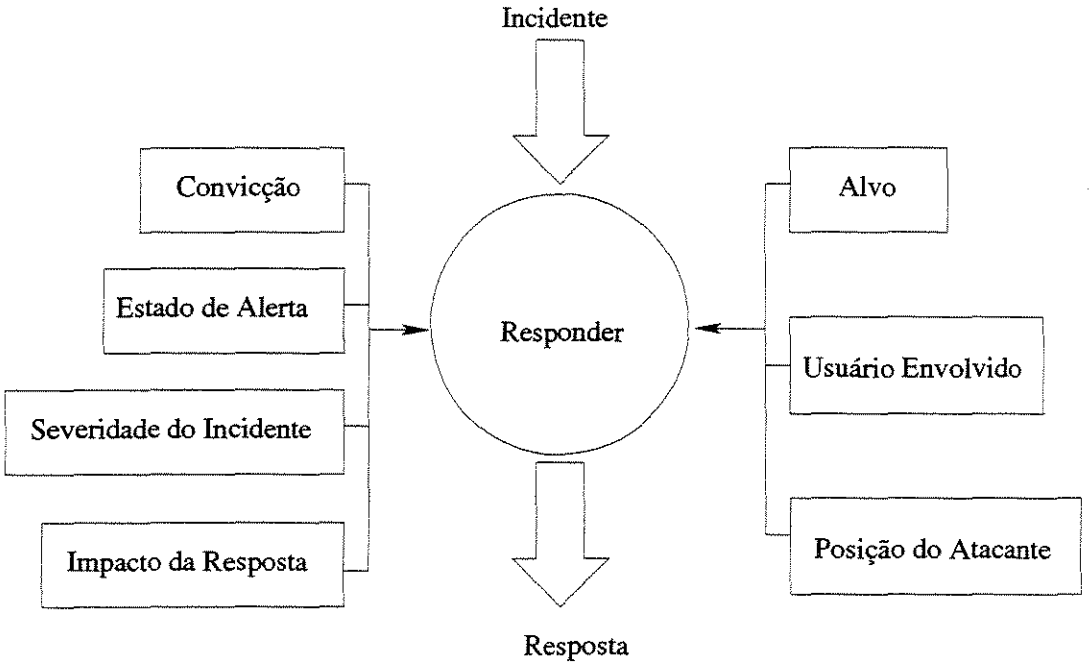

Figura 2.1: Fatores que influenciam um processo de resposta.

- **posição do atacante**: qual a localização do responsável pelo ataque com relação ao sistema atacado, ou seja, quais são as indicações das evidências quanto ao atacante estar interna ou externamente.

O centro da Figura 2.1 mostra um elemento denominado *responder*. Esse elemento é o responsável pela avaliação dos vários fatores, de modo a selecionar e invocar as ações de resposta. Na prática, a função do *responder* pode ser assumida tanto pelo administrador ou por um mecanismo de resposta automático. A designação desse papel ao administrador pode ter alguns limites práticos[82]. Primeiro, a administração de uma infraestrutura grande e complexa pode tornar essa tarefa extremamente difícil. Segundo, a adoção de *scripts* na geração do ataque pode tornar esse tipo de resposta ineficiente. Um mecanismo de resposta automático pode diminuir a chance de sucesso dos ataques, pois, se projetado e implementado corretamente, é capaz de executar ações de contenção logo que o ataque é encontrado. A resposta automática é discutida a seguir.

2.6.2 Resposta automática

Quando um ataque é detectado por um sistema de detecção de intrusão, é importante que se possua um conjunto de medidas de contenção contra o mesmo. Com esse conjunto de

medidas objetiva-se evitar futuras ocorrências do ataque e também impedir seu progresso, caso ainda esteja em andamento. Além disso, é interessante que tais medidas sejam executadas o mais breve possível, diminuindo o tempo em que o atacante explora o sistema atacado. A brevidade na execução das ações de contenção pode ser garantida através da utilização de um mecanismo de resposta automático. Entretanto,o tempo entre a detecção e o início da resposta não deve ser adotado como um critério único para o desenvolvimento de um mecanismo de resposta automático. Como já mencionado é necessário considerar diversos outros critérios para responder de modo a conter eficientemente o ataque e evitar que usuários legítimos sejam prejudicados com as ações de contenção.

Papadaki sugere em [81] que uma taxonomia de intrusões pode auxiliar na escolha de uma resposta apropriada e formar a base para a tomada de decisões em um mecanismo de resposta automático. Com isso, embora existam diversas taxonomias para intrusões[81, 68, 62], Papadaki *et al* desenvolveram uma taxonomia de intrusões sumarizada na Tabela 2.6.2 e descrita a seguir, direcionada para o desenvolvimento de respostas.

Taxonomia de intrusões direcionada para respostas

Uma taxonomia pode auxiliar um mecanismo de resposta a determinar os efeitos de uma intrusão em alvos específicos e influenciar no processo de tomada das ações. A taxonomia apresentada nessa seção divide as intrusões, de acordo com suas características, em cinco grandes classe: coleta de informação, falha de autenticação, comprometimento de *software*, *malware* e mau uso. Embora algumas classes estejam descritas em um alto nível, ainda é possível que os elementos de uma intrusão sejam abstraídos. As classes são descritas a seguir.

- **Coleta de informação**: Alguns ataques coletam informações sobre um alvo, identificando as vulnerabilidades existentes. Embora a coleta das informações não tenha um impacto sobre o sistema, o conhecimento conquistado pode ser posteriormente utilizado para a execução de outros ataques mais severos. *Probe, scan* e *sniff* são as intrusões que compõe essa categoria.

- **Falha de autenticação**: Para obter um acesso privilegiado específico, um processo ou usuário realiza sua identificação e autenticação. A fraude do processo de autenticação é um objetivo comum de atacantes, realizada de três modos principais: *masquerading, spoofing* e *bypassing.*

- **Comprometimento de *software***: As intrusões que exploram vulnerabilidades de *software* são incluídas nessa classe. Existem três principais categorias de vulnerabilidades dentro de um sistema que são vulnerabilidades de projeto, implementação

Incidente	Alvo	Resultado					
		Urgência	Severidade	Impacto			Incidentes potenciais
				C	I	D	
1 - Coleta de Informação							
Probe/ Scan	Servidor Externo	Baixa	Baixa	√		√	Spoof
	Servidor Interno	Média	Alta	√			Bypass
	Est. de Trabalho	Média	Média	√			Comp. de soft
	Comp. de rede	Baixa	Baixa	√		√	Malware
Sniff	Servidor Externo	Baixa	Baixa	√			Masquerade
	Servidor Interno	Alta	Alta	√			Bypass
	Est. de Trabalho	Média	Média	√			Comp. de Soft
	Comp. de rede	Média	Média	√			
2 - Falha de Autenticação							
Masquerade/ Spoof	Servidor Externo	Alta	Alta	√		√	Mau uso
	Servidor Interno	Alta	Alta	√			Malware
	Est. de Trabalho	Média	Média	√			Comp. de Soft
	Comp. de rede	Alta	Alta	√		√	
Bypass	Servidor Externo	Alta	Média	√			
	Servidor Interno	Alta	Alta	√			Mau uso
	Est. de Trabalho	Alta	Média	√			Malware
	Comp. de rede	Alta	Média	√			
3 - Comprometimento de *Software*							
Buffer Overflow	Servidor Externo	Alta	Alta		√	√	Bypass
	Servidor Interno	Alta	Alta		√	√	DoS
	Est. de Trabalho	Alta	Média		√	√	Mau uso
	Comp. de rede	Alta	Média		√	√	Malware
Flood/ DoS	Servidor Externo	Alta	Alta			√	Spoof
	Servidor Interno	Alta	Alta			√	
	Est. de Trabalho	Média	Média			√	
	Comp. de rede	Alta	Alta			√	
4 - *Malware*							
Trojan Horse	Servidor Externo	Alta	Alta	√	√	√	Bypass, Malware
	Servidor Interno	Alta	Alta	√	√	√	Mau uso
	Est. de Trabalho	Alta	Alta	√	√	√	Comp. de Soft
	Comp. de rede	Alta	Alta	√	√	√	Coleta de Inf.
Virus/ Worm	Servidor Externo	Alta	Alta	√	√	√	Mau uso
	Servidor Interno	Alta	Alta	√	√	√	Malware
	Est. de Trabalho	Alta	Alta	√	√	√	Comp. de Soft
	Comp. de rede	Alta	Alta	√	√	√	Coleta de Inf.
5 - Mau uso							
Alteração não Autorizada	Servidor Externo	Alta	Alta		√	√	Malware
	Servidor Interno	Alta	Alta		√	√	
	Est. de Trabalho	Alta	Média		√	√	
	Comp. de rede	Alta	Alta		√	√	
Acesso não Autorizado	Servidor Externo	Alta	Baixa	√			
	Servidor Interno	Alta	Alta	√			Malware
	Est. de Trabalho	Alta	Média	√			Alteração
	Comp. de rede	Alta	Baixa	√			não Autorizada

e configuração[11]. Os principais incidentes incluídos nessa categoria são o *buffer overflow* e *denial of service*.

- ***Malware***: É a classe caracterizada pelas intrusões conduzidas por aplicações maliciosas (*malwares*). Os três principais tipos de *malware* são os vírus, *trojan horses* e *worms*.

- **Mau uso**: O mau uso é relacionado ao uso não autorizado ou inaceitável dos recursos do sistema. De certo modo, esse termo é genérico o suficiente para englobar todos os tipos de ataque, pois uma intrusão é definida como uma violação da política de segurança[62]. Entretanto, essa categoria abrange as intrusões que realizam o mau uso de arquivos ou dados dentro de um sistema após a obtenção de acesso não autorizado.

A taxonomia, além de identificar e classificar as possíveis classes de ataque, também menciona que é importante considerar e classificar os possíveis alvos de uma intrusão, uma vez que o impacto de um ataque está diretamente relacionado com seu alvo. Com isso, a taxonomia divide os possíveis alvos de uma intrusão nos seguintes grupos:

- **Servidores externos**: São servidores acessíveis a redes externas e representam a imagem pública de uma organização, por exemplo servidores de ftp, *web* e DNS.

- **Servidores internos**: São servidores acessíveis somente dentro da rede interna de uma organização.

- **Estações de trabalho**: São computadores utilizados pelos usuários de uma organização.

- **Componentes de rede**: Alguns equipamentos da infraestrutura de uma rede, tal como o roteador, a *switch* e o *firewall* também são alvos de ataques e são inseridos nesse grupo.

Além das classes de ataques e seus possíveis alvos, outra característica que deve ser considerada para a elaboração da resposta são os resultados de uma intrusão, que a taxonomia expressa pelos seguintes itens:

- **Urgência**: Cada ataque possui um período singular de duração e é preciso considerar qual a disponibilidade de tempo para a execução da resposta. Baseado nisso, a urgência relata a necessidade do início de um processo de resposta, refletindo de modo parcial a velocidade do ataque.

- **Severidade**: Esse item relata a magnitude ou a extensão do ataque. Quanto mais severo é um ataque, maior é a necessidade de contenção, buscando a eliminação do impacto e da ameaça introduzida no sistema.
- **Impacto**. O impacto expressa uma avaliação do comprometimento da segurança do sistema atacado. Essa avaliação utiliza a definição de segurança que analisa a confidencialidade, integridade e disponibilidade[90] para quantificar o comprometimento do sistema.
- **Incidentes potenciais**. Esse item relata se algum outro tipo de incidente futuro é facilitado como conseqüência do ataque.

A taxonomia classifica a urgência e a severidade através de níveis (baixo, médio e alto). O impacto é descrito através da relação das condições (confidencialidade, integridade e disponibilidade) envolvidas. Os possíveis incidentes futuros são relacionados com base no primeiro item da taxonomia que relaciona as classes de ataques.

Por fim, depois de definidas as classes dos possíveis ataques, alvos e resultados, a taxonomia faz uma relação dos itens definidos. A relação, sumarizada na Tabela 2.6.2, associa, para cada tipo de ataque, os resultados causados em cada um dos potenciais alvos.

Problemas com a resposta automática

Embora a resposta automática tenha vantagens óbvias relacionadas com seu uso, também introduz um risco para a segurança através da execução de ações automáticas incorretas. Invocar automaticamente métodos inapropriados podem resultar em uma ação insuficiente contra o ataque ou, no pior caso, interromper ou negar serviços para usuários legítimos. Geralmente, esse risco ocorre quando um falso positivo é detectado pelo IDS e o sistema executa ações de contenção contra eventos legítimos do sistema. O impacto das respostas para esses casos pode ser suavizado através da utilização de métodos adaptativos de resposta que, ao invés de responder estaticamente a todas as intrusões, deve prover um esquema inteligente de defesa que contenha o ataque efetivamente, ou até que o administrador possa tomar um papel ativo de defesa[16].

2.7 Ferramentas que complementam IDS

Existem ferramentas que complementam os sistemas de detecção de intrusão e que são geralmente rotuladas como tal, uma vez que apresentam funções similares. Essa seção descreve três dessas ferramentas: sistemas de análise de vulnerabilidades, verificadores de integridade de arquivos, *honey pots*.

- **Analisadores de vulnerabilidades**: essas ferramentas realizam testes para determinar se uma rede ou uma máquina é vulnerável a ataques conhecidos, representando um caso especial da detecção de intrusão. Como os testes de busca no sistema são geralmente exaustivos, também permitem que administradores encontrem problemas causados por erros de configuração ou por falhas da política de segurança.

- **Sistemas verificadores de integridade**: o verificadores de integridade geralmente utilizam *checksums* criptográficos para detectar diferenças e mudanças em arquivos ou objetos do sistema. A utilização de tais ferramentas é importante, pois é comum que em um ataque ocorra alterações em arquivos do sistema, por exemplo para a instalação de *backdoors* que garantem um meio de acesso futuro ao atacante. Além da utilidade citada, os verificadores de integridade são importantes para a realização de um processo de forense computacional em um sistema atacado. Um exemplo de um verificador de integridade é o Tripwire[57].

- **Honey Pots**: *honey pots* são pseudo-sistemas projetados para atrair um atacante, evitando a invasão no sistema verdadeiro. Em geral, os *honey pots* são implantados para impedir que um atacante acesse o sistema, para coletar informações sobre os eventos de um ataque e para encorajar o atacante a permanecer no sistema durante um tempo suficiente para uma resposta.

2.8 Conclusão

Sistemas de detecção de intrusão são sistemas que automatizam o processo de monitoramento e análise dos eventos de um sistema ou de uma rede em busca de problemas de segurança. Um sistema de detecção sozinho não é capaz de garantir a segurança de uma organização, ao invés disso é parte de uma infraestrutura composta por outros mecanismos e técnicas de segurança. Entretanto, é um mecanismo fundamental para tornar a segurança de um sistema computacional mais robusta. O presente capítulo apresentou as técnicas e conceitos empregados nos sistemas de detecção de intrusão. Para a organização e descrição das técnicas adotadas, foi apresentado um modelo genérico de um IDS dividido em três componentes básicos, sendo que cada componente é detalhado ao longo do capítulo. Os componentes são: fonte de informação, análise e resposta.

Capítulo 3

Imunologia Computacional

O sistema imunológico tem sido empregado como modelo para o desenvolvimento de aplicações em diversas áreas da computação[26, 25]. 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 [17]. Já a conexão entre imunologia e detecção de intrusão teve início em 1994 com as publicações [35, 56], desencadeando uma série de outros trabalhos.

Esse capítulo descreve a relação entre o sistema imunológico humano e a detecção de intrusão. Para tal, a Seção 3.1 descreve sucintamente as estruturas e o funcionamento do sistema imunológico humano. A Seção 3.2 apresenta algumas pesquisas que aplicam os conceitos do sistema imunológico na detecção de intrusão e resposta.

3.1 Sistema imunológico humano

É impossível entender o sistema imunológico como modelo de um sistema de segurança de computadores sem antes compreender seu funcionamento. Nesta seção são apresentadas as estruturas básicas do sistema imunológico humano e as etapas da resposta imunológica.

3.1.1 Organização estrutural

O sistema imunológico é dividido em sistema inato e sistema adaptativo [88]. O sistema inato é caracterizado por sua natureza congênita e por sua capacidade limitada de diferenciar um agente patogênico de outro, reagindo de maneira semelhante contra a maioria dos agentes infecciosos. Além disso, 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, como a pele e os ácidos gástricos; as proteínas do sangue, incluindo as proteínas

complementares e outros mediadores de inflamação; e as células conhecidas como fagócitos (macrófagos, monócitos e neutrófilos), responsáveis pela eliminação de partículas estranhas.

Em contraste com o sistema inato, o sistema adaptativo é capaz de identificar especializadamente um determinado agente patogênico, permitindo uma resposta mais eficiente. Além disso, ele é capaz de "memorizar" um agente infeccioso e responder mais vigorosamente a novas exposições a esse micróbio. Os componentes do sistema adaptativo são os linfócitos (linfócitos T e linfócitos B) e seus produtos, como os anticorpos.

Os mecanismos de ambos os sistemas, inato e adaptativo, constituem um sistema integrado de defesa em que um grande número de células e moléculas agem cooperativamente [26]. O sistema inato não só provê a primeira linha de defesa, mas também atua em diversas etapas da resposta do sistema adaptativo.

3.1.2 Conceitos básicos

O corpo humano conta com um sistema de defesa que é capaz de garantir sua sobrevivência mesmo diante de vírus ou bactérias potencialmente mortais. Para conseguir isso, o sistema de defesa dispõe de alguns componentes com funcionalidades distintas que, unidos, compõem um sistema extremamente eficiente e robusto.

A tarefa básica atribuída ao sistema imunológico é o de distinguir o normal (ou *self*) do estranho (ou *nonself*) e eliminar o que for estranho. Como *self* são consideradas as células e moléculas do corpo, e como *nonself* qualquer material estranho, por exemplo vírus e bactérias. A seguir estão descritos sucintamente os componentes mais importantes do sistema imunológico humano.

3.1.2.1 Defesas em várias camadas

A arquitetura do sistema imunológico é uma arquitetura multi-camadas, contendo esquemas de defesas em alguns níveis. Desse modo, mesmo que um agente patogênico supere alguma camada, não assegura que o corpo contrairá alguma patologia. As camadas de proteção podem ser divididas como segue:

- **barreiras físicas**: essa camada é composta pelas barreiras físicas impostas pelo corpo. A pele funciona como um escudo de proteção contra invasores. O sistema respiratório também auxilia na defesa, através dos pêlos nasais e mucos secretados.

- **barreiras fisiológicas**: essa camada é formada por substâncias secretadas e outros fatores fisiológicos que dificultam a sobrevivência dos agentes infecciosos no corpo. Fluidos tais como a saliva, o suor e a lágrima contém enzimas destrutivas. Os ácidos estomacais exterminam a maioria dos microrganismos ingeridos junto com a comida

e a água. O pH e a temperatura do corpo apresentam condições desfavoráveis de vida para alguns invasores.

- **sistema imunológico inato e sistema imunológico adaptativo**: essa camada é composta por um conjunto de células, descrito na seção seguinte, que atuam cooperativamente para proteger o corpo humano contra doenças. Os sistemas inato e adaptativo são descritos na Seção 3.1.1.

3.1.2.2 Células do sistema imunológico

O sistema imunológico conta com um conjunto de células para iniciar e conduzir um processo de detecção e resposta a um agente patogênico. A seguir estão descritas as duas classes de células mais relevantes: os linfócitos e os fagócitos.

Linfócitos

Os linfócitos são as células responsáveis pela habilidade do corpo de distinguir e reagir especificamente a um grande número vírus e bactérias. Existem dois tipos principais de linfócitos: os linfócitos B e os linfócitos T, também chamados de células B e T.

Os linfócitos B têm seu nome originado da medula óssea (*bone marrow* no inglês), que é onde acontece seu processo de maturação. Cada célula B é programada para produzir anticorpos que atuam como receptores. Os anticorpos desempenham um papel importante tanto na detecção como na resposta imunológica, pois são proteínas específicas que reconhecem e se ligam aos antígenos, as proteínas que compõem bactérias. Desse modo, quando um agente infeccioso entra no corpo, ele é confrontado com uma vasta quantidade diferente de linfócitos B e anticorpos.

As células T são assim denominadas devido ao seu processo de maturação que ocorre em um órgão linfático chamado timo. As funções desse tipo de linfócito incluem o controle das ações de outras células do sistema imunológico e o ataque direto a células infectadas. Para tal, as células T são divididas em quatro subclasses: T ajudante, T citotóxica, T supressora e T de memória. As células T ajudantes são essenciais para a ativação das células B e dos fagócitos. As células T citotóxicas são capazes de eliminar micróbios, vírus ou células infectadas, injetando químicas nocivas capazes de perfurar a membrana da célula e causar sua destruição. A células T supressoras são vitais para o controle da resposta imunológica, inibindo a ação de outras células imunológicas quando necessário. Sem as células T supressoras, uma resposta imunológica perderia o controle resultando em reações alérgicas e doenças auto-imunes. Por fim, as células T de memória são linfócitos já ativados em alguma reação a um agente patogênico específico, e permitem que o sistema imunológico responda com mais rapidez e eficiência a futuras infecções causada pelo mesmo agente.

Fagócitos

Os fagócitos são as células brancas do sangue capazes de ingerir e digerir microrganismos e antígenos. Além disso, alguns fagócitos também possuem a habilidade de apresentação dos antígenos aos linfócitos, que consiste em expor fragmentos do antígeno em sua superfície depois de um processo de fagocitose. Os fagócitos mais importantes são os monócitos e os macrófagos. Os monócitos circulam através do sangue e migram nos tecidos, onde tornam-se macrófagos. Os macrófagos são as células do sistema imunológico que apresentam os antígenos às células T depois de digerí-los e, por isso, desempenham um papel importante para o começo de uma resposta imunológica.

3.1.2.3 Seleção Negativa

Os linfócitos são criados através de um processo aleatório que produz uma vasta variedade de células com receptores diferentes. Por ser aleatório, esse processo pode criar linfócitos que reconheçam organismos do corpo(*self*) como *nonself*. Para evitar isso, após serem criados, os linfócitos passam por um processo de maturação denominado seleção negativa. Esse processo de maturação ocorre no timo, para as células T, e na medula óssea, para as células B. Nele, as células imaturas são expostas a um grande número de células *self*. Se, nesse estágio, algum linfócito se ligar a qualquer célula *self*, ele é eliminado. Os linfócitos restantes desse processo, considerados maduros, são distribuídos através de todo o corpo humano para atuar no processo de detecção.

Memória adquirida

Uma resposta imunológica bem sucedida resulta na criação e proliferação de linfócitos específicos para o agente patogênico que deu origem a resposta. Tais linfócitos específicos, denominados células de memória, codificam a informação para a detecção desse agente patogênico ou outros agentes com estruturas similares. As células de memória capacitam o sistema imunológico, em encontros subseqüentes a esses agentes patogênicos, a reagir com maior rapidez e eficácia se comparada a primeira exposição do agente patogênico ao corpo. A primeira reação a um agente específico é chamada resposta primária. Já as reações subseqüentes ao mesmo tipo de agente são chamadas respostas secundárias. A Figura 3.1 exemplifica os processos de resposta primária e secundária para um determinado antígeno, apontando suas diferenças através da produção de anticorpos.

Embora as células de memória sejam desejáveis, os linfócitos presentes no corpo ainda não ativados também são fundamentais, pois são necessários para a detecção de novas ameaças.

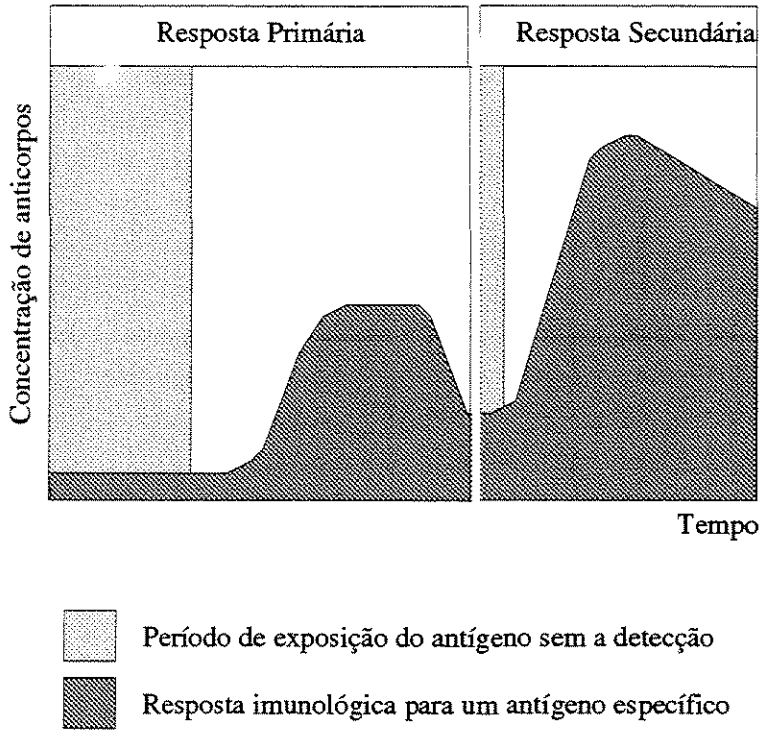

Figura 3.1: Respostas primária e secundária para um determinado antígeno

Coestimulação

Os linfócitos T, em sua fase de maturação, podem não ser expostos a todas as células *self*. Esse lapso cria a possibilidade de que algum linfócito T, ao passar à fase madura, cause uma auto-reação, ou seja, destrua células do corpo[75]. Entretanto, a chance de que uma auto-reação ocorra é muito pequena, pois para que uma resposta seja iniciada, além da ligação com o antígeno, uma célula T necessita de um sinal extra de estímulo. O sinal, conhecido como coestimulação, é geralmente um sinal químico produzido quando o corpo sofre algum dano. A coestimulação pode ser originada de células do próprio sistema imunológico ou de outras células do corpo.

3.1.3 Resposta imunológica

Os imunologistas tradicionalmente descrevem o problema resolvido pelo sistema imunológico como o problema de distinguir o normal (ou *self*) do estranho (ou *nonself*) e eliminar o que for estranho. O *self* é visto como as células e moléculas do corpo, e o *nonself* é qualquer material estranho, particularmente bactérias, parasitas e vírus.

A distinção entre *self* e *nonself* é uma tarefa bastante difícil por algumas razões [32, 76]. Primeiro, os componentes do corpo humano são construídos a partir da mesma matéria prima dos *nonself*, basicamente proteínas. O sistema imunológico detecta a presença de um agente patogênico através da percepção das proteínas desse invasor. Além disso, a quantidade de padrões diferentes de proteínas que o sistema imunológico deve reconhecer é muito maior que a capacidade do corpo humano de gerar esses padrões.

Uma vez que o agente patogênico transpôs as barreiras iniciais do sistema inato (a pele, por exemplo), inicia-se a resposta imunológica. O processo todo é baseado no reconhecimento de proteínas estranhas, encontradas na superfície dos agentes infecciosos, chamadas antígenos. Esse reconhecimento se dá pela reação das proteínas da superfície das células do sistema imunológico, chamadas receptores, com os antígenos dos invasores.

A reação entre os receptores e os antígenos é determinada por suas propriedades físicas e químicas, de modo que essa reação é altamente específica e cada receptor reconhece um conjunto limitado de antígenos estruturalmente relacionados [26]. A Figura 3.2 ilustra essa reação.

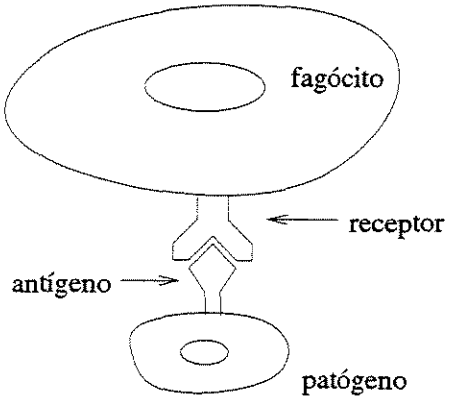

Figura 3.2: Reação entre receptor e antígeno.

A habilidade de reconhecer a maioria dos antígenos requer uma grande diversidade de receptores. Essa diversidade é parcialmente atingida pela geração de receptores através de um processo genético que introduz um fator aleatório [26]. Porém, essa geração aleatória pode resultar em receptores que reagem com proteínas *self*, causando problemas de auto-imunidade que é praticamente resolvida através da seleção negativa.

Uma vez na circulação, se o receptor de uma célula do sistema imunológico reage com antígenos, em uma concentração suficiente, um reconhecimento ocorre, disparando um conjunto complexo de eventos que leva à eliminação dos micróbios. Esse conjunto de eventos é denominado resposta imunológica, e pode ser dividido em três fases [26], como

segue.

Fase de detecção

Esta fase é iniciada com a detecção de microrganismos estranhos, por parte dos fagócitos presentes na corrente sanguínea e tecidos. Os fagócitos possuem uma capacidade intrínseca de reconhecer microorganismos estranhos [26], uma vez que são uma espécie de coletores de lixo responsáveis pela eliminação de partículas indesejáveis.

Ocorre ainda um processo chamado opsonização, onde uma série de proteínas complementares revestem a superfície dos micróbios, aprimorando a capacidade de detecção dos fagócitos. Esse aprimoramento se dá pela reação dos receptores dos fagócitos com essas proteínas. As proteínas complementares participam também no processo de inflamação, causando o aumento do suprimento de sangue e migração de outras células de defesa para o local da infecção.

O fagócito que detectou algum agente patogênico, reage ao invasor através do processo de fagocitose[1]. Com a dissolução do micróbio no interior do fagócito, este passa a produzir moléculas de proteína, chamadas de moléculas MHC (*Major Histocompatibility Complex*), contendo fragmentos dos antígenos do micróbio. Tais moléculas MHC são expelidas pela membrana plasmática dos fagócitos, ficando presas na superfície.

Fase de ativação do sistema adaptativo

Esta fase inicia-se com o processo de apresentação dos antígenos estranhos. Durante esse processo, os fagócitos envoltos por moléculas MHC "apresentam" os fragmentos de antígenos aos linfócitos T, onde estes reconhecem um antígeno específico nas moléculas MHC da superfície dos fagócitos. Os linfócitos T possuem receptores em sua superfície que reagem com um tipo específico de antígeno, permitindo a identificação precisa do agente patogênico.

Os linfócitos T, ao reagirem com os antígenos, passam a multiplicar-se, gerando quatro tipos de clones [26]: linfócitos T ajudantes, citotóxicos, supressores e de memória. Os linfócitos T ajudantes estimulam a ação dos fagócitos e a multiplicação dos linfócitos B, através da liberação de citozinas (proteínas). Os linfócitos T citotóxicos são capazes de identificar as células contaminadas, através das moléculas MHC com fragmentos de antígenos que tais células produzem, e destruí-las. Já os linfócitos T de memória guardam informações a respeito do agente patogênico em sua carga genética, permitindo uma resposta mais aprimorada a uma nova exposição a esse micróbio. Os linfócitos T supressores atuam no controle da reação, inibindo as células do sistema imunológico quando a infecção já está sob controle.

[1]Processo de englobamento [26].

Os linfócitos B, que possuem anticorpos específicos para o agente patogênico, também reagem com os antígenos, iniciando sua multiplicação e diferenciação em dois tipos de células: células B plasmáticas e de memória. As células B plasmáticas são responsáveis pela produção de anticorpos. Os linfócitos B de memória possuem a função semelhante ao linfócito T de memória.

Com esse processo de ativação, um exército especializado é recrutado para combater o agente infeccioso.

Fase de contra-ataque

Nesta fase, as células plasmáticas passam a produzir anticorpos que reagem especificamente com os antígenos dos micróbios, impedindo que estes contaminem outras células. Além disso, os anticorpos revestem a superfície dos micróbios permitindo a identificação com maior precisão pelos fagócitos.

Os linfócitos T citotóxicos passam a destruir as células infectadas e, conforme a concentração de antígenos estranhos diminui, os estímulos químicos são gradativamente inibidos, até que o contra-ataque chega ao fim.

3.2 Imunologia computacional

Nesta seção é discutida a analogia entre o sistema imunológico e segurança de redes de computadores.

3.2.1 Princípios do sistema imunológico

O estudo do sistema imunológico revela um conjunto de princípios organizacionais que podem servir de base para o desenvolvimento de um sistema de segurança [96]. Esses princípios são apresentados como segue.

Descentralização e localidade

No sistema imunológico, a ação de uma célula de defesa não é iniciada por algum mecanismo centralizador, mas quando alguma condição anormal é detectada pela célula. Essa característica descentralizada indica a inexistência de um ponto central de falha no sistema, garantindo uma maior robustez. Além de descentralizado, o sistema imunológico é capaz de agir em diferentes localidades paralelamente, não se fazendo necessária a mobilização de todo o "exército" para um determinado ponto de infecção.

Multi-camadas

Nenhum mecanismo do sistema imunológico garante isoladamente a segurança do

corpo. Ao invés disso, esses mecanismos constituem um sistema integrado de defesa em que um grande número de células e moléculas agem cooperativamente, cada qual exercendo uma função especializada.

Diversidade

O sistema imunológico é capaz de identificar uma grande diversidade de antígenos, proporcionando uma reação especializada contra a invasão de diferentes tipos de agentes patogênicos.

Robustez e tolerância a falhas

Nenhuma célula isolada do sistema imunológico é essencial, podendo ser substituída, caso seja infectada ou morta, sem comprometer o funcionamento do sistema. A disponibilidade de células combinada à ausência de um controle hierárquico caracteriza a robustez e a tolerância a falhas do sistema imunológico.

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 aprendizado na detecção de novos agentes patogênicos, armazenando a habilidade adquirida de reconhecimento em uma memória, conhecida como memória imunológica. A capacidade de reconhecimento especializa-se a cada agente patogênico similar reconhecido, tornando respostas futuras mais eficientes se comparadas às anteriores.

Auto-proteção

Qualquer célula do corpo humano pode ser atacada por um agente patogênico, incluindo as células que compõem o sistema imunológico. Sendo assim, os linfócitos podem proteger o corpo de outros linfócitos comprometidos por uma infecção.

Identificação por meio de comportamento

Em uma infecção, as células contaminadas são identificadas através das moléculas MHC com fragmentos de antígenos que tais células produzem, caracterizando um "comportamento anormal".

Tamanho do exército

O sistema imunológico regula o número de linfócitos de maneira a desenvolver um contra-ataque suficiente para a quantidade de agentes patogênicos.

3.2.2 Pesquisas em segurança de computadores

Forrest e Hofmeyr mencionam quatro classes de pesquisas que aplicam princípios e funcionalidades do sistema imunológico à segurança de computadores[34]: detecção de vírus, detecção de intrusão baseada em máquina, detecção de intrusão baseada em rede e resposta automática. As classes e seus respectivos trabalhos são descritos nas seções seguintes.

3.2.3 Detecção de intrusão baseado em rede

Os trabalhos a seguir utilizam conceitos do sistema imunológico para a construção de sistemas de detecção de intrusão baseados em rede.

LISYS

Hofmeyr e Forrest introduzem uma arquitetura de um sistema imunológico artificial chamada ARTIS(*Artificial Immune System*)[46]. Essa arquitetura, é aplicada à segurança de computadores na forma de um sistema de detecção de intrusão baseado em rede, denominado LISYS (*Lightweight Intrusion Detection System*)[45, 46, 47, 36, 13].

Os autores apontam o sistema de detecção como sendo distribuído, robusto, dinâmico, diverso e adaptativo. Tais princípios são justificados pelo fato do sistema ser uma cópia direta do esquema de detecção do sistema imunológico humano, que apresenta tais características. Esse sistema detecta conexões TCP não usuais na rede protegida pelo sistema. A definição de *self* é feita em termos de conexões TCP, determinadas pelos endereços IP de origem e destino, e pela porta TCP de serviço. Cada conexão é representada por uma cadeia de 49 *bits*, sendo que o *self* é formado pelo conjunto das conexões que normalmente são observadas na rede.

O sistema apresenta um único tipo de detector, que combina propriedades de várias células do sistema imunológico, assumindo diferentes estados. Cada detector é representado por uma cadeia de 49 *bits* e um conjunto de estados. A geração dos detectores é aleatória, passando pelo processo de seleção negativa. A detecção é realizada através da comparação com sucesso de *bits* r-contíguos.

A Figura 3.3 resume o ciclo de vida de um detector. Inicialmente, todo detector é criado de modo aleatório e, então passa por um processo de seleção negativa. Nesse estado, onde permanece em estado imaturo, se o detector apresentar semelhanças com alguma cadeia de bits representando uma conexão válida uma única vez, ele é substituído. Caso

sobreviva ao processo de seleção negativa, o detector passa ao estado maduro e é usado para monitorar o sistema.

Um detector maduro possui um tempo de vida finito. Ao final de sua vida ele é substituído por um novo detector, a menos que tenha ultrapassado um limite mínimo de semelhanças com conexões ilegítimas. Nesse caso, torna-se um detector de memória. Se o detector estiver ativo e não receber alguma forma de coestimulação (do administrador de segurança, por exemplo), ele morre. Entretanto, se houver coestimulação, torna-se um detector de memória, com um tempo de vida indefinido, necessitando de apenas uma semelhança para ser ativado.

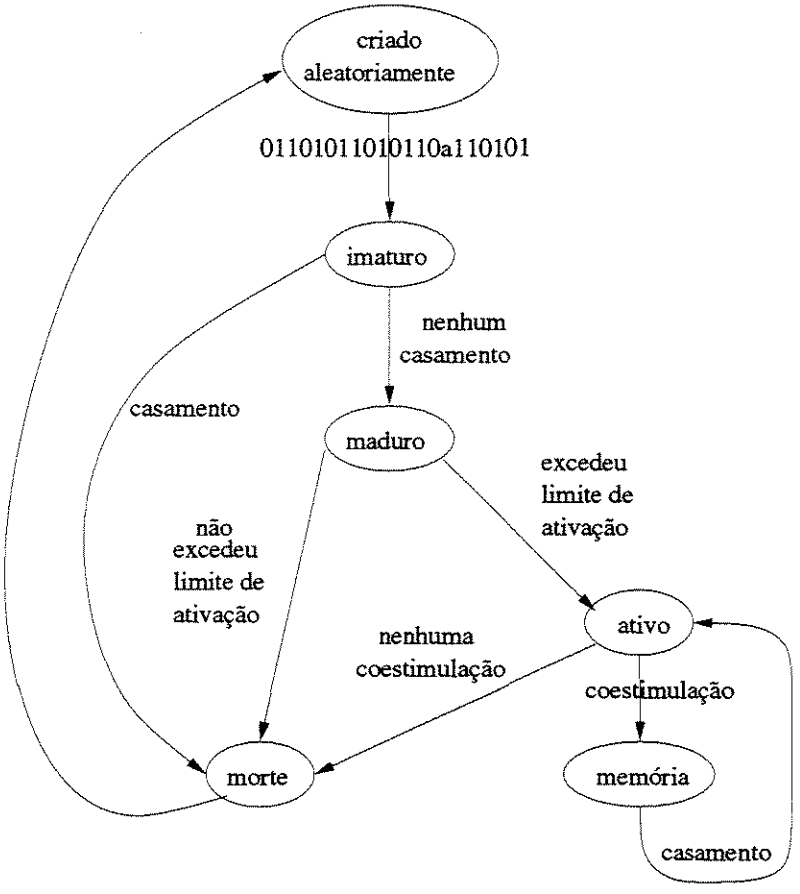

Figura 3.3: Ciclo de vida de um detector.

Kim e Bentley

Kim e Bentley, em [60], notaram que o sistema imunológico atende a algumas condições necessárias para um sistema de detecção de intrusão de redes. Com isso, baseados no trabalho de Forrest e Hofmeyr, também desenvolveram um sistema de detecção de intrusão baseado em rede[59, 58, 61].

A arquitetura do sistema é composta de um IDS primário e um IDS secundário. O IDS primário constrói os detectores através de dois processos evolucionários: evolução da biblioteca de genes que obtém informações de detectores efetivos, e seleção negativa que gera detectores e seleciona quais são aptos para o monitoramento. O IDS secundário realiza a detecção com um número limitado de detectores. Esse IDS clona os detectores que possuem bom desempenho, transformando-os em detectores de memória e alimentando a biblioteca de genes do IDS primário.

O sistema possui algumas semelhanças com o LISYS que inclui: a distribuição dos detectores entre vários nós da rede, o uso de detectores de memória e um mecanismo para controlar o limiar do sistema de detecção de anomalia, conforme a taxa de comparações com sucesso cresce. Diferente do LISYS, entretanto, a geração dos detectores é centralizada em um único nó, e o reconhecimento de uma anomalia depende da comunicação entre nós. Outra diferença é o método de geração dos detectores, que envolve um processo mais complexo quando comparada a geração aleatória do LISYS.

SANTA

O IDS SANTA (*Security Agents for Network Traffic Analysis*), proposto por Dasgupta em [19, 20], é um sistema que emula mecanismos do sistema imunológico para detectar anomalias de um modo distribuído. Para isso, o SANTA utiliza a tecnologia de agentes móveis que, no SANTA, são responsáveis por monitorar a rede em vários níveis do sistema, como o tráfego de rede, os processos, os recursos e as atividades de usuários procurando por possíveis correlações dos eventos monitorados com padrões de anomalia. As anomalias encontradas são reportadas ao administrador, sendo também possível que o sistema execute ações para conter a violação de segurança.

A arquitetura do IDS é composta por um conjunto de agente móveis que cooperam entre si. Essa arquitetura divide-se em três níveis: agentes de ação/decisão, agentes de comunicação e agentes monitores. Os agentes monitores patrulham os nós da rede procurando por anomalias. Os agentes de decisão/ação, baseados nos linfócitos T do sistema imunológico, dividem-se em agentes ajudantes que reportam a situação do sistema para o usuário, agentes citotóxicos que tomam ações de contenção (finalizar processos, por exemplo) nas ocorrências de intrusões, e agentes supressores que revogam ações tomadas por outros agentes. Por fim, os agentes de comunicação, correspondentes às citozinas

secretadas pelos linfócitos T para regular a resposta do sistema imuno, servem como negociadores e transportam mensagens.

3.2.4 Detecção de intrusão baseada em máquina

O trabalho a seguir aplica os conceitos do sistema imunológico para a construção de um sistema de detecção de intrusão baseado em máquina.

Monitoramento de processos

Forrest, Hofmeyr e Somayaji apresentam em [38, 37] um método de detecção de intrusão, baseado no sistema imunológico humano. A definição de *self* é feita em termos de seqüências curtas de chamadas ao sistema executadas por processos privilegiados, provendo uma assinatura compacta do comportamento normal dos processos[38]. Posteriormente, esse método de detecção é utilizado em outros trabalhos[37, 48, 102, 95, 94].

O método proposto apresenta dois estágios. No primeiro estágio, são extraídos traços de comportamento normal dos processos e a base de dados de padrões normais é construída. No segundo estágio, as seqüências de chamadas ao sistema executadas pelos processos são comparadas com os padrões armazenados na base de dados. Se uma seqüência não é encontrada na base de dados do processo, é reportada a ocorrência de uma possível invasão.

Para construir a base de dados, é obtido um registro de chamadas ao sistema da aplicação a ser analisada durante um período de funcionamento normal. Esse registro é então percorrido por uma janela do tamanho de k chamadas. Ao percorrê-lo, são adicionadas, na base de dados que define o comportamento normal, todas as seqüências de chamadas ao sistema de tamanho k contidas no registro.

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

Com a base de dados construída, um algoritmo similar é empregado para realizar a detecção. Esse algoritmo procura por seqüências de chamadas ao sistema de tamanho k, nas chamadas produzidas pelo processo monitorado, que não estão inclusas na base de dados. As anomalias são determinadas com base nessas seqüências encontradas.

3.2.5 Detecção de vírus

Outra aplicação dos conceitos do sistema imunológico pode ser encontrada em algumas pesquisas de detecção de vírus de computadores. As pesquisas estão descritas como segue.

Algoritmo de detecção de mudanças

Forrest relatou que o problema de detectar vírus em computadores poderia ser visto como uma instância do problema de distinguir o *self* do *non-self*[35]. Como um vírus altera o arquivo infectado, essa pesquisa propôs um algoritmo de detecção de mudanças, baseado no algoritmo de seleção negativa, como um método de autenticação de arquivos no problema da detecção de vírus.

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

1. Gere um conjunto de detectores que falham em reconhecer o *self*;
2. Use os detectores para monitorar os dados protegidos;
3. Sempre que um detector for ativado (reconhecer algo), uma mudança deve ter ocorrido em um dado protegido, e a localização da mudança é conhecida.

O *self* é definido como um conjunto de cadeias de *bits* de mesmo tamanho, geradas pela segmentação dos dados protegidos. Cada detector é também uma cadeia de *bits* de tamanho igual ao *self*. O reconhecimento por parte dos detectores é modelado como o "casamento" entre pares de cadeias. Um "casamento" perfeito entre duas cadeias de mesmo tamanho indica que os símbolos são idênticos em cada posição da cadeia. Entretanto, o algoritmo utiliza um tipo especial de "casamento", chamado bits r-contíguos (mais plausível com o modelo imunológico). Desse modo, o algoritmo busca por r posições contíguas, cujos símbolos são idênticos, em um par de cadeia de *bits*.

O algoritmo proposto é probabilístico e regulável, pode ser distribuído e é capaz de detectar vírus ainda não identificados. Entretanto, incorpora uma definição estática de *self*, não tolerando mudanças legítimas no sistema proporcionadas pelas atividades e novos comportamentos dos usuários que, por exemplo, alteram arquivos e executam novos processos. Além disso, o custo da geração dos detectores é exponencial com relação ao tamanho do conjunto de *selfs*.

De modo a superar a principal desvantagem da estratégia descrita, D'haeseleer *et al.* propuseram dois algoritmos geradores de detectores que executam em tempo linear em relação ao tamanho da entrada[28]. Os algoritmos executam em duas fases e são baseados em programação dinâmica.

Abordagem alternativa

No sistema desenvolvido por Kephart[56], um conjunto de anticorpos para vírus ou *worms* previamente não encontrados é gerado para promover uma resposta mais rápida e forte para futura infecções.

Nessa abordagem, os vírus conhecidos são detectados por seqüências do seu código, chamadas de assinatura, e vírus desconhecidos são detectados por seu comportamento insólito. Para isso, um conjunto diverso de programas-armadilha são mantidos em áreas estratégicas na memória do sistema para capturar amostras de vírus. Essas armadilhas são projetadas para serem atrativas de modo a capturar os tipos de vírus que se alastram com facilidade.

Cada um dos programas-armadilha é examinado periodicamente, a procura de modificações em seu código. Se alguma modificação é encontrada, é quase certo que um vírus desconhecido está no sistema e cada uma das armadilhas modificadas contém uma amostra sua. Os programas infectados são processados por um extrator de assinaturas que extrai e desenvolve uma caracterização para o vírus. Além disso, também é pesquisado o modo como os vírus se anexam aos programas, para desenvolver métodos de reparação das aplicações infectadas. Uma vez que as assinaturas são construídas, elas são testadas em uma grande quantidade de programas legítimos para assegurar que não proporcionará falsos positivos.

Kephart também pesquisou sobre a minimização do risco de respostas auto-imunes, onde um *software* legítimo é identificado erroneamente como suspeito. Outras referências do autor sobre detecção e eliminação de vírus baseadas no sistema imunológico são [54, 55].

3.2.6 Resposta automática

Alguns dos IDSs baseados no sistema imunológico também são capazes de responder automaticamente a uma intrusão. Esse sistemas são descritos a seguir.

pH

O pH (*process homeostasis*)[95, 94] é um sistema de detecção de intrusão e resposta que utiliza o método de monitoramento de processos através da análise das chamadas ao sistema descrito nessa seção. Esse sistema monitora todos os processos em execução no computador, disparando respostas quando anomalias são detectadas. A resposta é realizada através do atraso ou do cancelamento da execução de chamadas ao sistema no processo onde foi encontrada a anomalia.

SANTA

O sistema de detecção de intrusão SANTA, descrito anteriormente, também possui uma forma de resposta automática. Sua resposta é executada por um tipo de agente de ação/decisão do sistema, os agentes citotóxicos. Esses agentes terminam a execução dos processos relacionados com a anomalia detectada. Essa ação é disparada quando o nível

de ameaça determinado por um controlador de lógica *fuzzy* é médio-alto ou alto. Uma vez que a execução do processo é finalizada, o agente reporta sua ação ao administrador.

3.3 Conclusão

O sistema imunológico humano é considerado um bom modelo de defesa, pois é capaz de garantir a sobrevivência de um indivíduo por um período considerável, mesmo diante de falhas. Esse modelo tem servido de inspiração para diversos projetos desenvolvidos na área de segurança de computadores. Grande parte destes trabalhos adota o sistema imunológico como fonte de inspiração para o desenvolvimento de sistemas de detecção de intrusão e, em geral, utilizam processo de seleção negativa para o desenvolvimento de sistemas de detecção de anomalia. Forrest e Hofmeyr mencionam quatro classes de pesquisas que aplicam princípios e funcionalidades do sistema imunológico à segurança de computadores: detecção de vírus, detecção de intrusão baseada em máquina, detecção de intrusão baseada em rede e resposta automática. O capítulo apresentou sucintamente o funcionamento do sistema imunológico e suas principais características. Além disso, foram descritos os principais trabalhos que relacionam o sistema imunológico à segurança de computadores.

Capítulo 4

Agentes Móveis

Agentes móveis são aplicações inteligentes que locomovem-se entre máquinas de uma rede para cumprir tarefas designadas por um usuário[84]. Dentre o grande conjunto de tarefas possíveis de serem atribuídas, pode-se citar como exemplo: coleta e difusão de informações, realização de computações e alteração de agentes existentes em outros nós na rede. Entretanto, para que tarefas desse tipo sejam cumpridas, é necessária uma infraestrutura na rede que ampare a execução e transporte dos agentes.

Os agentes móveis possuem, dentre outras, duas propriedades principais que tornam seu uso atrativo: mobilidade e autonomia. Mobilidade é a habilidade da aplicação mover-se entre nós de uma rede, podendo continuar sua execução após o transporte. Autonomia é a capacidade de operar de maneira assíncrona e independente do seu processo criador. Tais propriedades propiciam a construção de sistemas distribuídos mais robustos e tolerantes a falhas[63].

Esse capítulo tem como objetivo fazer uma revisão da tecnologia de agentes móveis, descrevendo suas características, demonstrando seus benefícios e seu impacto no projeto de sistemas distribuídos. A Seção 4.1 apresenta os conceitos e definições básicas da tecnologia. A Seção 4.2 descreve os componentes de um sistema de agentes móveis. A Seção 4.3 descreve os paradigmas de computação em redes, comparando os tradicionais ao paradigma de agentes móveis. Alguns dos principais sistemas de agentes móveis são mostrados na Seção 4.4. A Seção 4.5 apresenta um esforço de padronização para a tecnologia. E, por fim, a Seção 4.6 aborda a aplicação de agentes móveis na detecção de intrusão.

4.1 Definições

Essa seção apresenta os conceitos e componentes básicos utilizados na tecnologia de agentes móveis. Para isso, a Seção 4.1.1 traz a definição de agente, descrevendo também as características de um agente móvel.

4.1.1 Agentes

Um agente pode ser definido como uma aplicação que atua em favor de uma entidade, cumprindo as tarefas designadas[63]. Essa definição enquadra-se bem na perspectiva de um usuário final e, embora esteja correta, é uma definição vaga. Uma definição mais completa pode ser elaborada utilizando as características comuns entre os diversos tipos de agentes existentes. Uma característica compartilhada por esse tipo de aplicação é a de que todos os agentes executam em uma plataforma. Além disso, tem a habilidade de interagir com sua plataforma de execução e de agir assíncrona e autonomamente sobre ela. Com isso, uma definição de agente, na perspectiva do sistema, pode ser considerada como uma aplicação situada em um plataforma de execução e que possui as seguintes propriedades:

- **reativo**: pode-se dizer que um agente é reativo pois age de acordo com um modelo de estímulo-resposta dependente de mudanças no ambiente, e não de acordo com um modelo pré-definido de atuação[78]. Os estímulos de uma reação podem ser, além de mudanças no ambiente de execução, mensagens enviadas por outros agentes.

- **autônomo**: controla suas ações e independe de outros agentes para tomá-las. Em [33], um agente autônomo é definido como sendo um agente que está ativo durante todo seu tempo de execução, tornando o usuário ciente da ocorrência de novos eventos ou buscando informações de interesse.

- **contínuo**: possui a habilidade de manter sua identidade e estado ao longo do tempo, independentemente da condição ou estado da plataforma de suporte[33].

As propriedades citadas são requeridas para caracterização de uma aplicação como um agente. Entretanto, um conjunto de outras propriedades também podem ser encontradas em agentes, são elas:

- **comunicativo**: comunica-se com outros agentes.

- **móvel**: capaz de mover-se entre nós de uma rede para cumprir uma determinada tarefa.

- **adaptativo**: adapta-se de acordo com experiências anteriores. Um agente pode ser capaz de aprender e alterar seu comportamento de acordo com uma experiência.

4.1.2 Mobilidade

Mobilidade é uma propriedade ortogonal de agentes, isto é, nem todos os agentes são móveis. Agentes que não possuem tal propriedade são chamados de agentes estacionários

e executam somente no sistema onde sua execução é iniciada. Nesse caso, quando o agente precisa interagir com agentes ou informações localizadas remotamente, é utilizado um mecanismo que implementa alguns dos paradigmas tradicionais de computação em redes, como por exemplo *remote procedure call* (RPC)[50]. Os paradigmas de computação em redes são descritos na Seção 4.3.

Em contraste a um agente estacionário, um agente móvel não está ligado ao sistema onde inicia sua execução. Após ser criado, pode transportar seu estado[1] e código para outra plataforma de execução na rede, onde a execução prossegue. Para que um agente possa usufruir de sua mobilidade em uma rede, deve existir uma plataforma de agentes em cada máquina que deseja recebê-lo e executá-lo. A plataforma também é responsável pela troca de mensagens entre agentes.

A mobilidade capacita aos agentes serem tolerantes a falhas pois, diferente das aplicações que utilizam o paradigma cliente-servidor, não são feitas suposições sobre a disponibilidade de uma máquina em particular. Com isso, a falha de uma máquina específica, ou mesmo um conjunto de máquinas, não compromete o funcionamento de um agente. Mesmo que a máquina abrigando o agente falhe, pode ser possível prosseguir sua execução em outra máquina. Em suma, ao contrário do paradigma cliente-servidor, nenhuma máquina é necessariamente essencial.

4.2 Elementos de um sistema de agentes móveis

Um sistema de agentes móveis baseia-se em dois componentes fundamentais: o agente e a plataforma. É a plataforma que fornece o suporte para que o agente execute, provendo serviços como criação, finalização, transferência e comunicação. Os componentes básicos são descritos respectivamente nas seções 4.2.1 e 4.2.2. A Seção 4.2.3 descreve o comportamento de um agente em função dos serviços disponibilizados pela plataforma.

4.2.1 Agente

Um agente móvel é uma entidade que possui cinco atributos: estado, implementação, interface, identificador e responsável. Quando um agente é transportado pela rede seus atributos são conservados e por isso permitem que a execução prossiga no destino. Os atributos são descritos a seguir.

- **Estado**: O estado de um agente pode ser definido como o conjunto de informações que descrevem sua execução em um determinado instante. Quando um agente é

[1]Entende-se por estado os atributos dos agentes que permitem o prosseguimento da execução em seu destino.

transportado pela rede, seu estado também é transportado permitindo o prosseguimento da execução no destino. Infelizmente, algumas linguagens de programação não possibilitam a captura e o transporte do estado em sua forma exata. Nesse caso, uma solução é transportar uma aproximação do estado que ainda capacite o retorno da execução no destino. Essa aproximação é bem apropriada para agentes implementados em Java, pois geralmente não são capazes de acessar e capturar seu estado de execução[63]. Isso acontece porque para uma aplicação Java não é provido acesso a informações da pilha, e ainda que fosse, as diferentes representações da pilha nas máquinas virtuais Java tornariam impraticável a tarefa de transferir o estado de execução completo de um agente em um ambiente heterogêneo.

- **Implementação**: Como qualquer outro programa de computador, um agente móvel necessita do seu código para executar. Quando o agente se move, existem duas opções a serem tomadas concernentes ao código. A primeira, é transportá-lo junto com os outros atributos do agente. A segunda, é não transportar o código e, caso o destino não possua, é possível requisitá-lo remotamente, utilizando por exemplo o paradigma de código sob demanda.

- **Interface**: Um agente provê uma interface que permite sua interação com outros agentes ou com o ambiente. Essa interface pode ser modelada, por exemplo, por um conjunto de métodos acessíveis externamente.

- **Identificador**: Todo agente possui um identificador único durante a sua execução. Identificadores são importantes principalmente para a localização dos agentes, necessária em eventos como troca de mensagens ou locomoção.

- **Responsável**: O responsável é um atributo que identifica uma entidade utilizada como autenticação pelos sistemas que um agente tem acesso. O atributo geralmente consiste de um nome e possivelmente de outros identificadores. Pode-se mencionar pelo menos dois identificadores para o atributo: o autor da implementação do agente e o responsável legal e moral pelo comportamento do agente.

4.2.2 Plataforma

Agentes móveis necessitam de uma plataforma de execução, também chamada de ambiente, que suporte todo o seu período de execução, incluindo sua criação, término e migração. Além disso, a plataforma também deve prover ao usuário a capacidade de monitorar e controlar seus agentes. A seguir são descritas algumas características que uma plataforma de agentes móveis deve prover.

- **Suporte para execução de agentes.** Uma plataforma deve prover a capacidade de criação de agentes, levando em consideração os requisitos específicos do agente.
- **Suporte para gerência de agentes.** É necessário que uma plataforma permita aos usuários monitorar e controlar seus agentes. O controle requer no mínimos, os processos de criação e finalização. O monitoramento abrange principalmente o esquema de localização do agente, que devido a sua mobilidade e seu comportamento ativo, pode não ser uma tarefa trivial.
- **Suporte à mobilidade.** O suporte à mobilidade deve prover a transferência do estado e do código referentes a um agente.
- **Suporte para identificação única.** Os agentes devem possuir uma identidade, em geral atribuída pela plataforma, única em toda a rede.

Além das características descritas, outras características adicionais também podem ser providas pelas plataformas.

- **Segurança.** Existem muitos aspectos a serem tratados no que diz respeito a segurança de agentes. Alguns dos itens a serem levados em conta são: autenticação dos agentes ou da plataforma, controle de acesso a recursos e serviços, além da cifragem dos agentes em sua transferência. Esse item é tratado em mais detalhes na Seção 4.6.
- **Suporte a comunicação.** Algumas plataformas provêem a capacidade de comunicação entre agentes.

Além disso, assim como os agentes, a plataforma também possui um conjunto de atributos. que são apresentados a seguir[63].

- **Aplicação de suporte**: Geralmente, a plataforma sozinha não é capaz de atender a todas as requisições dos agentes. Por isso, necessitam de uma aplicação base que forneça esse suporte complementar, como acesso aos recursos do sistema. Para agentes móveis implementados em Java, podem ser consideradas como aplicações de suporte a máquina virtual Java e o sistema operacional.
- **Recursos**: As aplicações de suporte e a plataforma controlam o acesso aos serviços e recursos locais tais como rede, base de dados, processadores e memória, discos e outros serviços de *software* e *hardware*.
- **Localização**: A localização pode ser definida como a combinação do nome do ambiente e do endereço de rede da máquina.
- **Responsável**: Esse atributo é semelhante ao atributo homônimo do agente.

4.2.3 Comportamento de um agente

Essa seção descreve o funcionamento da criação, término, transferência e comunicação dos agentes.

4.2.3.1 Criação e finalização

Um agente é criado sobre uma plataforma. A criação pode ser tanto estimulada por um agente em execução na mesma plataforma ou também por um agente ou outro tipo de aplicação externos. Geralmente, no momento da criação, é permitido ao criador fornecer argumentos de inicialização. Três passos estão envolvidos nesse processo:

1. **Criação da instância e atribuição do identificador**. A classe, que especifica a interface e a implementação do agente, é carregada e o agente é instanciado. Em seguida, a plataforma atribui um identificador único para o agente.

2. **Inicialização**. O agente é inicializado com os argumentos providos pelo criador. Quando a inicialização é completada o agente já está corretamente instalado no ambiente.

3. **Execução autônoma**. Depois do agente devidamente instalado, dá-se início a sua execução, tornando-se independente de outros agentes.

Um agente pode terminar sua execução por iniciativa própria, por solicitação de outro agente na mesma plataforma ou, até mesmo, por agentes ou outras aplicações externas. Em geral, o fim da execução de um agente é dividida em dois passos. O primeiro é a preparação para o término, onde é permitido ao agente terminar sua tarefa em andamento. O segundo passo é a suspensão da execução, onde a plataforma suspende a execução do agente.

4.2.3.2 Transferência

O processo de transferência, assim como o início e término da execução, pode ser iniciado pelo próprio agente, por outro agente no mesmo ambiente ou por aplicações externas. São as plataformas de origem e destino que gerenciam a transferência. O processo descrito a seguir está dividido nas ações executadas pelos plataformas de origem e destino.

Origem

Antes, de ser dado o início do processo de transferência, o agente precisa obter a identificação da plataforma de destino. Uma vez que o destino está estabelecido, o agente

informa à plataforma local sua intenção de transferência que, ao receber a requisição, toma as seguintes ações:

1. **Suspensão do agente**. Primeiro, é permitido que o agente termine sua tarefa em andamento. Assim que a tarefa é finalizada, o agente é suspenso.

2. **Captura do estado do agente**. Nessa etapa, o estado do agente é capturado. Como exemplo, para agentes implementados em Java, pode ser citada a serialização que é o processo de criação de uma representação persistente do objeto do agente[6].

3. **Transferência do agente**. Uma conexão de rede com o destino é estabelecida e o agente é transferido.

Destino

Depois do agente ser devidamente autenticado, caso a plataforma adote algum método de autenticação, é dado início ao processo de recepção pelo destino. A plataforma então recebe os dados referentes ao agente, extrai seu estado nos dados e prepara a retomada da sua execução.

4.2.3.3 Comunicação

Agentes podem ser capazes de se comunicarem. Algumas vezes, a comunicação é possível até mesmo com agentes em plataformas diferentes de execução. Em geral, a comunicação pode ser realizada através de troca de mensagens entre agentes ou através da chamada de métodos com os parâmetros adequados. A comunicação através de troca de mensagens pode utilizar alguns dos seguintes esquemas.

- **Síncrono**. O esquema de resposta síncrono é o mais popular. Nele, uma mensagem é enviada e o agente tem sua execução bloqueada até que o receptor manipule e responda a mensagem, tornando o esquema bloqueante.

- **Semi-síncrono**. Esse esquema é não bloqueante. Nele, o agente obtém um manipulador de uma mensagem ao enviá-la que é utilizado para a verificação futura da resposta. Com isso, é permitido ao agente prosseguir sua execução.

- **Assíncrono**. Esse esquema não bloqueia a execução corrente. O agente simplesmente envia a mensagem e continua a execução sem aguardar uma resposta.

4.3 Paradigmas de computação em redes

A tecnologia de agentes móveis introduz um novo paradigma de computação em redes. Esse paradigma apresenta inovações se comparado aos paradigmas existentes, pois agrega novas características que auxiliam o projeto e desenvolvimento de sistemas distribuídos. Essa seção descreve os principais paradigmas de computação em redes[39]: cliente-servidor, código-sob-demanda, avaliação remota e agentes móveis. Em seguida é feita uma comparação dos paradigmas apresentados.

4.3.1 Paradigma cliente-servidor

Dentre os paradigmas mostrados, o paradigma cliente-servidor é o mais conhecido e utilizado[84]. Nele, uma máquina servidora oferece um conjunto de serviços que dão acesso a recursos. Portanto, os recursos e o código necessário para a execução do serviço estão contidos no servidor. Quando algum cliente deseja acessar um recurso através dos serviços disponibilizados, ele envia uma requisição ao servidor. A requisição informa qual serviço pretende ser utilizado, juntamente com um conjunto de argumentos. Como resposta, o servidor interpreta a requisição e executa o código correspondente, acessando os recursos envolvidos no serviço. Em geral, o serviço produz uma resposta que é entregue ao cliente através de mais algumas interações. A Figura 4.1 ilustra as interações realizadas no paradigma e mostra o tipo das informações manipuladas nesses eventos. Esse paradigma é amplamente empregado por diversas tecnologias. O RPC (*remote procedure calling*)[50], pode ser citado como exemplo.

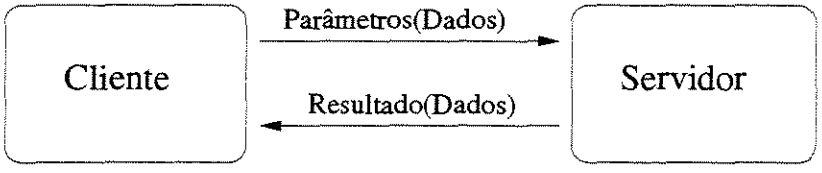

Figura 4.1: Paradigma cliente-servidor

4.3.2 Paradigma de avaliação remota

O paradigma de avaliação remota foi proposto por Stamos e Gifford como uma arquitetura alternativa para o paradigma cliente-servidor[97]. Nesse paradigma, o cliente possui o *know-how*, ou seja o código, para realizar um determinado serviço mas não possui os recursos necessários que, por sua vez, estão disponíveis no servidor. Quando o cliente

desejar realizar o serviço, ele envia o código ao servidor que então o executa e retorna o resultado ao cliente. A Figura 4.2 ilustra as interações entre o cliente e o servidor para o paradigma.

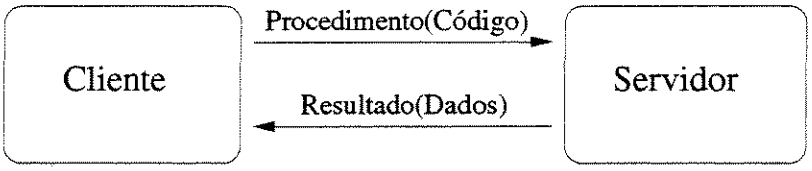

Figura 4.2: Avaliação Remota

4.3.3 Paradigma de código sob demanda

No paradigma de código sob demanda, o cliente possui os recursos necessários para realização de uma determinada tarefa, porém não possui o *know-how* para executá-la. Por isso, quando deseja executar a tarefa, o cliente interage com um servidor para obter o código necessário. O servidor então atende a requisição, permitindo ao cliente executar a tarefa. A Figura 4.3 ilustra as interações efetuadas no paradigma. *Applets Java*[31] são exemplos práticos desse paradigma, pois são requisitados nos servidores *web* pelos navegadores e então executados.

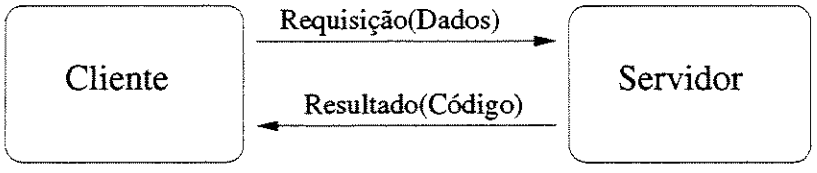

Figura 4.3: Codigo sob demanda

4.3.4 Paradigma de agentes móveis

A característica principal do paradigma de agentes móveis é a de que qualquer máquina pode abrigar o código, os recursos e a capacidade de processamento, descaracterizando o esquema de clientes e servidores utilizados nos paradigmas anteriormente descritos. Enquanto que os paradigmas de avaliação remota e código sob demanda focam a transferência de código entre máquinas, o paradigma de agente móveis move o componente computacional, juntamente com seu estado, código e alguns dados necessários.

No paradigma de agentes móveis, uma aplicação pode iniciar sua execução em um máquina, quando houver o *know-how* para tal. Se, em determinado momento, um conjunto de recursos amparados por outro nó da rede são necessários, a aplicação que executa a tarefa migra para o nó levando o *know-how* e possivelmente resultados intermediários para que possa continuar sua execução.

4.3.5 Comparação dos paradigmas

A Tabela 4.1 mostra as características de cada paradigma descrito. Com exceção do paradigma de agentes móveis, o paradigmas apresentados são estáticos com respeito a execução do código, pois uma vez criados, os componentes computacionais não mudam sua localização ou seu código durante a execução. Além disso, nos paradigmas tradicionais, uma vez iniciada a execução de uma aplicação, sua interação com recursos de uma máquina remota continua sendo remota até seu término. O paradigma de agentes móveis supera essa limitação provendo a mobilidade dos componentes computacionais.

Paradigma	**Local da execução**	**Iniciador**	**Armazenamento do código**	**Conteúdo transferido**
cliente/servidor	servidor	cliente	servidor	parâmetros
avaliação remota	servidor	cliente	cliente	código
código sob demanda	cliente	cliente	servidor	código
agentes móveis	rede	rede	rede	código/estado/dados

Tabela 4.1: Paradigmas de computação em rede.

Apesar disso, em [39] é afirmado que não existe o melhor paradigma em termos absolutos e que a escolha do paradigma a ser explorado depende do tipo específico da aplicação e de sua funcionalidade. Para cada caso, deve-se avaliar o comportamento da aplicação e os custos relativos a execução e comunicação e, baseado nisso, escolher o paradigma mais apropriado.

4.4 Sistemas de agentes móveis contemporâneos

Existem diversos sistemas de agentes móveis disponíveis atualmente. Muitos desses sistemas ainda estão sobre desenvolvimento e disponíveis para avaliação na Internet. A seguir estão descritos alguns dos sistemas de agentes móveis mais importantes.

4.4.1 Aglets Workbench

Java Aglets Workbench[63], desenvolvido pelo IBM Tokyo Research Laboratory, é um sistema de agentes móveis baseado na linguagem Java. No sistema, os agentes são chamados *aglets*, que é uma combinação dos termos *agent* e *applet*. A justificativa para a escolha dos termos é que o sistema espelha-se no modelo de *Applets* Java para o desenvolvimento das funcionalidades básicas do agente. A plataforma do Aglets Workbench possui duas interfaces, AgletProxy e AgletContext, que servem como base para os serviços necessários prestados aos agentes.

A interface AgletProxy atua como um manipulador de um agente, provendo um modo comum de acessá-lo. Em geral, um *aglet* possui diversos métodos públicos que, por questões de segurança, não deveriam ser acessados diretamente por qualquer agente. Com a interface AgletProxy, qualquer *aglet* que deseja comunicar-se com outro, precisa interagir através dessa interface. Outro papel importante do AgletProxy é que provê transparência com respeito a localização do agente representado pela interface. Já a interface AgletContext provê a interação com a plataforma de execução, permitindo que o agente utilize-a para obter informações, tais como o endereço da máquina onde está executando, ou até mesmo para criar um novo agente dentro da mesma plataforma.

A plataforma provê duas primitivas de migração aos agentes: *dispatch* e *retract*. *Dispatch* é a primitiva que realiza o transporte do agente para a plataforma de execução especificada como parâmetro. Esse esquema de migração é assíncrono e imediato. A primitiva simétrica *retract* procura um agente e o envia à plataforma de execução onde a primitiva foi invocada. Para construir uma representação do objeto de um agente, o *Aglets Workbench* utiliza o mecanismo de serialização provido pela linguagem Java. Além disso, um protocolo simples no nível de aplicação, chamado ATP (*Agent Transfer Protocol*), é utilizado no transporte dos agentes. O protocolo também dá suporte a comunicação entre agentes através da troca de mensagens, podendo ser tanto síncrona como assíncrona.

4.4.2 Mitsubishi Concordia

Desenvolvido pela Mitsubishi Eletric Information Technology Center, o Concordia[104] é um sistema de agentes móveis baseado em Java que apresenta mecanismos de segurança e confiabilidade. Esse sistema consiste de um conjunto de componentes que provêem algum tipo de serviço tais como comunicação, segurança, armazenamento persistente e administração. O componente responsável pela mobilidade dos agentes é o *Conduit Server*. Quando um agente deseja iniciar uma transferência, ele invoca um método provido pelo componente. Com isso, o *Conduit Server* suspende o agente e cria uma imagem persistente para ser transferida. Feito isso, o destino é contactado e a transferência é realizada. Esse sistema também utiliza o mecanismo de serialização de Java, *Java Object Seriali-*

zation, para auxiliar na transferência dos agentes. O modelo de segurança do Concordia provê suporte para cifragem dos agentes antes da transferência e também um esquema de controle de acesso aos recursos do sistema.

4.4.3 ObjectSpace Voyager

A ObjectSpace possui um pacote chamado Voyager[79], que possibilita a criação e controle de agentes implementados em Java. Os agentes são considerados uma classe especial de objetos que suportam um método de migração. A chamada do método, além de especificar o destino, especifica também o método a ser chamado quando o destino for alcançado, de modo que a execução do agente continue. Um mecanismo optimizado de serialização em Java é utilizado para a criação de uma imagem persistente dos agentes. O Voyager provê um esquema de comunicação entre objetos através de troca de mensagens, suportando tanto a troca síncrona como assíncrona. Além disso, permite que programadores criem aplicações de rede utilizando tanto os paradigmas tradicionais, quanto o de agentes móveis.

4.4.4 Ara

Ara[83] (*Agents for Remote Action*) desenvolvido pela Universidade de Kaiserslautern é uma plataforma para execução segura de agentes móveis em redes heterogêneas. Ara suporta agentes implementados nas linguagens Tcl, C e C++. Nela, os agentes podem migrar em qualquer ponto de sua execução, preservando completamente seu estado. Além disso, também é provido um esquema de troca de mensagens entre agentes.

Os agentes implementados em C/C++ são compilados em um *bytecode* chamado MACE. Quando um agente escrito em C/C++ deseja migrar, seu código e seu estado completo são transferidos para o destino, capacitando o retorno da execução. Além disso, o sistema Ara permite que um agente faça um *checkpoint* do seu estado interno durante sua execução. Diferente de outros sistemas que suportam múltiplas linguagens, o sistema Ara é *multi-threaded*. Com isso, o servidor e os interpretadores Tcl e MACE executam dentro de um único processo Unix e, embora essa abordagem dificulte a implementação, apresenta vantagens no desempenho e na comunicação. Os sistemas implementados em Java também são *multi-threaded*, apresentando as mesmas vantagens.

4.4.5 D'Agents

O D'Agents[42], anteriormente conhecido como Agent Tcl, suporta agentes implementados nas linguagens Tcl, Java e Scheme. Além disso também permite agentes estacionários escritos em C e C++. Assim como o Ara, o D'Agents provê uma instrução para migração e é capaz de capturar e restaurar o estado completo de um agente. Entretanto, diferente

do Ara, somente o servidor D'Agents é *multi-threaded* e cada agente é executado em um processo separado, o que simplifica a implementação, mas adiciona uma sobrecarga na comunicação entre os processos. A plataforma D'Agent possui um esquema de autenticação para os agentes e gerenciadores de recursos, onde cada gerenciador é responsável por um único recurso que é acessado com base na autenticação dos agentes e nas configurações do administrador.

4.4.6 Telescript

Telescript, desenvolvido pela General Magic[2], foi o primeiro sistema de agentes móveis comercial, e a inspiração para muito dos sistemas atuais. O sistema Telescript provê uma linguagem, um servidor e um protocolo de comunicação específicos. Um agente é implementado em uma linguagem orientada a objetos similar a Java e C++, e compilado em um *bytecode* que é interpretado por uma máquina virtual contida no servidor. A linguagem provê suporte à migração, operação e cooperação dos agentes, além da integração com aplicações externas. A migração no Telescript é amparada por um protocolo denominado PIP (*Plataform Interconnect Protocol*) que capacita tanto a codificação, como o transporte entre plataformas. Quando um agente chega a um ambiente, ele passa por um processo de autenticação e então tem suas permissões atribuídas, por exemplo a quantidade máxima de espaço em disco. Um agente que viola suas permissões é terminado imediatamente.

Odyssey

O Telescript, por ser baseado em uma linguagem e arquitetura de rede proprietárias, não obteve grande sucesso comercialmente. Em resposta ao sucesso da linguagem Java, a General Magic criou um novo sistema de agentes móveis chamado Odyssey. Esse sistema implementa, na linguagem Java, os conceitos utilizados no Telescript, resultando em uma biblioteca de classes que permitem a criação de agentes.

4.5 Padronização de agentes móveis

Essa seção descreve resumidamente os dois maiores esforços de padronização da tecnologia de agentes móveis. O padrão MASIF (*Mobile Agent System Interoperability Facility*) apresenta um esquema de interoperabilidade entre sistemas de agentes móveis e a organização FIPA (*Foundation for Intelligent Physical Agents*) conta com um conjunto de padrões que visa uniformizar as interfaces entre os diferentes sistemas de agentes.

[2]www.generalmagic.com

4.5.1 MASIF

O MASIF[71, 43] é o primeiro esforço de criação de um esquema de padronização para a tecnologia de agentes móveis, adotado como padrão em 1998 pelo Object Management Group. Seu objetivo é incentivar a utilização da tecnologia por meio da padronização, que propõe a interação entre plataformas heterogêneas. O MASIF é composto de uma coleção de definições e interfaces que provêem a interoperabilidade entre sistemas de agentes. Essa interoperabilidade abrange somente a interação entre plataformas, tais como interpretação, serialização e execução dos agentes, e não entre agentes e plataformas. Duas interfaces principais são definidas: *MAFAgentSystem* para transferência e gerência de agentes e *MAFFinder* para rotulação e localização de agentes. O padrão MASIF abrange quatro tópicos principais, descritos como segue:

- **Gerenciamento de agentes.** O MASIF define interfaces para criação, suspensão, continuação e término dos agentes, permitindo que um sistema controle agentes provenientes de outros sistemas.
- **Transferência de agentes.** O MASIF define métodos de recepção e busca de agentes. Isso possibilita que sistemas heterogêneos formem uma infraestrutura comum para o tráfego dos agentes.
- **Nomes dos agentes e sistemas.** A padronização da sintaxe e semântica dos nomes dos agentes e sistemas facilita o processo de identificação e localização. O MASIF suporta o rastreamento de agentes, que consiste em localizá-los em diferentes sistemas registrados no MAFFinder através dos nomes padronizados.
- **Nomes do tipo do sistema e da localização.** A transferência de um agente não pode ocorrer a menos que o sistema de destino seja capaz de prover o suporte necessário. A padronização dos nomes dos tipos de sistema auxilia essa verificação. Além disso, a padronização da localização auxilia transferência de agentes entre as plataformas.

4.5.2 FIPA

A *Foundation for Intelligent Physical Agents*[80] (FIPA) é uma organização sem fins lucrativos estabelecida em 1996. Seu propósito é desenvolver uma especificação de padrões para a tecnologia de agentes com o intuito de maximizar a interoperabilidade entre as tecnologias. A organização não procura padronizar a arquitetura interna dos agentes ou a forma de implementação. Ao invés disso, são especificadas interfaces necessárias para dar suporte a interoperabilidade. As especificações da FIPA abrangem quatro tópicos principais.

- **Gerência de agentes.** Inclue as interfaces para os serviços de suporte, criação e comunicação entre agentes, levando em conta a segurança e a mobilidade.

- **Comunicação de agentes.** Facilita a comunicação e interação entre agentes, tais como negociações, cooperação e troca de informações. Uma linguagem de comunicação de agentes (ACL) está definida para amparar essa interface.

- **Interação de agentes e outros tipos de aplicações.** Essa interface dá suporte a interação entre agentes e aplicações externas a plataforma, como por exemplo *drivers* dos dispositivos de *hardware.*

- **Interação de agentes e usuários.** Especifica como agentes devem interagir com usuários ou outros agentes que manipulam informações relacionadas ao usuário.

Para que haja cooperação entre agentes, é necessária uma linguagem que permita um alto nível de interação. Os padrões da FIPA fornecem um conjunto de funcionalidades por meio de uma linguagem de comunicação entre agentes denominada ACL (*Agent Communication Language*), juntamente com alguns protocolos de interação. Os protocolos de interação definem uma seqüência de mensagens que representam um diálogo entre os agentes, permitindo que cada um dos participantes responda de acordo com um padrão definido de interação.

A FIPA também propõe uma extensão da padronização MASIF com o intuito de formar uma estrutura unificada de agentes móveis.

4.6 Agentes móveis na detecção de intrusão e resposta

A tecnologia de agentes móveis é capaz de contribuir na área da detecção de intrusão, estendendo ou, até mesmo, adicionando novas funcionalidades aos IDSs. Essa contribuição deve-se, principalmente, às características do paradigma introduzido. Em [52], são mencionadas as vantagens trazidas pela tecnologia de agentes móveis na detecção de intrusão e resposta, que, entretanto, também pode introduzir vulnerabilidades, permitindo que o sistema de detecção seja atacado ou subvertido. As questões de segurança relacionadas com a tecnologia são um dos maiores obstáculos para sua ampla utilização[53, 99].

É importante notar que diversas ameaças introduzidas também surgem quando outros paradigmas de computação em redes são empregados. Porém, a escala das ameaças aumenta consideravelmente com a tecnologia, por oferecer uma grande oportunidade para abuso e mau uso. As novas ameaças devem-se ao fato de que, ao contrário dos outros

paradigmas, onde o responsável pela aplicação e o operador do sistema são os mesmos, o responsável pelo agente e o operador do sistema podem diferir.

4.6.1 Classificação das ameaças de segurança

Os componentes de um sistema de agentes móveis, plataformas e agentes, podem ser utilizados para classificar as ameaças de segurança para a tecnologia. Essa classificação define as classes de ameaças em função das possíveis origens e destinos de um ataque, resultando em quatro classes distintas: ameaças de agentes contra a plataforma, ameaças de agentes contra outros agentes, ameaças da plataforma contra agentes e ameaças de aplicações externas contra agentes. Os casos onde um agente ataca outro agente de plataforma remota ou uma plataforma atacando outra são cobertos pela última categoria, uma vez que tais ataques dependem da capacidade do canal de comunicação para explorar vulnerabilidades. A última categoria também inclui ataques ao sistema operacional.

Muitos sistemas existentes já contam com boas soluções para as ameaças descritas. Entretanto a ameaça de um servidor contra um agente ainda possui muitos aspectos não resolvidos e é considerado apenas por alguns sistemas e de forma parcial[100].

Agente contra a plataforma

O paradigma de agentes móveis requer que uma plataforma aceite e execute agentes cuja implementação é desconhecida. Tomando proveito disso, um agente pode utilizar algumas linhas de ataque. A primeira é obter acesso não autorizado a informações ou recursos providos pela plataforma. O acesso não autorizado pode ocorrer pela falta de mecanismos de controle de acesso adequados na plataforma. Além disso, mecanismos de identificação e autenticação fracos permitem a personificação, onde um agente tenta se passar por outro para utilizar recursos a qual não tenha acesso ou para não ser responsabilizado por ações que venha a realizar. Dependendo do nível de acesso, o agente pode até causar o término da execução da plataforma. Mesmo sem obter acesso não autorizado, um agente pode causar uma negação de serviço na plataforma, consumindo excessivamente os recursos computacionais.

Plataforma contra um agente

É óbvio que um agente está completamente exposto à plataforma que provê a base para sua execução. Por isso, tanto o seu código como dados podem tornar-se acessíveis à plataforma, permitindo que o ataque adote algumas abordagens diferentes. Uma plataforma pode personificar outro servidor para extrair informações, que pode ser realizada através do monitoramento das mensagens trocadas entre os agentes. Como um agente é muito

suscetível à plataforma, pode ter seu estado e código corrompidos ou modificados. A modificação pode ser realizada simplesmente pela falsificação das respostas de requisições feitas. Além disso, as requisições também podem ser recusadas, causando uma negação de serviço.

Agentes contra outros agentes

Esta categoria compreende os ataques em que um agente tenta explorar fraquezas na segurança de outros agentes presentes no mesmo servidor. Uma abordagem é a personificação para obter informações confidenciais, prejudicar o funcionamento de outro agente ou atribuir a responsabilidade de suas ações ao agente personificado. A negação de serviço pode ser causada, mesmo com mecanismos de controle razoáveis, através do envio de um grande número de mensagens a um agente. Além disso, um agente malicioso pode responder incorretamente a requisições de outros agentes ou até mesmo realizar um repúdio, negando a ocorrência de transações legítimas.

Outras entidades contra o sistema de agentes

Mesmo assumindo que agentes e plataformas sejam confiáveis, outras entidades podem tentar causar danos ao sistema de agentes. Os métodos mais óbvios envolvem ataques na comunicação entre agentes ou plataformas remotas através da personificação ou interceptação, permitindo o acesso não autorizado ou obtenção de informações confidenciais. Uma entidade pode monitorar o canal de comunicação, modificando e transmitindo agentes ou mensagens interceptadas. Ataques de negação de serviço também são possíveis realizando um mau uso dos serviços do sistema de agentes.

4.6.2 Agentes na resposta

A capacidade de alguns IDSs responderem automaticamente a ataques suaviza a carga atribuída aos administradores de sistemas, que é a de diagnosticar imediatamente as atividades suspeitas e tomar as respectivas ações corretivas. Entretanto, essa nova capacidade levanta questões relevantes quando a tecnologia de agentes é empregada. Quando os agentes são os responsáveis pela execução das ações, eles necessitam muitas vezes de privilégios administrativos para efetivar as respostas. Esse fator pode introduzir um risco na segurança do sistema, pois agentes maliciosos podem tentar a personificação dos responsáveis pela resposta.

Apesar das ameaças serem evidentes, elas podem ser corrigidas através de técnicas de segurança convencionais. Se um sistema de detecção restringe o processamento somente aos agentes assinados digitalmente pelo administrador de segurança, as vulnerabilidades

são consideravelmente reduzidas, uma vez que é difícil para o atacante alterar o código de um agente com o intuito de torná-lo malicioso. Entretanto, o atacante pode ser capaz de manipular os dados utilizados pelo agente e com isso levá-lo a executar ações maliciosas[53]. Técnicas adicionais podem ser aplicadas para combater os problemas mencionados. Tais técnicas incluem mecanismos de controle de acesso, métodos criptográficos para cifragem das informações trocadas, identificação e autenticação de usuários, agentes e plataformas e, por fim, mecanismos para realização de auditoria nas plataformas de agentes.

Algumas pesquisas que utilizam a tecnologia de agentes móveis para a execução de respostas à intrusão têm sido iniciadas. Tais pesquisas abordam diversos tipos de tarefas, como por exemplo o rastreamento automatizado do atacante, a coleta automatizada de evidências e a tomada de ações de contenção[53].

4.6.3 Trabalhos relacionados

Algumas das pesquisas que aplicam agentes na detecção de intrusão são mencionadas a seguir.

O AAFID (*Autonomous Agents for Intrusion Detection*) da Purdue University[12] emprega uma arquitetura hierárquica de agentes. Na raiz da hierarquia estão monitores, que provêem o controle global e realizam a análise da informação. Nas folhas estão agentes que coletam as informações relativas aos eventos. Os agentes são estáticos e situam-se em plataformas de propósito específico, chamados *transceivers*. Os *transceivers* realizam o controle dos agentes locais e análise ou redução do processamento da informação recebida.

O projeto JAM (*Java Agents for Meta-Learning*) da Universidade de Columbia[66] utiliza agentes inteligentes construindo um *data mining* distribuído. O projeto possui dois componentes principais: agentes locais que aprendem a detectar ataques e prover serviços de detecção de intrusão dentro de um sistema de informação, e um sistema de *meta-learning* que combina o conhecimento adquirido pelos agentes locais. Outro projeto é o da Iowa State University, similar ao JAM, que também envolve um IDS baseado na tecnologia de agentes inteligentes [44].

O sistema IDA (*Intrusion Detection Agent*)[7] atribui aos agentes móveis a tarefa de rastrear intrusos entre várias máquinas envolvidas em uma intrusão. O IDA concentra-se em eventos específicos que podem ser relacionados a intrusões, chamados MLSI. Se um MLSI é encontrado, o IDA captura a informação relacionada, analisa e decide se a existe a ocorrência de uma intrusão. O sistema segue uma estrutura hierarquica, com um gerenciador central na raiz e uma variedade de agentes nas folhas. Um agente sensor situa-se em um nó na procura de um MLSI, e quando descobre, notifica o gerenciador que envia um agente de rastreamento para a máquina. O agente de rastreamento inicia um

agente de coleta de informação e continua sua tarefa, procurando nós da rede identificados como suspeitos. As informações coletadas dos nós suspeitos são levadas ao gerenciador onde são analisadas.

4.7 Conclusão

O presente capítulo apresentou uma breve revisão sobre a tecnologia de agentes móveis. Essa tecnologia adota um novo paradigma de computação em redes que dispõe de características interessantes para o desenvolvimento de sistemas distribuídos. Existem, atualmente, diversos sistemas de agentes móveis. O capítulo descreveu alguns dos sistemas mais importantes, ressaltando as peculiaridades particulares de cada um.

A tecnologia tem sido aplicada em diversas áreas da computação, incluindo a segurança de computadores. Entretanto, uma das maiores barreiras para a ampla utilização de agentes móveis são as questões relacionadas à sua segurança. Tais questões ainda não foram satisfatoriamente solucionadas e fornecem, portanto, uma grande fonte de pesquisa.

Capítulo 5

Sistema Imunológico de Segurança

Grande parte das pesquisas que relacionam o sistema imunológico humano à segurança de computadores empregam o esquema de geração aleatória de receptores e o processo de seleção negativa[21] que, em geral, é utilizada no desenvolvimento de técnicas de detecção de anomalia. Basicamente, é produzida uma base de dados do que é considerado como normal no sistema e os receptores gerados aleatoriamente são testados. Receptores que não forem ativados com as entradas da base de dados são utilizados para monitorar o sistema. Durante o monitoramento, se algum detector é ativado, é um indicativo de uma anomalia.

Entretanto, além de empregar o sistema imunológico humano somente como modelo para desenvolvimento de sistemas de detecção de anomalia, também é possível extrair dele características de detecção de mau uso. Tais características são encontradas na memória imunológica, que pode ser considerada como a base de dados de assinaturas de antígenos já presenciados e considerados perigosos. Além disso, também é possível afirmar que os anticorpos e os receptores das células B formam as assinaturas dos antígenos específicos já encontrados pelo sistema imunológico. Os componentes imunológicos mencionados capacitam uma resposta mais eficiente a novas exposições de invasores conhecidos.

Adicionalmente, além de ser possível extrair características de detecção de mau uso e anomalia, outro conceito bem interessante que pode ser empregado em sistemas de detecção de intrusão é o fato do sistema imunológico ser capaz de, autonomamente, alterar sua base de dados de mau uso. Essa característica abrange o processo de criação e refinamento das células de memória, estimuladas pelas reações aos antígenos encontrados. A alteração da base de dados inclui tanto a criação de assinaturas para ataques que ainda não haviam sido presenciados e o aperfeiçoamento das assinaturas existentes.

Com base no que foi introduzido, o grupo composto por Fabrício Sérgio de Paula, Marcelo Abdalla dos Reis e pelo autor desta dissertação, sob orientação do professor Paulo Lício de Geus, propôs uma nova abordagem no desenvolvimento de sistemas de detecção

de intrusão baseados no sistema imunológico humano. Essa abordagem considera o fato de que o sistema imunológico possui diversas características de detecção de mau uso em adição às características de detecção de anomalia já exploradas nas pesquisas anteriores.

Este capítulo apresenta o modelo de um sistema de segurança baseado no sistema imunológico humano que é capaz de detectar e responder a ataques. O sistema é um refinamento do modelo apresentado em [22] e descreve um sistema híbrido de detecção de intrusão e resposta que agrega ainda a capacidade de aprendizado e adaptação do sistema imunológico, podendo reagir a ataques desconhecidos[24, 23]. Para tal, a Seção 5.1 apresenta a estrutura do modelo, descrevendo seus componentes. E a Seção 5.2 evidencia as analogias entre o sistema proposto e o sistema imunológico.

5.1 Modelo estrutural

Essa seção apresenta o modelo estrutural do sistema de segurança imunológico que utiliza uma arquitetura híbrida e adota, com isso, tanto a detecção de mau uso como a detecção de anomalia. No modelo, ambos sistemas de detecção funcionam concorrentemente, aprimorando as possibilidades de resposta. A Figura 5.1 ilustra o modelo, apresentando seus componentes e o fluxo de informação entre eles. Os componentes do modelo são detalhados a seguir.

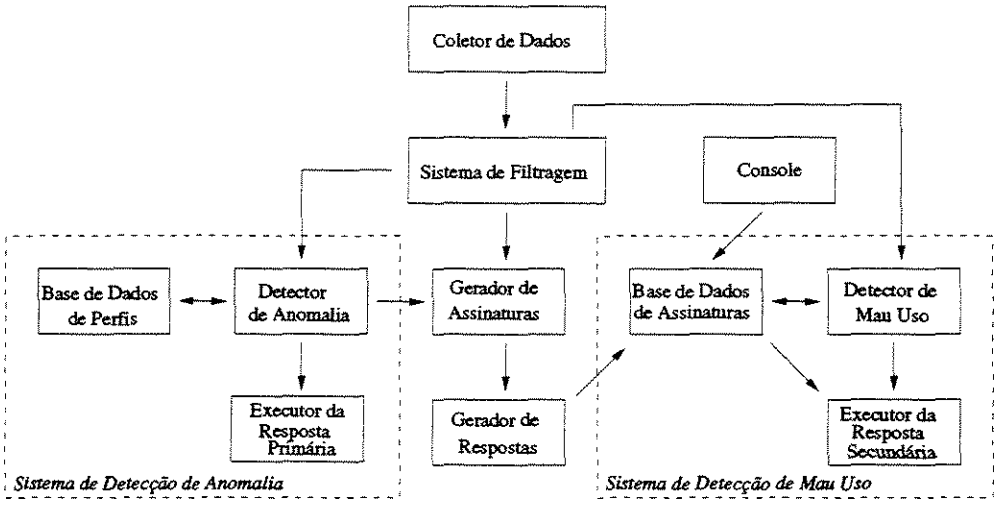

Figura 5.1: Modelo do IDS híbrido baseado no sistema imunológico.

5.1.1 Coletor de dados

O coletor de dados é responsável pela coleta de informações e por suprir o fluxo de eventos para o sistema de filtragem. A natureza da informação coletada pode variar de acordo com a estratégia adotada[1]: baseado em rede, máquina e aplicação[11]. O IDS proposto é aplicável para qualquer uma dessas estratégias.

5.1.2 Sistema de filtragem

O sistema de filtragem provê a redução dos dados de modo a identificar e remover informações redundantes e irrelevantes[11]. Depois de filtrado, o fluxo de informações é passado para os sistemas de detecção e, quando requerido, para o gerador de assinaturas.

5.1.3 Sistema de detecção de anomalia

A detecção de anomalia envolve um processo de estabelecimento de perfis que caracterizam o comportamento normal de um sistema. Os eventos ocorrentes são comparados a esses perfis à procura de desvios. Essa abordagem é capaz de acomodar adaptações de mudanças no comportamento ao longo do tempo, adicionando aprendizado e adaptabilidade para o IDS[11]. Os componentes do sistema de detecção de anomalia são descritos a seguir.

Base de dados de perfis

A base de dados de perfis é responsável por armazenar os perfis que descrevem o comportamento do sistema. Esses perfis podem ser traçados através de análises quantitativas, medidas estatísticas, redes neurais, algoritmos genéticos ou abordagens do sistema imunológico. Além disso, os perfis são periódica e automaticamente atualizados para prover uma detecção adaptativa[11].

Detector de anomalia

O detector de anomalia recebe o fluxo de eventos do sistema de filtragem e procura por eventos que caracterizem um comportamento anômalo. Para fazer isso, a informação recebida é comparada a um conjunto de perfis previamente estabelecidos e armazenados na base de dados de perfis. Se qualquer sinal de comportamento anormal é detectado, o detector de anomalia ativa o executor da resposta primária e alimenta o gerador de assinaturas com a informação detectada como anormal.

[1]É assumido que o sistema de detecção de anomalia pode utilizar estratégias de monitoramento diferentes em contraste com a única adotada pelo sistema de detecção de mau uso.

Executor da resposta primária

Uma vez ativado, o executor da resposta primária inicia uma série de medidas de contenção para diminuir e até mesmo bloquear a ação de um provável ataque. A reação iniciada pelo executor da resposta primária é limitada e geral, uma vez que o ataque não pode ser identificado especificamente. A principal proposta dessas medidas é minimizar os danos até que uma resposta específica e eficiente possa ser executada ou até que haja a intervenção do administrador.

5.1.4 Gerador de assinaturas

Uma característica inovadora do IDS proposto é a conversão da informação considerada como anômala em uma assinatura que identifica especificamente o ataque relacionado à anomalia encontrada. Essa conversão introduz uma capacidade de aprendizado, intrínseca da detecção de anomalia, para o sistema de detecção de mau uso, provendo um processo de detecção e resposta mais eficiente e preciso para futuras ocorrências do ataque. Desse modo, o IDS é capaz de gerar automaticamente assinaturas de ataques que são desconhecidos no sistema. O gerador de assinaturas é responsável pela conversão da informação anômala em assinaturas de ataques. Depois da geração da assinatura, o gerador de assinaturas ativa o gerador de respostas. O componente gerador de assinaturas é abordado em [30] que emprega a forense computacional para desempenhar sua funcionalidade.

5.1.5 Gerador de respostas

O gerador de respostas recebe a assinatura e elabora um conjunto de contramedidas específicas para o ataque. Tanto a assinatura como a resposta produzida são armazenadas na base de dados de assinaturas. Entretanto, a tarefa de desenvolver um conjunto de ações de contenção com base na assinatura recém criada, que atue especificamente no ataque, não é trivial. Primeiro, a assinatura desenvolvida pelo gerador de assinaturas deve prover informações suficientes para identificar genericamente o ataque e as áreas afetadas do sistema. Segundo, mesmo que isso seja suprido, o gerador de respostas deve adotar técnicas sofisticadas que permitam extrair, a partir da assinatura, os pontos afetados no sistema e, com base nisso, elaborar as ações que atuarão especificamente para as instâncias futuras do ataque. O gerador de respostas ainda não foi desenvolvido pelas pesquisas que abordam esse sistema de segurança.

5.1.6 Sistema de detecção de mau uso

A detecção de mau uso compreende a procura por padrões de atividades que caracterizem um ataque conhecido ou uma violação da política de segurança. Essa abordagem mostra-se eficiente e confiável e, como resultado, é a abordagem mais utilizada em IDSs comerciais[10]. Os componentes do sistema de detecção de mau uso são descritos a seguir.

Base de dados de assinaturas

A base de dados de assinaturas é responsável por armazenar as assinaturas dos ataques, relacionando-as às medidas de respostas específicas. As assinaturas são utilizadas pelo detector de mau uso, enquanto as contramedidas são consultadas pelo executor da resposta secundária. Desse modo, o IDS proposto pode detectar e responder especificamente a cada manifestação de ataques conhecidos no sistema. A introdução de novas assinaturas e contramedidas na base de dados de assinaturas podem ser conduzidas de duas formas:

1. automaticamente pelo gerador de respostas;

2. ou manualmente pelo administrador do sistema através do console.

Detector de mau uso

O detector de mau uso recebe o fluxo de dados de eventos do sistema de filtragem e compara com os padrões armazenados na base de dados de assinaturas. Se qualquer padrão é encontrado no fluxo, o detector de mau uso ativa o executor da resposta secundária. A detecção é conduzida em tempo real e utiliza a abordagem baseada em transição de estados[11, 49].

Executor da resposta secundária

Uma vez ativado, o executor da resposta secundária recebe o padrão identificado e busca na base de dados de assinaturas pelas contramedidas específicas relacionadas ao padrão. Quando recebe a resposta, o executor da resposta secundária executa as contramedidas.

5.1.7 Console

A interface entre o IDS proposto e o administrador do sistema é possibilitado pelo console. Essa interface permite a inclusão e remoção das assinaturas e contramedidas na base de dados de assinaturas e o ajuste do detector de anomalia.

5.2 Analogias entre o sistema imunológico humano e o modelo do IDS

Como descrito na Seção 3.1, o sistema imunológico humano é dividido em sistema inato e adaptativo. Uma analogia entre esses sistemas e o IDS proposto é ilustrado na Figura 5.2. O sistema inato é parcialmente representado pelo sistema de filtragem cuja função assemelha-se ao processo de apresentação do antígeno. Outras características principais do sistema inato, como a detecção e resposta não específica, estão presentes no sistema de detecção de anomalia (respectivamente no detector de anomalia e no executor da resposta primária), que é apropriadamente modelada dentro do sistema adaptativo considerando as características adaptativas da detecção de anomalia.

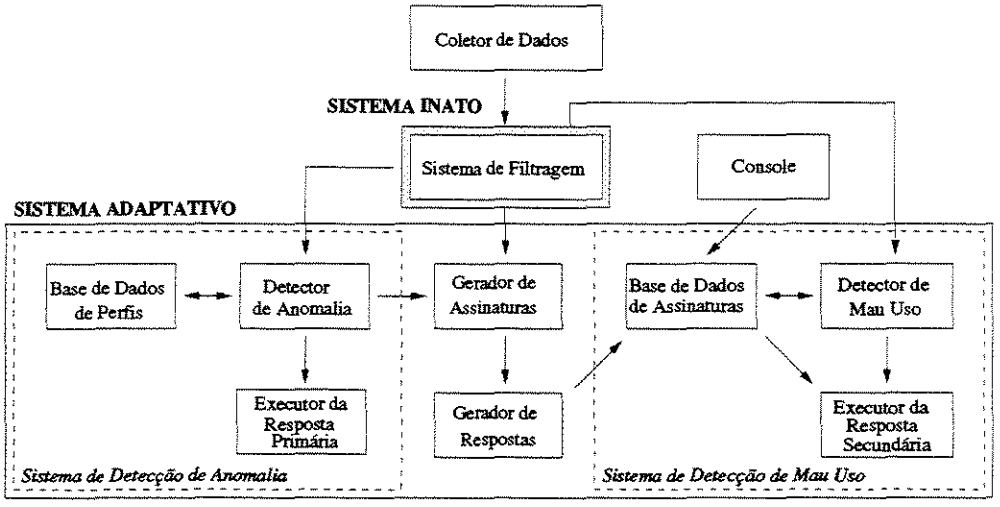

Figura 5.2: Analogias entre os sistemas inato e adaptativo com o IDS proposto.

Por outro lado, o sistema adaptativo é representado pelos componentes que implementam o aprendizado e a memória no IDS proposto. Além disso, alguns desses componentes tem outras características importantes do sistema adaptativo, tais como: detecção apurada e resposta específica.

Outras analogias são apresentadas na Tabela 5.1 que relaciona cada componente do IDS com as características do sistema imunológico humano.

Componentes do IDS	Sistema Imunológico
Coletor de dados	Fonte de proteínas *self* e *nonself*
Sistema de filtragem	Processo de apresentação do antígeno
Base de dados de perfis	Conjunto de receptores gerados aleatóriamente
Detector de anomalia	Detecção não específica do fagócito
Executor da resposta primária	Resposta primária do sistema inato
Gerador de assinaturas	Produção de células de memória
Gerador de respostas	Produção de anticorpos específicos
Base de dados de assinaturas	Conjunto de células de memória específicas
Detector de mau uso	Detecção realizada pelas células de memória
Executor da resposta secundária	Resposta específica do sistema adaptativo
Console	Imunidade adquirida por meio de vacinas

Tabela 5.1: Analogias entre os componentes do IDS proposto e o sistema imunológico.

5.3 Conclusão

Este capítulo apresentou o modelo de um sistema de segurança, baseado no sistema imunológico humano, que é capaz de detectar e responder a ataques, mesmo que previamente não conhecidos. Para tal, o modelo adota uma abordagem diferente dos trabalhos que relacionam o sistema imunológico à segurança de computadores e incorpora, além da detecção de anomalia, a detecção de mau uso. Além disso, também apresenta características de aprendizado, sendo capaz de inserir automáticamente assinaturas e respostas na base de dados de mau uso, a partir das atividades consideradas como anômalas.

Capítulo 6

Estrutura do Mecanismo de Resposta

O sistema imunológico humano, além de detectar agentes patogênicos, também é capaz de disparar uma resposta contra tais agentes. Essa resposta tem como objetivo eliminar intrusos e células contaminadas, e também restabelecer o funcionamento normal do corpo. O sistema de segurança apresentado, assim como o sistema imunológico humano, também incorpora um mecanismo de resposta contra invasões. Esse mecanismo é projetado como um sistema de resposta automático e, baseado no sistema imunológico, tem como principal funcionalidade conter o poder do ataque, além de permitir uma extensão que efetue a restauração das informações efetadas pelos danos provocados no sistema.

As respostas disparadas pelo sistema de segurança envolvem três componentes: executor da resposta primária, executor da resposta secundária e gerador de respostas. O executor da resposta primária e o executor da resposta secundária são responsáveis por iniciar um processo de resposta quando alguma detecção ocorre. Já o gerador de respostas é responsável por elaborar uma resposta com base na assinatura do ataque.

O presente trabalho restringe-se aos dois esquemas de resposta disponíveis no sistema de segurança, providos pelos componentes: executor da resposta primária e executor da resposta secundária. Os esquemas, denominados resposta primária e secundária, possuem suas particularidades e são invocados respectivamente pelos sistemas de detecção de anomalia e de mau uso. Os componentes desenvolvidos e abordados diretamente no trabalho compõem o mecanismo de resposta do sistema de segurança, e são diferenciados dos outros componentes na Figura 6.1.

Esse capítulo descreve o modelo do mecanismo de resposta do sistema imunológico, e sua estrutura é descrita a seguir. A Seção 6.1 aborda resumidamente os principais conceitos do mecanismo de resposta. A Seção 6.2 descreve a organização estrutural do mecanismo, descrevendo seus componentes. As Seções 6.3 e 6.4 detalham respectivamente

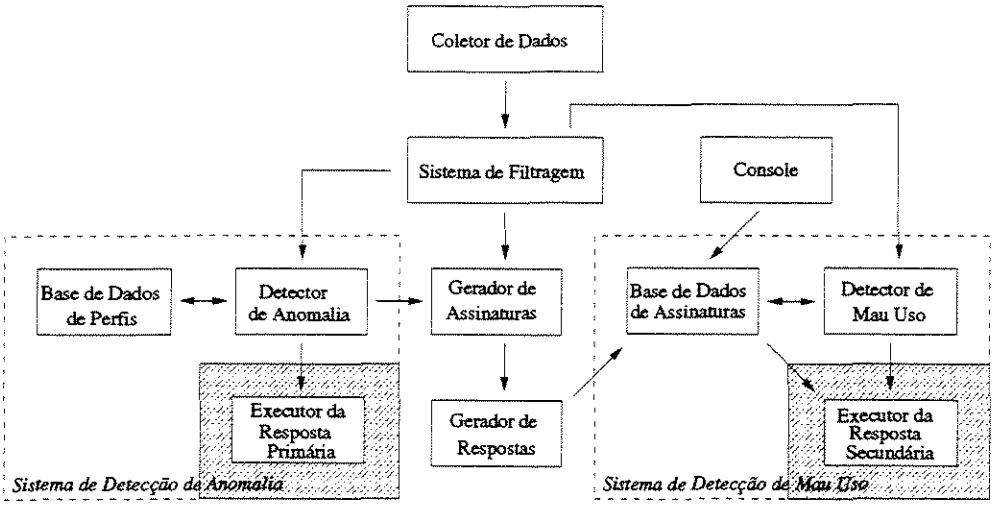

Figura 6.1: Componentes abordados diretamente no trabalho.

as respostas primária e secundária, que são os dois tipos de respostas implementadas pelo mecanismo. A Seção 6.5 descreve o esquema de política de resposta. A Seção 6.6 descreve o nível de confiança, um fator adicionado ao mecanismo para torná-lo mais adaptável. A Seção 6.7 descreve algumas analogias dos componentes do mecanismo de resposta com componentes do sistema imunológico humano. Por fim, a Seção 6.8 aborda de modo superficial questões relativas à segurança do mecanismo de resposta.

6.1 Conceitos básicos

A seguir estão descritas resumidamente as principais características do mecanismo de resposta.

6.1.1 Função do mecanismo de resposta

Como mencionado, o mecanismo de resposta tem como principal objetivo impedir rapidamente o progresso do ataque. Para contê-lo, são adotados um conjunto de ações, que atuam em dois níveis diferentes e limitam o uso dos recursos do sistema, impedindo o progresso das atividades ilícitas. Ações do **nível de processos** limitam a capacidade de processamento, bem como a quantidade e o acesso aos recursos utilizados pelos processos suspeitos, enquanto as ações do **nível de rede** interferem no tráfego de rede produzido pelas máquinas alvo das ações de resposta. Supõe-se que, com as ações disponibilizadas por esses dois níveis diferentes de atuação, seja possível limitar satisfatoriamente grande

parte das intrusões.

O mecanismo de resposta, além de prover um modo de lançar ações de contenção sobre seus alvos, também deve permitir sua revogação. Esse tipo de função é importante quando os alvos de uma resposta não apresentam mais risco ao sistema e, portanto, não necessitam de restrições. Entretanto, a revogação de uma resposta deve ser efetuada com cautela, preferivelmente pelo administrador, para assegurar que atividades futuras não causem danos ao sistema.

6.1.2 Tipos de resposta

Baseado no sistema imunológico humano, o mecanismo dispõe de duas respostas de contenção diferentes, denominadas respostas primária e secundária. A **resposta primária**, executada pelo componente executor da resposta primária do sistema de segurança, é acionada pela parte da detecção de anomalia. Essa resposta não é específica e as ações lançadas variam de acordo com uma avaliação do possível perigo do ataque, denominado **nível de risco**, que é fornecido pelo sistema de detecção. Esse tipo de resposta é inspirado no primeiro encontro do sistema imunológico humano com um agente patogênico específico, onde a resposta produzida , apesar de não ser específica, é capaz de exterminar o agente patogênico.

Já a **resposta secundária**, iniciada pelo executor da resposta secundária, é acionada pela parte da detecção de mau uso e conta com ações específicas. A justificativa para a especificidade, é que um processo de resposta secundária é invocado quando padrões de um ataque conhecido, sumarizado pela assinatura do ataque, ocorrem no sistema. Assim, como já são conhecidas as características da invasão, é possível preestabelecer um conjunto de ações de contenção e associá-lo a assinatura. A resposta secundária é baseada nas reações produzidas pelas células de memória do sistema imunológico humano, que bloqueiam e eliminam um ataque com maior eficiência quando comparado a uma resposta primária.

6.1.3 Características

Independente do tipo de resposta, as reações executadas pelo mecanismo apresentam-se como reações adaptáveis e distribuídas. A **adaptabilidade** permite que a resposta seja ajustada de acordo com as atividades do ataque, dimensionando o poder de suas ações a partir de estímulos recebidos. Os estímulos variam de acordo com a ação empregada e possibilitam, através do ajuste dinâmico da intensidade da ação, que respostas disparadas a falsos positivos ou respostas dotadas de estratégias inadequadas possam ter seu impacto reduzido sobre usuários e processos legítimos.

A adaptabilidade nas respostas é alcançada com o auxílio de algumas estratégias estabelecidas. A primeira delas é o cálculo de um valor chamado **nível de confiança**. O nível de confiança é determinado antes do mecanismo de resposta interferir no funcionamento do sistema, com base em algumas métricas previamente observadas. Esse valor procura avaliar o quanto uma ação de contenção parece ser aplicável em determinado momento. A segunda estratégia é o esquema adaptativo apresentado em algumas ações de contenção da resposta primária, que alteram sua intensidade dinamicamente. A terceira estratégia de adaptabilidade basea-se no fato da resposta ser distribuída. Nessa estratégia, uma ação de contenção executada por uma máquina, que envolve o bloqueio do tráfego de rede proveniente de outras máquinas internas da rede, pode ser revogada quando usuários não suspeitos são afetados.

Como citado, além de ser adaptável, a resposta também é **distribuída**, envolvendo uma reação entre vários nós da rede. Quando um ataque é detectado, além de ser iniciada uma reação local à máquina onde ocorre o alarme, chamada **reação direta**, também é disparado um processo de alerta que comunica o início da resposta a várias outras máquinas. O alerta capacita esse conjunto de máquinas a escolher suas ações com relação ao ataque. A reação conduzida em tais máquinas é chamada **reação estimulada**. Além disso, cada máquina envolvida nesse esquema de resposta é denominado **ponto de resposta**.

6.1.4 Política de resposta

Para cada ponto de resposta, o mecanismo requisita o estabelecimento de uma **política de resposta**. Essa política de resposta define as estratégias de execução das ações, disponibilizadas pelo mecanismo, que são incorporadas em uma reação. Para distinguir cada ponto de resposta, é considerado que cada máquina possui sua função específica e importância na rede, e por isso requisitam estratégias diferentes de respostas.

6.2 Organização estrutural

O mecanismo de resposta é organizado de modo a proporcionar uma resposta distribuída, formada por um conjunto de respostas localizadas. Para conseguir isso, conta com dois componentes básicos: o componente de controle/comunicação e o componente local de reação. O **componente de controle/comunicação** é composto por vários agentes móveis com funções específicas que interagem entre si de modo a prover uma resposta global. Os agentes se locomovem pela rede invocando, quando necessário, os **componentes locais de reação** que dão suporte para a execução das ações de contenção. Cada ponto de resposta da rede conta com um componente local de reação. Assim, em determinado momento, um ponto de resposta contém o seu componente local de reação e parte do

conjunto de agentes responsável pela coordenação e controle das resposta lançadas pelo mecanismo. A Figura 6.2 mostra um exemplo de uma rede com seus respectivos pontos de resposta.

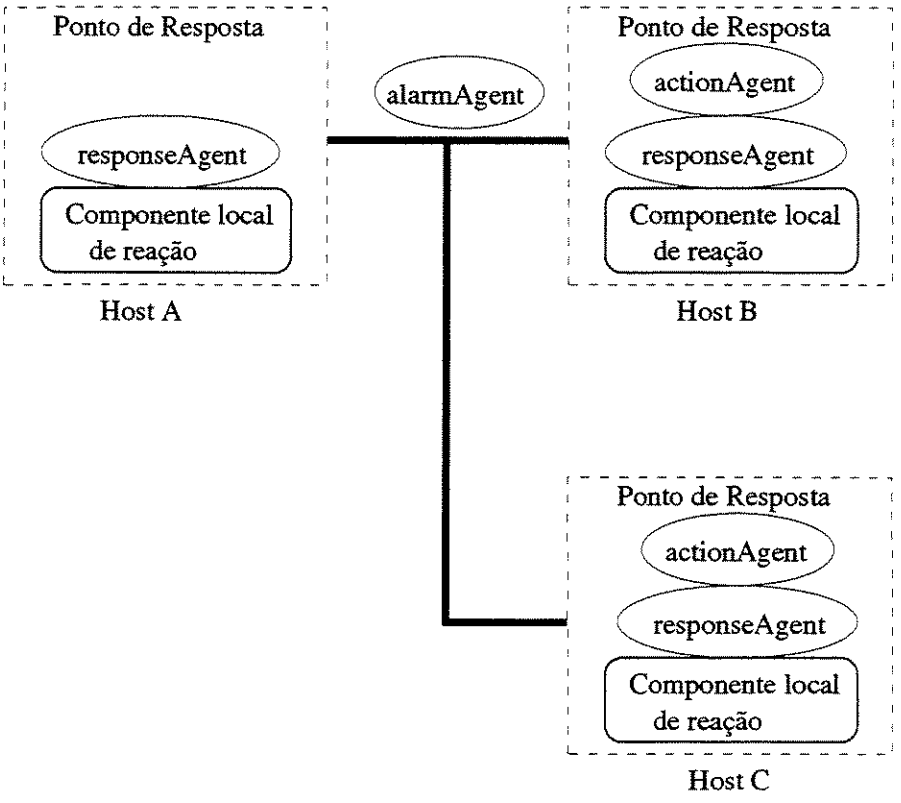

Figura 6.2: Exemplo de um conjunto de pontos de resposta.

A seguir, os dois componentes fundamentais são detalhados.

6.2.1 Componente de controle/comunicação

Quando um ataque é detectado pelo sistema de segurança imunológico, o mecanismo de resposta, além de reagir localmente, é capaz de alarmar e até mesmo disparar uma reação em pontos remotos. Para prover esse tipo de resposta distribuída, o mecanismo conta com o componente de controle/comunicação. O componente de controle/comunicação é a parte do mecanismo de resposta responsável pela coordenação das reações e comunicação entre os pontos de resposta. Para isso, é composto por vários agentes móveis, que atuam cooperativamente para desempenhar a função atribuída ao componente. A escolha de agentes móveis para a composição do componente de controle/comunicação deve-se prin-

cipalmente a algumas propriedades encontradas na tecnologia, descritas na Seção 4.1.1, que auxiliam e tornam mais robusto o projeto de um sistema distribuído. Tais propriedades, como mobilidade, autonomia e adaptabilidade, permitem redefinir o componente de controle/comunicação como um conjunto de aplicações autônomas e cooperativas capazes de reagir a estímulos recebidos do sistema de detecção, invocando ações e trafegando pela rede para conter um ataque.

Assim como ocorre com as células do sistema imunológico humano, no mecanismo de resposta, os agentes desempenham uma função específica. Além disso, cada agente é projetado para ter um alto nível de independência de forma que, se algum é comprometido, o mecanismo de resposta ainda é capaz de continuar suas tarefas. Apesar dessa independência, os agentes possuem uma grande interação para coordenar e executar as reações de uma resposta. Por meio dessa interação, é possível elaborar um esquema de comunicação e coordenação que ampare a execução das ações de uma resposta.

6.2.1.1 Agentes

Para ser possível elaborar e executar as reações diretas e estimuladas, diversos agentes móveis, com funcionalidades distintas, distribuídos pela rede interagem entre si e coordenam as ações de contenção. O **initAgent** é o agente responsável pela inicialização do mecanismo de resposta. O **primaryresponseAgent** e o **secondaryresponseAgent** são os agentes que gerenciam um ponto de resposta e são acessados quando deseja-se iniciar uma reação. O **alarmAgent** é o agente que realiza a comunicação entre pontos de resposta. O **logAgent** é o agente responsável por registrar os eventos ocorridos no mecanismo de resposta em arquivos. Por fim, os agentes encarregados de executar e coordenar as ações de contenção, chamados **reactionAgents**, são divididos em **directReactionAgent** e **stimulatedReactionAgent** respectivamente para as reações diretas e estimuladas. A Figura 6.3 ilustra simplificadamente a interação de cada agente do mecanismo de resposta. Os agentes são descritos mais detalhadamente nas seções seguintes.

InitAgent

O initAgent é o agente responsável por iniciar o mecanismo de resposta e é o único agente que necessita ser criado pelo administrador do sistema de segurança. Através dele são criados todos os outros agentes que integram o componente. A primeira tarefa do initAgent é a de estabelecer uma configuração para o mecanismo de resposta. A configuração é definida através da interpretação da política de resposta já preestabelecida[1]. Após o estabelecimento da configuração, o initAgent passa à criação dos agentes que atuarão nos pontos de resposta. Para isso, um primaryResponseAgent e um secondaryResponseA-

[1]A política de resposta é posteriormente discutida em mais detalhes na Seção 6.5.

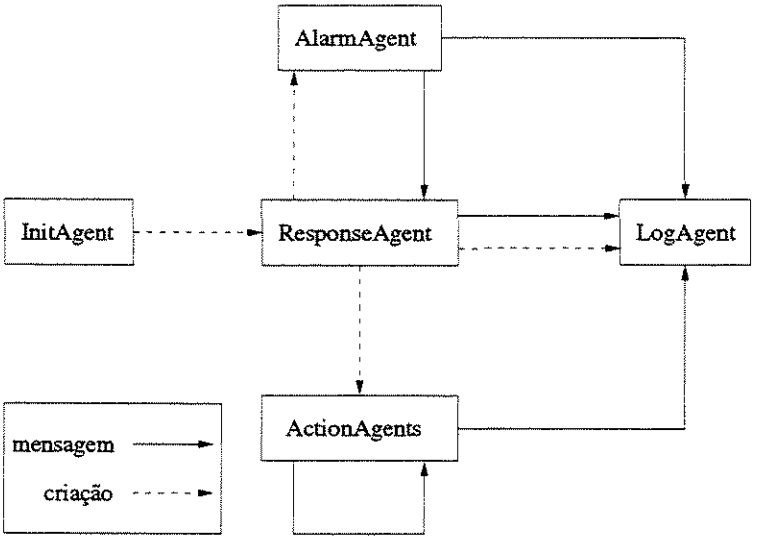

Figura 6.3: Interação entre os agentes do sistema de resposta

gent são criados e designados para cada ponto de resposta. Após a criação dos agentes fundamentais para o funcionamento do mecanismo, o initAgent finaliza sua execução.

ResponseAgent

O responseAgent é o agente que coordena um ponto de resposta da rede. Existem dois tipos de responseAgent, denominados primaryResponseAgent e secondaryResponseAgent, responsáveis por coordenar, respectivamente, as reações das respostas primária e secundária em um ponto de resposta. Independente do tipo de responseAgent, esse agente possui um único esquema de criação e execução. O ciclo de execução de um responseAgent é descrito a seguir.

Antes de atuar ativamente, cada responseAgent passa por uma fase de preparação, onde o agente é criado e preparado para a coordenação. Essa fase inicia-se com sua criação que é feita pelo initAgent de acordo com a política de resposta e com seu futuro ponto de resposta. Assim que é criado, o responseAgent transporta-se até o ponto de resposta pelo qual é responsável, e após seu estabelecimento, inicia-se a fase de coordenação.

Nesse ponto, o agente torna-se o responsável por iniciar todas as reações do ponto de resposta. As reações têm início quando alguma mensagem específica de alerta lhe é enviada. Assim que o agente a recebe, um actionAgent é criado para executar e controlar as ações da reação. Como o mecanismo de resposta conta com as reações direta e estimulada, um responseAgent pode receber dois tipos diferentes de mensagens para disparar uma reação. As mensagens e o procedimento adotado por um responseAgent ao recebê-las

são descritos a seguir.

- Mensagem do tipo **respond**. Esse tipo de mensagem indica o início de um processo de resposta e de uma reação direta. Quando alguma intrusão é detectada em uma máquina, o sistema de detecção envia esse tipo de mensagem ao responseAgent responsável. Assim que a recebe, o responseAgent cria um directActionAgent que executa a reação direta. Além disso, quando uma mensagem desse tipo é recebida, os outros pontos de resposta que compõem o mecanismo são alertados. Para isso, o responseAgent cria um conjunto de alarmAgents que é encarregado de propagar o alerta.

- Mensagem do tipo **alarm**. Essa mensagem é enviada por um alarmAgent proveniente de um ponto de resposta remoto. Seu recebimento, por um responseAgent, indica que uma resposta de contenção foi iniciada no ponto de criação do alarmAgent. Quando uma mensagem desse tipo é recebida, o responseAgent cria um stimulatedActionAgent responsável por executar uma reação estimulada para o processo de resposta.

O conteúdo e o formato das mensagens *respond* e *alarm* dependem do tipo de resposta iniciado. Essa diferença é justificada pelo fato das respostas primária e secundária serem fundamentadas em esquemas de detecção diferentes e, por isso, manupularem conjuntos distintos de informações. O conteúdo das mensagens para as respostas primárias e secundárias são comentadas respectivamente nas Seções 6.3 e 6.4.

AlarmAgent

O alarmAgent é o agente responsável pela comunicação entre os pontos de resposta. Sempre é criado por um responseAgent quando um processo de resposta é iniciado. No momento de sua criação, é designado a outros pontos para notificar a ocorrência de uma possível intrusão. Assim que chega a um ponto de resposta, um alarmAgent informa, através de uma mensagem do tipo *alarm*, ao responseAgent responsável que um processo de resposta foi iniciado no ponto de resposta de onde proveio. Claramente, os responseAgents notificados dependem do tipo de resposta que foi iniciado. Por exemplo, se uma resposta primária teve início, somente os primaryResponseAgents do mecanismo serão notificados.

ReactionAgent

Os reactionAgents são os agentes responsáveis pela execução e controle das ações de contenção em um processo de resposta. Para lançar uma ação, esses agentes interagem

com o componente local de reação, invocando as ações disponibilizadas. Mesmo depois da ação ser lançada, a interação prossegue com o objetivo de controlar sua intensidade ou mesmo removê-la.

Existem dois tipos diferentes de reactionAgents, criados por um responseAgent de acordo com o tipo de reação desejada. O directReactionAgent é criado quando um responseAgent é notificado pelo sistema de detecção e uma reação direta é iniciada. Já o stimulatedReactionAgent é criado para realizar uma reação estimulada quando um responseAgent é notificado por um alarmAgent a respeito do início de um processo de resposta remoto. Os dois tipos de reactionAgent são detalhados como segue.

- **DirectReactionAgent**: é o agente responsável por executar as reações diretas e, por isso, sempre é criado no ponto de resposta onde o ataque foi detectado. Antes de iniciar a execução das ações, o directReactionAgent obtém o nível de confiança e, baseado nesse valor, decide se as ações serão executadas e de que forma. Após essa etapa, se as ações forem lançadas, o agente continua sua execução para eventualmente alterar a intensidade das ações ou para removê-las, quando houver estímulos para tal. Sua execução só termina quando todas as ações de contenção lançadas terminarem.

- **StimulatedReactionAgent**: é o agente responsável por executar as reações estimuladas. Como o próprio nome sugere, uma reação estimulada executa um conjunto de ações como forma de reação a um ataque detectado em outra máquina. A reação efetuada por esse agente, pode tanto lançar ações de contenção como também solicitar a remoção de ações da reação direta quando for preciso. Por isso, uma reação estimulada permanece ativa enquanto a reação direta do mesmo processo de resposta existir. Esse tipo de controle auxilia a negociação entre pontos de resposta para a revogação de ações e é alcançado através da interação entre os stimulatedReactionAgents e o directReactionAgent de um mesmo processo de resposta.

Além disso, é através do reactionAgent que a capacidade de adaptação das ações de contenção é alcançada, tornando o mecanismo de resposta mais tolerável a respostas disparadas em detecções de falsos positivos ou a estratégias inadequadas de resposta.

LogAgent

O mecanismo de resposta possui um esquema de registros dos eventos efetuados por todos os agentes. Para isso, cada ponto de resposta possui um logAgent, que registra os eventos produzidos por outros agentes no ponto. Um logAgent é criado pelo responseAgent, assim que este chega a seu ponto de resposta.

Sempre que um agente realiza algum evento, ele envia uma mensagem ao logAgent do mesmo ponto de resposta para informá-lo. O logAgent, ao recebê-la, faz os processamentos devidos e registra os dados referentes ao evento em um arquivo. Esse arquivo armazena informações sobre todos os eventos que já ocorreram no ponto de resposta.

O fato de cada ponto de resposta possuir seu agente de registro de eventos, ao invés de um único logAgent registrar todos os eventos do mecanismo, é justificado por eventuais isolamentos de máquinas na rede que possam ocasionar a perda de alguns registros de eventos. Entretanto, além do mecanismo montar um registro de eventos para cada ponto de resposta, cada logAgent também é capaz de montar um registro contendo os eventos de todos os pontos. Para fazer isso, um logAgent cria um clone seu que varre os pontos de resposta. Para cada ponto de resposta visitado, o agente clonado busca o registro de eventos local e envia uma cópia para sua origem. Ao varrer todos os pontos de resposta, o logAgent clonado finaliza sua execução. Nesse momento, o logAgent possui todos os registros de eventos do mecanismo de resposta e, com isso, monta um único registro ordenando os eventos pela data de ocorrência. Um ponto importante a ser observado nesse processo, é que para realizar uma ordenação consistente de todos os registros de eventos, é necessário que as máquinas da rede tenham sincronia em seus relógios.

6.2.1.2 Etapas de funcionamento

O componente de controle/comunicação possui duas etapas de funcionamento. A primeira etapa é de inicialização, onde o mecanismo prepara todos os pontos de resposta com os agentes necessários. A segunda etapa é de ativação, onde os agentes do mecanismo de resposta já estão prontos para reagir e aguardam algum estímulo para iniciar uma resposta.

Etapa de inicialização

O esquema de inicialização encarrega-se de preparar o mecanismo de resposta para o esquema de ativação. Nele, é estabelecida a estrutura para a comunicação e controle entre os pontos de resposta. O esquema é iniciado com a criação de um agente de inicialização, o initAgent. O agente de inicialização, por sua vez, cria os agentes de gerência das respostas primária e secundária, primaryResponseAgent e secondaryResponseAgent, para cada ponto de resposta. Assim que é criado, cada agente gerenciador desloca-se até o ponto de resposta de que é incumbido e em seguida cria um agente para registrar os eventos do ponto, o logAgent. Essa fase encerra-se quando todos os agentes estão em seus respectivos pontos de resposta prontos para iniciar qualquer tipo de reação. A Figura 6.4 ilustra um exemplo de criação e transporte dos agentes de gerência.

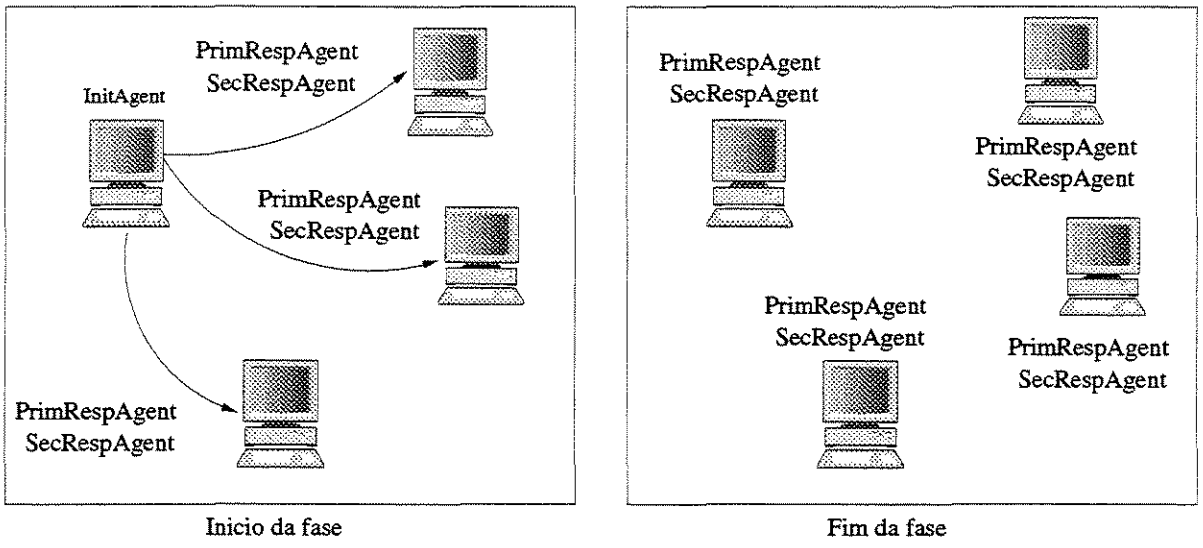

Figura 6.4: Fase de inicialização do mecanismo de resposta

Etapa de ativação

No esquema de ativação, todo o mecanismo de resposta já está iniciado e cada ponto de resposta está pronto para reagir caso receba algum estímulo. Esse esquema tem início quando, em qualquer ponto de resposta algum primaryResponseAgent recebe uma mensagem do sistema de detecção de anomalia ou algum secondaryResponseAgent recebe uma mensagem do sistema de detecção de mau uso, acusando a ocorrência de um ataque.

Assim que a mensagem é recebida, o agente gerenciador cria um agente responsável pela reação direta tomada nesse ponto de resposta, o directReactionAgent. Na criação desse agente, é passada a mensagem de ativação da resposta recebida pelo agente gerenciador. Com base nessa mensagem, a reação direta é elaborada e executada.

Além disso, o agente gerenciador ainda cria um conjunto de alarmAgents que fica responsável por alertar os outros pontos de resposta a respeito da detecção do ataque e do início de um processo de resposta. Quando é criado, um agente de alarme desloca-se aos pontos de resposta da rede. O agente, ao chegar em um ponto, envia uma mensagem ao agente gerenciador, relatando o início de uma reação em um máquina remota. Este, ao recebê-la, dá início a uma reação estimulada, criando um stimulatedReactionAgent para executar essa reação. O agente de reação estimulada finaliza sua execução e, conseqüentemente, a execução de suas ações após o término da reação direta do mesmo processo de resposta. A Figura 6.5 ilustra a atividade dos agentes envolvidos no esquema de ativação.

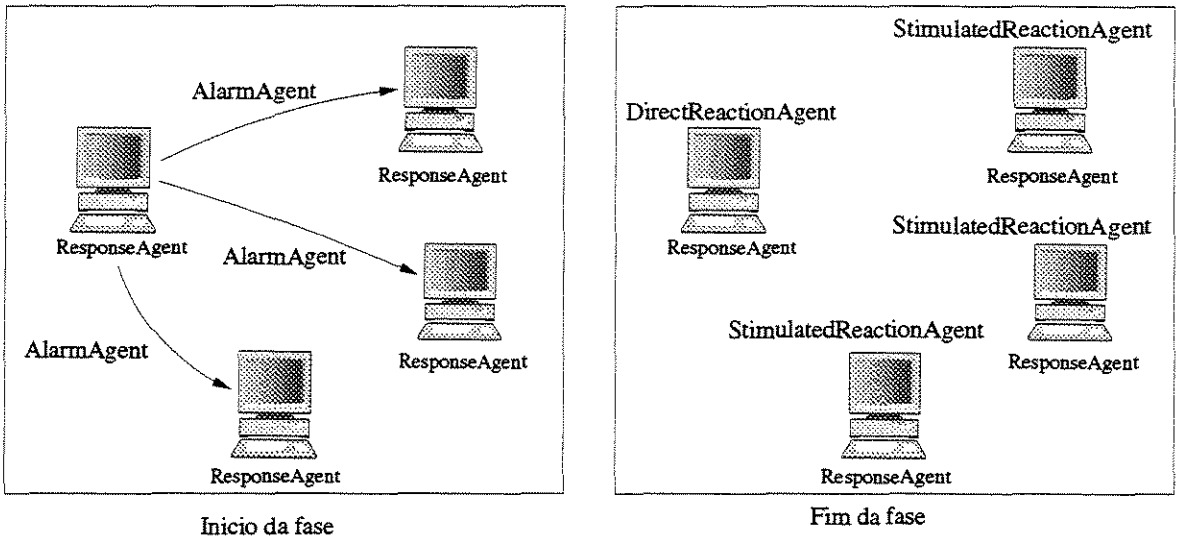

Figura 6.5: Fase de ativação do sistema de resposta

6.2.2 Componente local de reação

O componente local de reação é a parte do mecanismo de resposta que provê o suporte para a execução das ações de contenção, implementando todas as ações possíveis de serem empregadas em uma resposta. Independente do tipo de resposta, primária ou secundária, ou de reação, direta ou estimulada, o componente local de reação sempre é invocado por um reactionAgent quando alguma ação de contenção é necessária.

Grande parte das pesquisas direcionadas a automatizar o processo de resposta a intrusão provêem ações que atuam restringindo a execução de processos[20, 94] ou o tráfego de rede[98, 86]. Baseado nisso, o conjunto de ações de contenção, disponibilizado pelo mecanismo de resposta, atua em dois níveis do sistema: nível de processos e nível de rede. Espera-se que uma reação composta por ações desses dois níveis seja capaz de limitar e até mesmo extinguir o poder de uma invasão.

Para que um reactionAgent possa utilizar as ações, o componente local de reação disponibiliza diversas primitivas que são invocadas com os parâmetros apropriados pelo agente. As ações e suas respectivas primitivas para cada um dos níveis são descritas como segue.

6.2.2.1 Nível de processos

Esse nível de ação tem como objetivo controlar o acesso aos recursos utilizados por um processo alvo de uma resposta, diminuindo a chance de que suas atividades violem

a segurança do sistema. Para isso, as ações do nível de processo estão divididas em três classes elaboradas com base nas condições que garantem a segurança de um sistema[90]: confidencialidade, integridade e disponibilidade. A primeira classe do nível de processos procura manter a confidencialidade e a integridade das informações do sistema, controlando o acesso ao sistema de arquivos para evitar leituras e modificações ilícitas. A segunda classe é a da disponibilidade, que fornece ações para garantir que os recursos do sistema permaneçam disponíveis para outros processos em execução. Além das duas classes mencionadas, ainda existe uma terceira classe, as ações de emergência, que interrompem a execução de um processo e geralmente são utilizadas quando o poder de contenção das outras ações não é suficiente.

Estados de um processo

Um processo, no escopo do mecanismo de resposta pode assumir três estados diferentes: normal, suspeito e intruso. Todo processo é inicialmente qualificado como normal pelo mecanismo de resposta e só tem seu estado alterado quando algum dos responseAgent no ponto de resposta recebe uma mensagem do sistema de detecção, informando o envolvimento do processo em um alarme. Nesse caso, o processo passa de normal para o estado de intruso e um reactionAgent aplica um conjunto de ações para conter suas atividades. Se as ações de contenção de um processo classificado como intruso são retiradas, ele passa para o estado de suspeito onde fica sob análise do mecanismo de resposta. Um processo suspeito pode voltar ao estado de intruso caso volte a apresentar atividades de risco para o sistema. O processo ainda pode voltar para o estado normal quando supõe-se que não existe mais risco para o sistema. A Figura 6.2.2 ilustra um diagrama com os possíveis estados de um processo e suas transições.

Como mencionado, quando um processo está no estado intruso, o mecanismo de resposta executa um conjunto de ações para limitar e conter suas atividades. As ações de contenção desse nível estão descritas a seguir.

6.2.2.1.1 Disponibilidade

Essas ações têm como objetivo evitar que os recursos do sistema sejam esgotados de forma maliciosa por um processo.

Consumo de memória

Um modo de conter um processo é limitando sua quantidade de memória alocada. A imposição de um limite de alocação, além de distribuir melhor o recurso entre os processos em execução, pode restringir a operação do processo, diminuindo o poder da invasão. A ação é estabelecida pela seguinte primitiva:

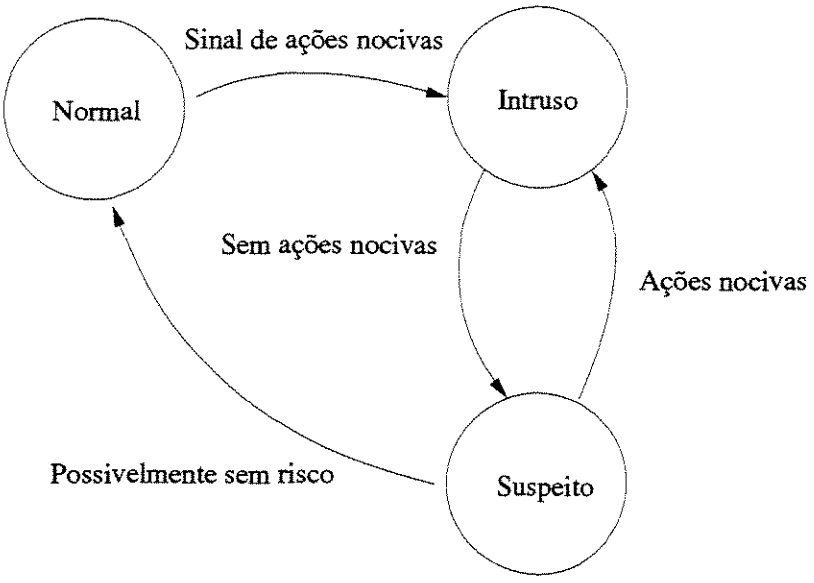

Figura 6.6: Estados de um processo.

```
memlimit(processID, memoryMAX)
```

A primitiva `memlimit` limita a quantidade de memória alocada ao processo `processID` em `memoryMAX`. Desse modo, só é permitido ao processo alocar mais memória se a quantidade já alocada for inferior ao limite estabelecido. Se o limite é excedido, então a execução do processo é parada, podendo prosseguir posteriormente se a primitiva for invocada com um valor de `memoryMAX` negativo.

Consumo do processador

Outra forma de limitar a atividade de um processo é fixando um limiar máximo de uso do processador. De modo semelhante ao consumo de memória, essa ação fixa uma porcentagem máxima de uso do processador. A primitiva que representa essa ação é a seguinte:

```
cpulimit(processID, percent)
```

A primitiva `cpulimit` limita a porcentagem de uso do processador para o processo `processID` em `percent`, onde `percent` é um valor entre 0 e 100. Para retirar essa ação é necessário invocar a primitiva com um valor de **percent** negativo.

Número máximo de conexões

Limitar o número máximo de conexões de rede simultâneas para um processo pode ser um modo interessante de garantir que os recursos de rede estejam sempre disponíveis. O limite de conexões para um processo é fixado através da seguinte primitiva:

```
conectionlimit(processID, max)
```

A primitiva `conectionlimit` limita a quantidade de conexões de rede simultâneas, estabelecidas pelo processo `processID`, em `max` conexões. Para retirar a ação é necessário invocar a primitiva com um valor de `max` negativo.

Quantidade de processos filhos

Essa ação é semelhante a descrita anteriormente, mas limita a quantidade de processos filhos através da seguinte primitiva:

```
childplimit(processID, max)
```

Acesso à disco

É importante, em alguns casos, que a quantidade de informações buscadas no disco rígido por um processo seja limitada. Esse limite, além de retardar suas atividades, permite um melhor compartilhamento do recurso com os outros processos em execução. A ação que desempenha essa limitação é representada pela primitiva a seguir:

```
disklimit(processID, max)
```

A primitiva `disklimit` limita o acesso no disco rígido em `max` *bytes* por segundo para o processo `processID`. Para retirar a ação é necessário invocar a primitiva com um valor de `max` negativo.

6.2.2.1.2 Confidencialidade e integridade

Essas ações tem como objetivo evitar que a informação de um sistema seja acessada ou alterada de forma não autorizada. Para isso, o mecanismo de resposta reage através do estabelecimento de limites de acesso ao sistema de arquivos. O estabelecimento dessa ação é viabilizado através da seguinte primitiva:

```
fslimit(processID, policy)
```

A primitiva `fslimit` estabelece a política de controle de acesso `policy` para o processo `processID`. Uma política de controle de acesso é composta por um nome e por um conjunto de pares, onde cada par é formado por um caminho no sistema de arquivos e pela permissão de acesso concedida ao mesmo. Três tipos diferentes de permissões podem ser relacionadas a um caminho. As permissões, descritas em ordem crescente de restrição, são: w(escrita e leitura), r(leitura) e n(nenhuma). O exemplo mostrado na Figura 6.7 ilustra uma política de controle de acesso chamada `pol` para um sistema Unix. A política `pol` libera acesso de leitura a qualquer arquivo do sistema através do par `/ r`, nega qualquer tipo de acesso para arquivos do diretório `/home` e permite acesso de escrita e leitura para o diretório `/tmp` através da regra `/tmp w`.

```
pol:
      /       r,
      /home   n,
      /tmp    w;
```

Figura 6.7: Exemplo de uma política de controle de acesso.

As políticas de controle de acesso utilizadas como argumento na primitiva são preestabelecidas na política de resposta. É importante ressaltar, que esse tipo de ação somente incrementa as restrições impostas pelo mecanismo de controle de acesso nativo do sistema, ou seja, jamais libera acesso a algo normalmente restrito. Assim, no exemplo descrito anteriormente, se o mecanismo de controle de acesso nativo do sistema negar acesso ao diretório `/tmp`, uma ação `fslimit` que invoca a política `pol` não libera o acesso de escrita e leitura nesse diretório para o processo alvo da ação.

6.2.2.1.3 Ações de emergência

Duas ações com maior potencial são disponibilizadas pelo mecanismo. Essas ações de contenção, chamadas de ações de emergência, em geral são executadas quando os outros tipos de ações não são capazes de conter o efeito da intrusão. Com as ações de emergência, almeja-se parar qualquer atividade em andamento de um processo identificado como intruso. As ações são detalhadas a seguir.

Finalizar a execução do processo

A ação que finaliza a execução de um processo é interessante de ser executada em situações extremas, onde as ações já lançadas não produziram o resultado esperado. Sua maior desvantagem é que, ao contrário de todas as outras ações do mecanismo, não possui um esquema reversível. A primitiva invocada nessa ação é a seguinte:

```
kill(processID)
```

A primitiva `kill` termina a execução do processo `processID`.

Parar a execução temporariamente

Essa ação, menos agressiva que a ação anterior, pára temporariamente a execução do processo, permitindo um posterior retorno de sua execução. A primitiva para execução da ação é a seguinte:

```
stop(processID,status)
```

A primitiva `stop` pára temporariamente a execução do processo `processID`, se `status` for *true*. Caso contrário, é suposto que a execução já está parada e, com isso, a primitiva permite ao processo retornar sua execução.

Uma grande vantagem nesse tipo de ação com relação a anterior é que, além da possibilidade de retorno da execução, torna viável a realização de uma análise forense[2] no processo alvo. As evidências obtidas na análise podem, então, ser utilizadas na reconstrução e restauração do sistema e no aprimoramento da segurança, prevenindo uma nova exploração da vulnerabilidade e evitando outra perpetração.

6.2.2.2 Nível de rede

Quando um alarme disparado pelo sistema de detecção envolve atividades controladas por uma máquina remota, é importante que a resposta inclua ações de contenção para impedir a troca de informações entre as máquinas envolvidas. Para isso, o mecanismo conta com as ações no nível de rede que tem como objetivo impedir e conter o progresso da invasão através do estabelecimento de restrições no tráfego de rede. As ações de contenção do nível de rede e suas primitivas são descritas como segue.

6.2.2.2.1 Bloquear tráfego

Essa ação bloqueia o tráfego de rede, proveniente de uma determinada máquina, destinado a um determinado serviço. Bloquear um serviço para uma máquina pode ser importante para evitar sua exploração por parte do atacante e, com isso, diminuir o poder do ataque na máquina alvo da ação. A primitiva para essa ação é a seguinte:

```
ipblock(src, dport, protocol, status)
```

[2]Forense computacional é o ramo da criminalística que compreende a aquisição, preservação, restauração e análise de evidências computacionais, quer sejam componentes físicos ou dados que foram processados eletronicamente e armazenados em mídias computacionais[77]

A primitiva `ipblock` bloqueia todo o tráfego originado na máquina `src` para a porta `dport` do destino quando o argumento `status` é *true*. Se a primitiva é invocada com *false* é suposto que a ação já foi previamente imposta e, com isso, é retirada.

6.2.2.2.2 Limitar a quantidade de conexões

Alguns ataques de negação de serviço exploram máquinas, estabelecendo uma grande quantidade de conexões para algum serviço que atende requisições em uma porta TCP específica[72]. Para casos como o mencionado, o mecanismo disponibiliza a ação de contenção que limita a quantidade de conexões tcp simultâneas. Essa ação é acessada pela seguinte primitiva:

```
tcplimit(src, dport, max, status)
```

A primitiva `tcplimit` limita a quantidade de conexões simultâneas com origem em `src` e destino na porta `dport` em `max` conexões. O argumento `status` é similar ao da primitiva `ipblock`.

6.2.2.2.3 Bloquear conexões de um usuário

Outra ação de contenção do nível de rede consiste em impedir que um usuário estabeleça conexões. Esse tipo de ação difere das já descritas, pois é concentrada nas conexões com origem no máquina onde a ação é aplicada, ao contrário das anteriores que selecionam o tráfego que chega à maquina.

```
useroutblock(uid, status)
```

Essa primitiva, diferente das já apresentadas, aborda as conexões com origem na máquina onde é aplicada a regra, impedindo que o usuário `uid` estabeleça conexões.

6.3 Resposta Primária

A resposta não específica do mecanismo, denominada resposta primária, é executada pelo executor da resposta primária e estimulada pela parte do sistema de segurança reponsável pela detecção de anomalias. A detecção de anomalia não reconhece um tipo específico de ataque, ao invés disso, considera uma intrusão como a ocorrência de comportamentos anômalos no sistema. Com isso, considerando a limitação de informações providas por esse tipo de detecção, as ações da resposta primária são baseadas nos possíveis riscos que o ataque é capaz de proporcionar. O potencial de dano do ataque, chamado de **nível de risco**, é elaborado pelo sistema de detecção de anomalias com base nos comportamentos anormais encontrados e fornecido ao mecanismo de resposta sempre que uma resposta

primária é iniciada. O agente responsável então consulta a política de resposta onde estão definidos todos os níveis de risco, bem como as ações e suas intensidades que determinam a estratégia de execução para a resposta.

Para aprimorar a resposta primária, foram incorporados algoritmos adaptativos de execução para algumas das ações. Tais algoritmos alteram dinamicamente a intensidade da ação e também o nível de risco do ataque de acordo com estímulos recebidos. Por isso, cada ação executada na resposta primária possue um nível de risco específico.

Além do nível de risco, o início de uma resposta primária requer a identificação do alvo das ações de contenção. Esse alvo pode ser tanto um processo, um máquina, um serviço e um usuário. A seguir estão descritas as características e o funcionamento de um processo de resposta primária.

6.3.1 Adaptabilidade

A resposta primária, por ser inespecífica e basear-se principalmente no nível de risco do ataque, inclui em suas ações um esquema adaptativo. Esse esquema adaptativo permite que a intensidade de uma ação de contenção varie gradativamente de acordo com estímulos. O tipo do estímulo é específico para cada esquema de ação. Os algoritmos de adaptabilidade, descritos a seguir, ajustam a intensidade da ação em períodos regulares de tempo denominados **ciclos**, podendo também executar uma ação de emergência quando necessário. A seguir estão descritos os algoritmos de adaptação dinâmica para as ações na resposta primária.

6.3.1.1 Controle do consumo do processador

A ação de contenção que controla o consumo máximo do processador, adota um esquema de adaptabilidade na resposta primária, permitindo que o processo utilize o processador de acordo com sua disponibilidade. Para isso, dois algoritmos distintos são propostos. Esses algoritmos não dependem diretamente do esquema utilizado pelo escalonador de processos e determinam a cada ciclo a porcentagem máxima de uso do processador para o processo envolvido no susposto ataque.

A tarefa essencial dos algoritmos é a de fixar o limite máximo de consumo do processador com base na quantidade disponível de processamento, ou seja, observando a utilização do processador pelo restante dos processos em execução no sistema.

Primeiro algoritmo

Esse algoritmo atribui o limiar de consumo de um processo de acordo com os valores estabelecidos na definição dos níveis de risco. Nele, um processo é iniciado com o limiar correspondente de acordo com o seu nível inicial de risco. O processo então executa

durante um ciclo, restrito pelo limiar imposto. Se, ao final desse período, o processador apresentar algum tempo de ociosidade, o processo pode ter seu nível de risco diminuído ou continuar com o mesmo nível dependendo da política de resposta. Entretanto, se o processador não apresentar tempo de ociosidade, o processo tem seu nível de risco aumentado e, com isso, o limiar de consumo torna-se mais restrito. Se o processo alcançar seu nível máximo de risco durante certo tempo, uma ação de emergência pode ser invocada. O pseudo-código simplificado do algoritmo é mostrado na Figura 6.8.

```
enquanto()
{
   nivel = nívelinicial;
   enquanto( nivel >= nivelminimo && nivel <= nivelmaximo )
   {
      consumo = ConsumoMax(nivel);
      cpulimit(procID, consumo);
      wait(ciclo);
      se CPUIdle() entao
         nivel --;
      senao
         nivel ++;
   }
   se nível < nivelminimo então
   {   //processo vai para o estado de suspeito
      cpulimit(procID, -1);
      while(cpuIdle())
      {
         wait(ciclo);
      }
   }
   senão se nível > nivelmaximo então
      stop(procID, true);
}
```

Figura 6.8: Primeiro algoritmo de controle dinâmico do consumo do processador.

Segundo algoritmo

O segundo algoritmo aborda o problema de um modo diferente. O tempo máximo de uso do processador, nesse caso, é dimensionado somente com base na observação do tempo de ociosidade do processador durante um período. No algoritmo, sempre que o processador não apresentar algum período de ociosidade, o processo alvo fica sem utilizá-lo e seu tempo

de ociosidade é medido ao longo de um ciclo. O limite é então estabelecido de acordo com os valores observados e o processo retorna sua execução. O pseudo-código da Figura 6.9 descreve o algoritmo simplificadamente.

```
consumo = 0;
enquanto(1)
{
   cpulimit(procID, consumo);
   wait(ciclo);
   away = 0;
   se cpuIdle() então
   {
      consumo = consumo + cpuIdlePercent();
   }
   senão
   {
      cpulimit(procID,0);
      enquanto(!cpuIdle())
      {
         wait(ciclo);
         away++;
      }
      se( away > awayMax) então
      {
         stop(procID);
         exit();
      }
      senão
         consumo = cpuIdlePercent();
   }
}
```

Figura 6.9: Segundo algoritmo de controle dinâmico do consumo do processador.

6.3.1.2 Memória

A quantidade de memória consumida por um processo envolvido em um alarme também é dinamicamente controlada por um algoritmo adaptável na resposta primária. Esse algoritmo determina a quantidade máxima de memória alocada permitida ao processo, de acordo com a quantidade disponível no sistema. O nível de risco estabelece, para esse algoritmo, o limite máximo de consumo inicial e o mínimo de memória livre disponível no sistema.

A tarefa do algoritmo é restringir a alocação de memória do processo, de modo a reservar constantemente uma quantidade de memória disponível no sistema, para atender ao restante dos processos em execução. Com isso, o cálculo do limite de consumo de memória do processo envolve os seguintes valores: quantidade de memória disponível no sistema e a quantidade mínima necessária de memória livre. Se, em determinado instante, a quantidade de memória disponível no sistema torna-se inferior ao mínimo requerido então o processo tem sua execução interrompida até que o limite inferior seja atingido através da liberação de memória por outros processos em execução. Entretanto, se o processo exceder seu limiar de consumo então uma ação de emergência é invocada.

O pseudo-código apresentado na Figura 6.10 descreve o algoritmo.

```
nivelMem(nivel, maxPermitidoProc, MinMemoriaSistema);
memlimit(procID, maxPermitidoProc);
away = 0;
enquanto()
{
   wait(ciclo);
   max = (MemoriaLivre() - MinMemoriaSistema) + max;
   se max < maxPermitidoProc então
   {
      away = 0;
      memlimit(procID, max);
   }
   senão
   {
      away = away + 1;
      se away > maxaway então
      {
         stop(procID, true);
         exit();
      }
      memlimit(procID, maxPermitidoProc);
   }
   wait(ciclo);
}
```

Figura 6.10: Algoritmo de controle dinâmico da alocação de memória.

6.3.1.3 Sistema de arquivos

Esse tipo de ação restringe o acesso aos arquivos de um sistema. Seu esquema adap-

tativo é dependente das políticas de controle de acesso preestabelecidas, onde existe uma política associada a cada nível de risco do ataque. A ação, inicialmente, impõe a política de controle de acesso correspondente ao nível de risco fornecido. Dependendo da quantidade de acessos negados, o processo pode ter o seu nível de risco incrementado e, com isso, sujeitar-se a uma política de controle de acesso mais restritiva. Se o processo exceder o limiar de acessos negados imposto pelo último nível de risco, então a ação de emergência é executada. O algoritmo da Figura 6.11 detalha o esquema adaptativo.

```
fslimit(procID,policy[nivel]);
enquanto()
{
   wait(ciclo);
   se(limiar[nivel] < acessosnegados) então
   {
      se(nivel == nivelMax) então
      {
         stop(procID, true);
         exit();
      }
      senão
      {
         nivel = nivel + 1;
         fslimit(procID,policy[nivel]);
      }
   }
}
```

Figura 6.11: Algoritmo adaptativo de controle de acesso.

6.3.1.4 Nível de rede

A adaptabilidade, para as ações do nível de rede, concentra-se no esquema de revogação de ações através de requisições remotas. Em um processo de resposta primária, se uma máquina remota é envolvida na detecção, o agente de reação consulta a política de resposta para verificar quais são as ações a serem invocadas de acordo com o nível do ataque. Assim que são definidas, as ações são lançadas e podem ser removidas se houver alguma requisição remota para tal. Uma requisição é efetuada quando algum usuário ou processo não envolvido na detecção deseja estabelecer conexões com a máquina que executou a ação. Nesse caso, antes de remover a ação, o grafo de dependência é consultado e, de acordo com o valor, a ação pode ser removida.

6.3.2 Reação direta

A reação direta é disparada pelo sistema de detecção de anomalia e, com base na política de resposta e no nível de risco do ataque, são executadas as ações de contenção. Nesse caso, o directReactionAgent recebe o nível de risco do ataque e os alvos da resposta. As ações são então invocadas e permanecem em execução até que o administrador intervenha ou alguns dos seguintes eventos ocorram:

- nível de processo: alguma das ações adaptativas se intensifica até que uma ação de emergência seja executada;
- nível de rede: as ações são retiradas por solicitações de pontos de resposta remotos.

6.3.3 Reação estimulada

A reação estimulada é disparada quando um alarmAgent notifica o início de uma reação direta em um ponto de resposta remoto. Uma reação estimulada pode alertar o sistema de detecção de seu ponto de resposta sobre a possível ocorrência de um ataque na rede, fazendo com que a detecção torne-se mais aguçada. Além disso, uma reação estimulada pode ser importante para impedir a propagação do poder do ataque e, para isso, a política de resposta pode especificar, para uma reação estimulada, ações do nível de rede de acordo com o nível de risco do ataque.

Juntamente com as características já citadas, que são o impedimento da propagação do ataque e a ativação do sistema de detecção, a reação estimulada pode envolver um processo de revogação de ações de uma reação direta. Esse tipo de negociação é importante pois, em alguns casos, as ações do nível de rede de uma reação direta podem interferir na comunicação entre máquinas e prejudicar usuários e processos legítimos. Quando esse fato ocorre, o stimulatedReactionAgent notifica o directReactionAgent correspondente ao mesmo processo de resposta e solicita a revogação das ações diretas do nível de rede que estão interferindo na comunicação.

6.4 Resposta Secundária

A resposta secundária é estimulada pela detecção de mau uso do sistema de segurança. Por ser baseada em assinaturas, a detecção de mau uso detecta um tipo específico de ataque. Como a forma de ação da invasão é conhecida, é possível elaborar e definir antecipadamente um esquema de resposta. Conforme especificado no Capítulo 5, essas respostas preestabelecidas são armazenadas na base de dados de assinaturas e, quando solicitadas, são buscadas pelo executor da resposta secundária com base na assinatura do

ataque. Após obter o esquema de resposta, o executor da resposta secundária executa as ações de contenção.

Para dar início a uma resposta secundária, é necessário que o secondaryResponseAgent de algum ponto de resposta da rede seja acionado. Esse agente invoca então os agentes responsáveis pela reação de contenção e propagação do alarme recebido.

6.4.1 Adaptabilidade

A resposta secundária é sempre iniciada pelo sistema de detecção de mau uso e, como esse tipo de detecção geralmente apresenta taxas menores de falso positivos se comparada a detecção de anomalia[11], possui uma estratégia de resposta preestabelecida em uma base de dados. Por esse motivo, a resposta secundária não apresenta um esquema adaptável como o da resposta primária, restringindo-se a revogação de ações do nível de rede através de uma reação estimulada.

Entretanto, é importante evidenciar que um ataque pode ser detectado por ambos sistemas de detecção, fazendo com que o mecanismo de resposta inicie tanto uma resposta primária como uma secundária para o mesmo ataque. Com isso, além da resposta fixada pela base de dados, as ações adaptáveis da resposta primária também são aplicadas.

6.4.2 Reação direta

Na reação direta da resposta secundária, o directReactionAgent encarregado das ações de contenção recebe a assinatura do secondaryResponseAgent e busca na base de dados de assinaturas a estratégia de resposta referente a assinatura. Uma estratégia de resposta armazenada na base de dados de assinaturas é composta por um conjunto de primitivas com seus respectivos parâmetros. Com a estratégia de resposta obtida, o agente então executa as ações de contenção. Eventualmente, o directReactionAgent pode ser notificado por algum stimulatedReactionAgent para a revogação de algumas ações do nível de rede.

6.4.3 Reação estimulada

A reação estimulada da resposta secundária é similar a da resposta primária. Nesse caso, uma reação estimulada pode alertar o sistema de detecção de mau uso de seu ponto de resposta sobre a possível ocorrência de um ataque na rede, informando a assinatura do ataque. A estratégia de resposta pode especificar, além das ações da reação direta, um conjunto de ações de contenção para a reação estimulada. Esse tipo de ação é importante principalmente para evitar a propagação do ataque. Por fim, a reação estimulada pode envolver um processo de revogação de ações de uma reação direta semelhante ao da resposta primária.

6.5 Política de resposta

O mecanismo de reposta requisita que, para cada ponto de resposta, seja estabelecida uma política de resposta. Essa política define as estratégias de execução das respostas primárias e secundárias. Ao invés de especificar uma única política para todos os pontos de resposta, o mecanismo adota um esquema de classificação dos pontos de resposta e então define uma política de resposta para cada uma das classes.

Para distinguir cada ponto de resposta, o mecanismo leva em conta que cada máquina possui sua função específica e importância na rede, e por isso requisita estratégias diferentes de respostas. A distinção dos pontos de resposta é baseada na taxonomia de intrusões apresentada na Seção 2.6.2, que classifica os possíveis alvos de um ataque em quatro classes distintas: **servidor externo, servidor interno, estação de trabalho e componente de rede**. Toda máquina da rede é enquadrada na classificação e, com isso, tem sua política de resposta definida.

6.5.1 Política para a resposta primária

A resposta primária é executada com base na definição dos possíveis níveis de risco que uma intrusão pode assumir. Com isso, a política de resposta para a resposta primária é concentrada principalmente na definição desses níveis.

6.5.1.1 Nível de processos

Na resposta primária, cada nível de risco define a intensidade das ações do nível de processo e, com isso, capacitam a adaptabilidade das respostas.

- **Consumo do processador**. O primeiro algoritmo controla o consumo máximo do processador, e necessita que para cada nível seja definido o limite de consumo. Já o segundo algoritmo requer a definição do tempo máximo que um processo pode ficar sem utilizar o processador, representado no código da Figura 6.9 pela constante `awayMax`.

- **Consumo de memória**. O algoritmo que limita o consumo de memória exige a definição de dois parâmetros para cada nível de risco do ataque: a quantidade mínima de memória disponível no sistema e a quantidade máxima de memória alocada permitida a um processo. Esses parâmetros são representados respectivamente pelas constantes **MinMemoriaSistema** e **maxPermitidoProc** da Figura 6.10.

- **Quantidade de processos filhos**. Para esse tipo de ação, a política de resposta relaciona para cada nível um limiar máximo de processos filhos.

- **Quantidade de conexões**. Similar a ação anterior, a política de resposta contém em cada nível de risco a quantidade máxima de conexões simultâneas.
- **Sistema de arquivos**. Esse tipo de ação restringe o acesso aos arquivos de um sistema estabelecendo, para cada nível, uma política de acesso. Além disso, cada nível conta com a quantidade máxima permitida de acessos negados.

6.5.1.2 Nível de rede

A política de resposta estabelece, para cada nível do ataque, um conjunto de ações do nível de rede. Cada nível conta com as ações da reação direta e, quando for o caso, com as ações da reação estimulada.

6.6 Nível de Confiança

Dependendo do alvo da resposta, existe o risco das ações de contenção prejudicarem usuários ou processos legítimos, mesmo que uma resposta não seja lançada na detecção de um falso positivo. Para diminuir a chance das ações prejudicarem tais usuários ou mesmo a segurança do sistema, o mecanismo de resposta introduz um método de avaliação do impacto de uma resposta no sistema. Esse método calcula um valor numérico entre 0 e 1 denominado **nível de confiança** da resposta.

Sua aplicação restringe-se as ações do nível de rede e seus possíveis valores são calculados antes do mecanismo de resposta estar apto a responder, fazendo com que o nível de confiança sempre influencie na resposta. O nível de confiança pode ser consultado em dois momentos diferentes da execução de uma ação. No primeiro caso, é consultado para decidir se uma determinada ação será lançada no sistema. No segundo caso, o nível de confiança é consultado quando o agente recebe uma requisição para revogar uma ação, influênciando na decisão. A seguir estão descritos os métodos abordados para o cálculo do nível de confiança.

6.6.1 Grafo de dependência

O nível de confiança nas ações do nível de rede é determinado com o auxílio de um grafo, chamado grafo de dependência, que representa a utilização dos serviços disponibilizados pelas máquinas da rede. O grafo G de dependência é um grafo orientado, ponderado e bipartido com as partições A e B, onde cada vertice $u \in A$ representa uma máquina da rede e cada vértice $v \in B$ representa um serviço disponibilizado por uma determinada máquina. Os vértices $u \in A$ são caracterizados simplesmente por uma identificação única da máquina, como seu endereço de rede, enquanto que os vértices $v \in B$ são caracterizados

por um par formado pela identificação da máquina e pelo serviço disponibilizado. As arestas E do graf são da forma (u, v), sendo que o peso de qualquer aresta $(u, v) \in E$ representa a dependência do serviço v pela máquina u.

Para construir G é preciso primeiro definir as partições A e B, de modo que A inclua todas as máquinas da rede e B inclua todos os serviços utilizados pelas máquinas $u \in A$. A seguir, com os vértices de G definidos, são determinadas as arestas $(u, v) \in E$ através da verificação de quais serviços v estão disponíveis para u. A métrica que determina o valor do peso das arestas é a freqüência do uso do serviço v por u. Essa frequência é contabilizada através da quantidade de requisições enviadas por u à v durante um período t de tempo. A ausência de uma aresta (u, v) indica que o serviço v não está disponível para u.

Para o cálculo do nível de confiança, é suposto que o possibilidade de uma ação de contenção prejudicar usuários legítimos, que desejam usar o serviço v de uma máquina u, é diretamente proporcional à dependência de u por v. Com isso, o nível de confiança é obtido através de uma função $f(x, z) = max(0, 1 - \frac{x}{z})$, onde x é o peso de uma aresta (u, v) e z é um limite superior de requisições para o serviço disponibilizado em v. Como a função f indica, é com base em z que x é avaliado para o cálculo do nível de confiança. Esse limite superior z representa a quantidade de requisições, durante um período t, que caracterizam um serviço como essencial para a rede. Cada tipo de serviço conta com um valor de z específico que é definido pelo administrador com base nas características da rede.

6.7 Analogias com o sistema imunológico humano

A modelagem do mecanismo de resposta, assim como a modelagem do sistema de segurança imunológico, também é fortemente baseada nos conceitos do sistema imunológico humano. Essa fundamentação é interessante principalmente para absorver os princípios organizacionais do sistema imunológico humano, descritos na seção 3.2.1, que são desejáveis para um sistema de detecção de intrusão e resposta. A seguir, agentes e esquemas do mecanismo são relacionados a componentes e processos de defesa do sistema imunológico.

PrimaryResponseAgent

O primaryResponseAgent é o agente responsável por gerenciar a resposta primária do mecanismo, que é o processo de resposta não específico. Esse agente assemelha-se em funcionamento aos linfócitos T e B não específicos presentes no corpo humano. Tais células monitoram o corpo a procura da presença de antígenos, apesar de ainda não ter encadeado ou ser originada de nenhum processo de resposta e portanto não ser específica na detecção

de algum antígeno em particular. Entretanto, esses linfócitos também produzem uma reação, apesar de não ser tão rápida e eficiente quanto a reação conduzida pelos linfócitos de memória. Além disso, desempenham um papel importante para a defesa do corpo pois capacitam uma resposta a doenças ainda não contraídas pelo corpo.

SecondaryResponseAgent

O secondaryResponseAgent é o agente responsável por gerenciar a resposta específica estimulada pela detecção de mau uso, a resposta secundária do mecanismo. A funcionalidade desse agente pode ser comparada a dos linfócitos T e B de memória do sistema imunológico, que são as células do corpo que já se depararam com algum antígeno ou foram criados ao longo de um processo de resposta e são, portanto, específicas para um antígeno. Isso não impede que esse tipo de célula inicie uma resposta para outro antígeno com estrutura similar ao específico. Com tais características, a célula é capaz de responder rápida e eficientemente a uma invasão.

Outra similaridade que pode ser apresentada é o processo de apresentação do antígeno efetuado pelo macrófago. Nesse processo é extraída a assinatura do agente infeccioso e exposta aos linfócitos. A apresentação do antígeno é similar a ativação de uma resposta secundária, efetuada através do recebimento de alguma mensagem, contendo a assinatura de um ataque, proveniente do sistema de detecção. Assim como no corpo humano, é através desse estímulo que todo o processo de resposta é desencadeado.

ReactionAgent

Os reactionAgents são os agentes responsáveis por executar e conduzir as reações direta e estimulada do mecanismo. Suas funcionalidades são similares aos linfócitos T e B criados quando algum processo de resposta do sistema imunológico humano é iniciado através da estimulação de uma célula de defesa. Quando alguma resposta imunológica tem início, as células ativadas reproduzem-se, gerando o exército para combater a infecção. Similarmente, quando um responseAgent é ativado, ele cria os reactionAgents e os alarmAgents para combater o ataque e conter as partes afetadas do sistema.

Além disso, as ações de contenção lançadas por tais agentes desempenham um papel similar ao dos anticorpos e das toxinas, criados respectivamente pelos linfócitos B plasmáticos e pelos linfócitos T citotóxicos. Como os anticorpos em geral são mais específicos, podem ser relacionados às ações de um processo de reposta secundário, ao contrário das toxinas liberadas pelas células T citotóxicas que são melhor equiparadas, por serem mais genéricas, às ações das respostas primárias.

Por fim, o pedido de remoto de revogação de ações realizados pelos reactionAgents são baseados nas células T supressoras do sistema imunológico. As células supressoras,

inibem o processo de resposta do corpo humano através da liberação de substâncias.

AlarmAgent

Um alarmAgent é sempre incumbido de propagar o alarme disparado em um ponto de resposta, proporcionando uma reação distribuída e ativando assim outros pontos de resposta. Sua função assemelha-se a dos linfócitos T ajudantes que ativam outras células do sistema imunológico através da liberação de citozinas, equivalentes em funcionalidade às mensagens enviadas pelo alarmAgents.

Nível de confiança

O nível de confiança tem como objetivo dimensionar a dependência de uma máquina por um serviço. Dependendo do valor do nível, uma ação pode não ser lançada ou ser revogada. A consulta desse valor é importante para evitar que usuários e processos legítimos sejam prejudicados com as ações de contenção executadas.

O sistema imunológico apresenta uma estratégia semelhante a do mecanismo de resposta através da coestimulação. A coestimulação força um linfócito T a aguardar por um segundo sinal antes de iniciar uma resposta imunológica. Esse processo é importante, pois evita que o desencadeamento de reações auto-imunes no corpo, onde células *self* são erroneamente reconhecidas como *non-self*[75].

6.8 Segurança do mecanismo de resposta

Um assunto pouco abordado na área de segurança, é o fato de que um sistema de detecção de intrusão também pode ser alvo de um ataque. Em geral, esse tipo de ataque é disparado com duas motivações principais: impedir a detecção ou manipular o mecanismo de resposta. No primeiro caso, um ataque procura desabilitar os componentes de detecção para esconder sua atuação no sistema. No segundo caso, o ataque é disparado contra o sistema de detecção ou contra o mecanismo de resposta para causar a execução de ações de contenção no sistema, prejudicando usuários legítimos através de uma negação de serviço.

Para evitar tal acontecimento, o mecanismo de resposta deve adotar técnicas de segurança de modo que dificulte sua exploração por parte de um atacante. As seções seguintes descrevem sucintamente as questões pertinentes à segurança do mecanismo de resposta.

6.8.1 Segurança no componente de controle/comunicação

A segurança do componente de controle/comunicação concentra-se basicamente em garantir a segurança dos agentes móveis e da plataforma base para a execução dos mesmos.

Entretanto, para que um sistema de agentes móveis seja considerado seguro, ele deve atender a requisitos de segurança, tais como: confidencialidade, integridade, autenticidade, não-repudio, disponibilidade e anonimato[99]. Além disso, é importante que o canal de comunicação utilizado pelo mecanismo de resposta seja seguro, dificultando o comprometimento das reações. A Seção 4.6.2 já aborda o assunto, onde são sugeridos alguns métodos de prevenção das ameaças para agentes móveis aplicados a resposta automática.

6.8.2 Segurança no componente local de reação

A segurança do componente local de reação consiste, basicamente, em evitar que entidades do sistema, com exceção dos reactionAgents e do administrador, tenham acesso as primitivas das ações. Tal proteção é dependente do modelo de implementação adotado. Mas, geralmente restringe-se a proteger o acesso a determinadas aplicações.

6.9 Conclusão

O modelo do mecanismo de resposta automático apresentado no presente capítulo corresponde a um detalhamento dos componentes responsáveis pelas respostas disparadas pelo sistema de segurança imunológico. Os componentes do sistema de segurança abordados no mecanismo de resposta são o executor da resposta primária e o executor da resposta secundária. O mecanismo de resposta engloba dois tipos diferentes de resposta: primária e secundária. A resposta primária, disparada pelo sistema de detecção de anomalia, não é específica e é baseada no possível risco que o ataque oferece ao sistema. As estratégias de resposta relacionadas aos possíveis riscos são estabelecidas na política de resposta do mecanismo. Já a resposta secundária, disparada pelo sistema de detecção de mau uso, é específica, pois é baseada na assinatura do ataque. Além disso, independente do tipo de resposta, o mecanismo executa suas reações de modo a torná-las adaptáveis e distribuídas. Tais características contribuem para uma resposta mais robusta e eficaz.

Capítulo 7

Implementação e Resultados

A implementação de um protótipo do modelo do mecanismo de resposta automático proposto no capítulo anterior é importante, pois permite a observação de seu funcionamento na prática. Além disso, também é possível observar a interação entre seus componentes e o impacto que a execução de uma resposta pode provocar no sistema.

O protótipo do mecanismo de resposta conta com uma implementação simplificada, que abrange a implementação do componente de controle/comunicação, composto pelos agentes móveis, e a implementação de algumas ações de contenção do componente local de reação. Os agentes móveis desenvolvidos executam as funções propostas no modelo e, por isso, controlam as respostas lançadas pelo mecanismo de modo a torná-las adaptáveis e distribuídas.

As ações de contenção implementadas possibilitam a elaboração de uma resposta capaz de limitar atividades classificadas como um possível ataque. Cada ação direciona seu poder de contenção para um alvo diferente do sistema e, por isso, adota uma estratégia específica de implementação. O ambiente escolhido para implementação do protótipo foi o Linux RedHat 7.2 com *kernel* 2.4.7-10. As Seções 7.1 e 7.2 descrevem, respectivamente, a implementação do componente de controle/comunicação e do componente local de reação.

7.1 Componente de controle/comunicação

O componente de controle/comunicação é responsável por gerenciar todas as ações e efetuar as negociações entre os pontos de resposta. A implementação desse componente é indispensável para a observação da interação entre os agentes móveis do mecanismo de resposta.

O sistema de agentes móveis adotado na implementação do componente foi o Aglets Workbench versão 2.0.1, que permite a implementação de agentes na linguagem Java. O compilador e a máquina virtual Java utilizados para a execução do protótipo per-

tencem ao pacote de desenvolvimento jdk1.3.1. A escolha do Aglets Workbench para a implementação dos agentes deve-se, principalmente, à ampla biblioteca de classes disponibilizada. Essa biblioteca provê um conjunto de classes que facilita a implementação de funcionalidades como troca de mensagens, mobilidade, clonagem e interação com a plataforma.

Além disso, o Aglets Workbench ainda dispõe de uma plataforma já implementada, chamada Tahiti, que foi adotada para hospedar os agentes nos pontos de resposta. Na plataforma, a interação entre os agentes pode ser efetuada através de troca de mensagens, ou através da passagem de parâmetros quando um agente é criado por outro. Uma descrição sobre o sistema de agentes utilizado é apresentada na Seção 4.4.1. A seguir estão descritos o funcionamento dos agentes do mecanismo, a simulação de sua interface com os componentes de detecção e o esquema de registro de eventos.

7.1.1 Funcionamento

Como descrito na seção 6.2.1, primeiramente o initAgent é criado para obter a configuração do mecanismo de resposta e, em seguida, cria os responseAgents, baseado nessa configuração. A configuração é composta por um conjunto de arquivos, que são acessados pelo agente em sua criação e representam a política de resposta do mecanismo. Depois de conhecida a configuração, os responseAgents são criados.

O modelo discutido no Capítulo 6 propõe dois tipos distintos de responseAgents. Entretanto, optou-se por implementar somente um tipo genérico desse agente, ao invés do primaryResponseAgent e do secondaryResponseAgent. Essa simplificação é justificada pelo funcionamento similar apresentado nos dois agentes de gerência das respostas. Dessa maneira, a implementação de um único tipo de responseAgent já permite a observação das interações no componente.

Assim que é criado, um responseAgent migra para o ponto de resposta que lhe foi atribuído. Quando chega ao ponto, cria o agente de registro de eventos e aguarda por qualquer mensagem que estimule um processo de resposta.

7.1.2 Interface com o sistema de detecção

A interface do mecanismo de resposta com o restante do sistema de segurança dá-se através do envio de mensagens aos responseAgents pelos sistemas de detecção. Como o presente projeto não abrange os componentes de detecção do sistema de segurança, optou-se por criar uma simulação. Nesse caso, um alarme que deveria ser disparado pelos componentes de detecção é enviado através da interação com uma interface gráfica.

Essa interface, além de permitir o início de um processo de resposta, também capacita a solicitação da revogação de ações e a obtenção de um arquivo de *log* contendo os

eventos de todos os pontos de resposta. Para iniciar um processo de resposta primária, é necessário especificar seu alvo e o nível de risco apresentado pelo suposto ataque. As ações de cada nível estão especificadas em arquivos de configuração. Já, o início de um processo de resposta secundária, requer a especificação de uma primitiva e seus respectivos argumentos. Para revogar uma ação é necessário que sejam especificados os alvos e a identificação da ação. Por fim, a solicitação de um arquivo de log, contendo os eventos gerados por todos os pontos de resposta, não exige argumentos de entrada.

As requisições efetuadas através da interface gráfica são enviadas, em uma mensagem, ao responseAgent do ponto de resposta. O agente, ao recebê-la, aciona os agentes necessários para efetuar a tarefa requisistada. As figuras 7.1 e 7.2 ilustram, respectivamente, as reações direta e estimulada de um processo de resposta, mostrando os agentes criados para os dois pontos de resposta na plataforma Tahiti.

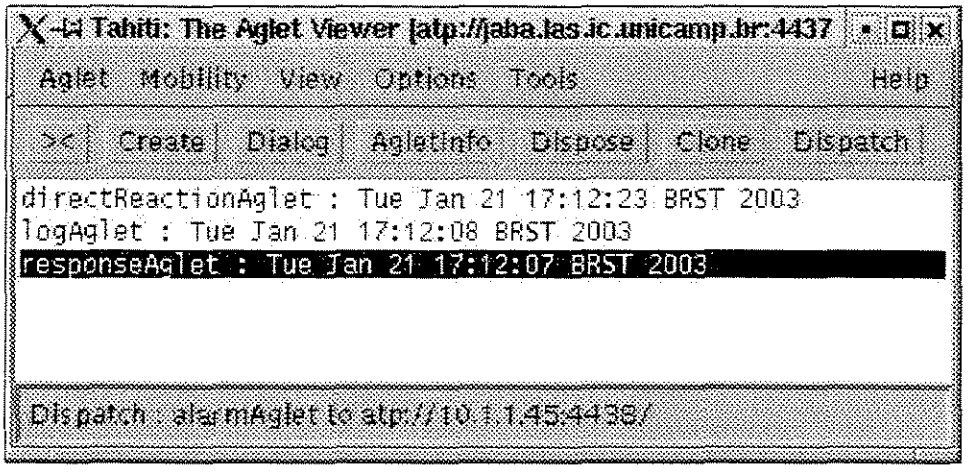

Figura 7.1: Exemplo de uma reação local no Tahiti.

7.1.3 Esquema de registro de eventos

A implementação do agente responsável pelo registro de eventos do mecanismo de resposta adota um método simples para captura de eventos. Esse método consiste em manter um logAgent, em cada ponto de resposta, aguardando mensagens constantemente. Essas mensagens são enviadas por outros agentes do mesmo ponto de resposta, onde cada uma contém informações sobre um evento ocorrido. O logAgent, ao receber uma mensagem, registra, em um arquivo, as informações sobre o evento. As figuras 7.3 e 7.4 exemplificam, respectivamente, trechos de arquivos de registro de eventos das reações direta e estimulada de um processo de resposta.

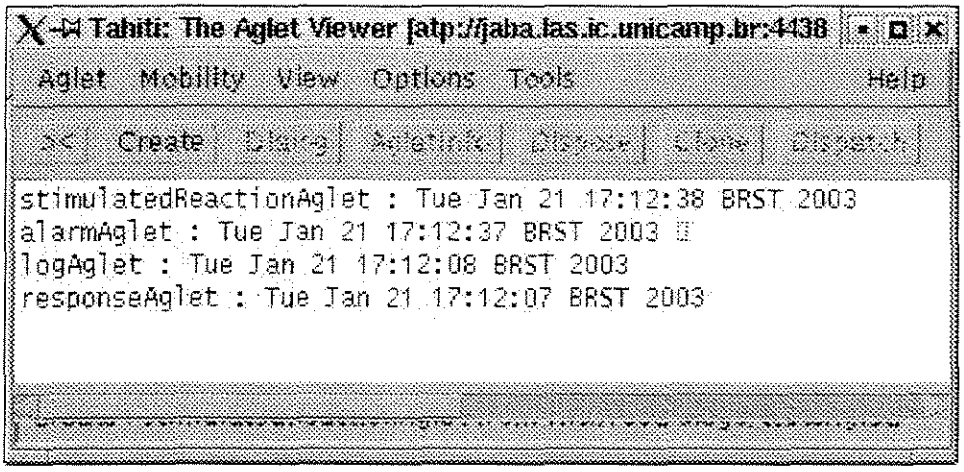

Figura 7.2: Exemplo de uma reação remota no Tahiti.

7.2 Componente local de reação

O componente local de reação é composto pela implementação das ações de contenção. As seguir, estão descritos os métodos adotados para a implementação das ações do nível de rede e processos. Apesar do componente não dispor da implementação de todas as ações propostas no modelo, o conjunto de ações implementadas já é suficiente para validá-lo.

7.2.1 Nível de rede

As ações do nível de rede utilizam o filtro de pacotes iptables[51] para a implementação das primitivas. Nesse nível, a aplicação e a revogação de ações são implementadas, respectivamente, como a inserção e remoção de regras do filtro. Algumas das regras criadas necessitam da opção `-m iplimit` para quantificar as conexões simultâneas estabelecidas na máquina. Essa opção demanda a instalação de um *patch* para o *kernel*, denominado *patch-o-matic*.

A aplicação e revogação de uma ação são expressas através da inserção e remoção de regras do filtro de pacotes. Esses dois eventos são diferenciados por um único argumento em uma regra: `-I` no caso da inserção e `-D` no caso da remoção. A implementação das primitivas descritas a seguir apresentam somente regras que expressam a aplicação de ações.

7.2.1.1 ipblock(src,port,protocol,status)

A primitiva `ipblock(src,port,protocol,status)` impede que uma máquina remota

```
1/21/03 12:03:53 - responseAglet: Message received type respond
1/21/03 12:03:53 - responseAglet: mail sent to root@localhost
1/21/03 12:03:53 - responseAglet: directReactionAglet created
                                  successfully
1/21/03 12:03:53 - responseAglet: alarmAglet to 10.1.1.46:4438
                                  created successfully
1/21/03 12:03:54 - alarmAglet: dispatch successfully
1/21/03 12:03:54 - alarmAglet: responder in 10.1.1.46:4438
                               notified
1/21/03 12:03:54 - directReactionAglet: Error on action execution:
                                        kill(1212)
1/21/03 12:03:54 - directReactionAglet: Action executed
1/21/03 12:04:04 - directReactionAglet: Execution finalized
```

Figura 7.3: Registro dos eventos de uma reação direta.

```
1/21/03 12:03:54 - responseAglet: Message received type alarm
1/21/03 12:03:54 - responseAglet: stimulatedReactionAglet
                                  created successfully
1/21/03 12:03:54 - stimulatedReactionAglet: Action executed
1/21/03 12:04:04 - stimulatedReactionAglet: Execution finalized
```

Figura 7.4: Registro dos eventos de uma reação estimulada

acesse um serviço disponibilizado, através da seguinte regra do filtro de pacotes:

iptables -I INPUT -p `protocol` *-s* `src` *-dport* `port` *-j REJECT*

Como um exemplo, invocando a primitiva `ipblock(10.1.1.45, 22, tcp, true)` na máquina com endereço 10.1.1.50, o seguinte comando é executado:

iptables -I INPUT -p tcp -s 10.1.1.45 -dport 22 -j REJECT

Essa regra impede que conexões, originadas da máquina com endereço 10.1.1.45, para a máquina com endereço 10.1.1.50 sejam estabelecidas através do serviço ssh (porta 22). A seguir pode ser visto no que uma tentativa de conexão resultaria, como visto do lado cliente:

```
[diego@jaba diego]\$ ssh 10.1.1.50
Secure connection to 10.1.1.50 refused.
```

7.2.1.2 tcplimit(src,port,max,status)

A primitiva `tcplimit(src,port,max,status)`, que limita a quantidade de conexões simultâneas provenientes de uma máquina remota, é implementada pela seguinte regra do filtro de pacotes:

iptables -I INPUT -p tcp -s `src` *-dport* `port` *-m state -state NEW -m iplimit -iplimit-above* `max` *-j REJECT*

Um teste simples realizado para exemplificar a corretude da regra foi limitar, na máquina com endereço 10.1.1.50, o número de conexões remotas para o serviço de ssh oriundas da máquina 10.1.1.45 em, no máximo, duas conexões. A primitiva com os argumentos `tcplimit(10.1.1.45,22,2,true)` foi então invocada, executando o seguinte comando:

iptables -I INPUT -p tcp -s 10.1.1.45 -dport 22 -m state -state NEW -m iplimit -iplimit-above 2 -j REJECT

O trecho a seguir exemplifica o resultado da ação para uma tentativa de exceder o limite imposto.

```
[diego@jaba diego]$ ssh 10.1.1.50
diego@skywalker's password:
[1]+  Stopped                 ssh 10.1.1.50
[diego@jaba diego]$ ssh 10.1.1.45
diego@skywalker's password:
[2]+  Stopped                 ssh 10.1.1.50
[diego@jaba diego]$ ssh 10.1.1.50
Secure connection to 10.1.1.50 refused.
[diego@jaba diego]$
```

Apesar de restringir a quantidade de conexões simultâneas, essa regra apresenta a limitação de contabilizar somente as conexões estabelecidas após sua imposição, tornando a ação indiferente com respeito as conexões já estabelecidas.

7.2.1.3 useroutblock(uid, status)

Nesse caso, a primitiva `useroutblock(uid,status)`, que impede um determinado usuário de estabelecer conexões com máquinas remotas, é implementada pela regra mostrada a seguir:

iptables -I OUTPUT -p tcp -m owner -uid-owner `uid` *-j REJECT -reject-with tcp-reset*

Como exemplo, a primitiva `useroutblock(diego,true)` foi aplicada para negar o estabelecimento de conexões ao usuário diego na máquina com endereço 10.1.1.45. O seguinte comando foi executado:

iptables -I OUTPUT -p tcp -m owner –uid-owner diego -j REJECT –reject-with tcp-reset

A seguir é mostrado um exemplo de atuação da regra em uma tentativa de conexão.

```
[diego@jaba diego]$ ssh 10.1.1.40
ssh: connect to address 10.1.1.40 port 22: Connection refused
[diego@jaba diego]$ ssh 10.1.1.3
ssh: connect to address 10.1.1.3 port 22: Connection refused
```

7.2.2 Nível de processos

As ações do nível de processos adotam estratégias diferentes de implementação. A seguir são descritas as primitivas implementadas, bem como a descrição das implementações e dos testes realizados.

7.2.2.1 cpulimit(processID, percent)

A implementação da primitiva que limita o consumo do processador exigiu alteração do código fonte do *kernel* do Linux. Antes de abordá-la diretamente, são mencionadas algumas características, baseadas em [89, 1], pertinentes ao sistema operacional adotado e relevantes para a compreensão da implementação.

- **estrutura dos processos**: Cada processo, no Linux, é representado por uma estrutura de dados chamada `task_struct`. Essa estrutura armazena informações a respeito de um único processo em execução, como por exemplo as referências dos arquivos utilizados, o tempo de criação do processo e o tempo consumido em processamento. Em um determinado instante, a estrutura `task_struct` do processo que está utilizando o processador pode ser referenciada pelo ponteiro `current`.

- **contagem de tempo**: Sempre que uma interrupção de relógio, ou *tick*, é recebida no Linux, a variável global do kernel chamada `jiffies` é incrementada. Desse modo, essa variável contém a quantidade de interrupções de relógio recebidas pelo sistema operacional desde o início de sua execução. Essa variável é adotada como referência de tempo para o funcionamento do escalonador de processos.

- **escalonador de processos**: Seleciona o processo mais apropriado, dentre os processos em execução, para ocupar o processador[92]. No *kernel* adotado para a implementação do protótipo, a escolha é realizada com base em uma prioridade atribuída

a cada processo, adotando o esquema *round-robin* de escalonamento. Sempre que é invocado, o escalonador busca algumas informações na estrutura `task_struct` dos processos em execução. A prioridade de um processo é armazenada na variável `priority` da estrutura. Já a variável `counter` contém a quantidade máxima, em *ticks*, de uso consecutivo do processador permitida a um processo. Essa variável é ajustada periodicamente de acordo com o valor de `priority` quando o processo é escolhido no escalonamento e é decrementada, a cada *tick*, durante o uso do processador.

A execução do escalonador pode ser iniciada em diferentes momentos durante o funcionamento do sistema operacional: quando o processo corrente abandona o processador ou ao fim de uma chamada ao sistema. Sempre que é invocado, o escalonador escolhe o processo com maior prioridade para utilizar o processador. Se o processo corrente não é o escolhido, então ele é suspendido e outro é executado. Quando isso ocorre, seu contexto é armazenado na sua estrutura `task_struct`, sendo recuperado quando o uso do processador for retomado novamente pelo processo.

Para implementar a primitiva `cpulimit(processID,percent)`, foram realizadas as seguintes modificações no *kernel* do Linux:

1. **Adição de variáveis para quantificar o consumo do processador para cada processo.**

   A estrutura alterada foi a `struct task_struct` armazenada no arquivo sched.h, onde foram adicionadas as seguintes variáveis:

```
struct task_struct{ . . .
   /* número de jiffies de quando a ação teve início */
   unsigned long limit_cpu_start;
   /* número de ticks gastos em um determinado período*/
   unsigned long num_ticks_cpu;
   /* porcentagem de consumo máximo do processador */
   long max_use_cpu;
};
```

   A porcentagem, na implementação, é representada pelo inteiro `max_use_cpu` que varia entre 0 e 10000. A escolha de um inteiro, ao invés de uma variável de ponto flutuante, é justificado pelo fato dos registradores da unidade de ponto flutuante não serem salvos nas trocas de contexto.

2. **Alteração em trechos específicos do escalonador de forma a consultar e manipular as variáveis adicionadas na estrutura dos processos.**

   Os campos adicionados na estrutura `task_struct` são inicializados na implementação da função `do_fork()` situada no arquivo fork.c, onde foi adicionado o seguinte código:

```
p->limit_cpu_start = 0;
p->num_ticks_cpu = 0;
p->max_use_cpu = 10000;
```

   A quantificação do uso do processador é conduzida na função `update_one_process()` contida no arquivo timer.c. Essa função é invocada sempre que uma interrupção de relógio é recebida. A função atualiza a variável `counter` do processo corrente. Nela, foi adicionado o seguinte trecho de código.

```
if(p->cpu_limit_start)
{
   /* incrementa o número de ticks utilizados
      se a ação foi aplicada ao processo*/
   p->num_ticks_cpu++;
   /* verifica se o processo corrente ainda
      pode utilizar o processador*/
   if(limit_pcpu(p) < 0)
      p->need_resched = 1;
}
```

   A função `limit_pcpu()`, utilizada no código mostrado e descrita a seguir, foi criada para verificar se o limite de uso já foi excedido por um processo. Se consumo ainda não foi excedido, a função retorna o valor da variável `counter`. Caso contrário, o valor -1001 é retornado. Além disso, uma chamada da função `limit_pcpu()` também foi incluída dentro da função `goodness()`, que é invocada no escalonador para obter as prioridades de todos os processos na fila de espera.

```
int limit_pcpu(struct task_struct *tsk)
{
   unsigned long ticks_used, ticks_diff, pcpu, max_pcpu;

   /* se a ação não foi aplicada retorne
```

```
        o valor da prioridade */
    if(!tsk->start_limit_pcpu)
        return tsk->counter;

    /* Cálculo da porcentagem já usada
        pelo processo */
    ticks_diff = (unsigned long)(jiffies - tsk->cpu_limit_start + 1);
    max_pcpu = (unsigned long)(ticks_diff * tsk->max_use_cpu);
    ticks_used = (unsigned long)(tsk->num_ticks_cpu + 1);
    pcpu = (unsigned long)(ticks_used * 10000);

    /* verifica se o processo já utilizou
        a quantidade permitida*/
    if(pcpu <= max_pcpu)
        return tsk->counter;

    /* se o limite foi excedido */
    return -1001;
}
```

3. **Adição de uma chamada ao sistema para ajustar as variáveis que controlam o consumo do processador.**

   A chamada ao sistema denominada `cpulimit` foi adicionada para aplicar a ação que controla do consumo do processador. Sua única tarefa é atribuir valores adequados aos campos adicionados na estrutura de um determinado processo. A implementação da chamada é apresentada a seguir.

```
asmlinkage long sys_cpulimit(pid_t pid, long porc)
{
    struct task_struct *p;
    unsigned long flags;
    /* verifica a porcentagem */
    if(porc < 0 || porc > 10000)
        return -EINVAL;
    /* encontra o processo alvo */
    p = find_task_by_pid(pid);
    if(p == NULL)
        return -ESRCH;
```

```
    save_flags(flags);
    cli();
    p->cpu_limit_start = jiffies;
    p->num_ticks_cpu = 0;
    p->max_use_cpu = porc;
    sti();
    restore_flags(flags);

    return 0;
}
```

Para avaliar a funcionalidade adicionada ao *kernel*, foram criadas duas aplicações, que testam a execução da ação e o primeiro algoritmo adaptativo descrito na seção 6.3.1. A aplicação *control*, que lança a ação sobre um processo, foi criada para validar empiricamente a eficácia da implementação. Ao invocá-la, são passados, como parâmetro, a identificação do processo e o limite máximo de consumo permitido. O programa de teste criado para ser o alvo da resposta, consiste basicamente de um laço infinito que executa algumas operações aritméticas simples. Sem nenhuma restrição, uma instância desse programa ocupa praticamente todo o processamento disponível no sistema.

O consumo médio do processo alvo foi medido em intervalos regulares de dez segundos com auxílio da aplicação *top*. O primeiro conjunto de testes consistiu em limitar um processo que, sem restrições, apresentava consumo próximo de 100%, ou seja, não havia outros processos em execução no sistema com algum consumo relevante.

Feito isso, o segundo conjunto de testes limitava um processo alvo que concorria com vários outros processos pelo processador. Essa situação foi criada de modo que o processador não apresentasse tempo de ociosidade, ao contrário do primeiro conjunto de testes, onde o processador ficava ocioso quando o alvo era limitado. A tabela 7.1 resume os testes e resultados obtidos na aplicação da ação.

Como demonstrado na Tabela 7.1, a ação obteve resultados satisfatórios, fazendo com que o processo alvo apresentasse um consumo médio bem próximo do limite imposto. A ação é ainda mais precisa para os casos onde o processador não apresenta tempo de ociosidade.

Depois de validada a ação, o algoritmo adaptativo foi testado. Os objetivo desses testes foi principalmente verificar quais valores de ciclo[1] apresentavam melhores resultados. Para isso, foi construída uma política de resposta com limiares entre 0,5% e 50%

[1]Como descrito na seção 6.3.1, ciclo é o período de tempo decorrido entre os reajustes das intensidades das ações utilizado nos algoritmos adaptativos da resposta primária.

	Sistema sem atividades				Sistema com atividades			
					2 Processos		5 Processos	
Limite %	1	10	50	90	1	25	1	13
Média %	1,885	10,05	49,99	89,967	0,957	24,957	0,964	12,978
Desvio Padrão	3,263	1,822	1,920	2,963	0,051	0,0513	0,049	0,193
Consumo Máx. %	10,3	13,7	56	95,2	1	25	1	13,3
Consumo Mín. %	0	4,4	44,3	84	0,9	24,9	0,9	12,7

Tabela 7.1: Resultados obtidos com a primitiva cpulimit

variando em 0,5% para cada nível. A carga do processador foi simulada em 75%, e como resultado esperava-se que o programa alvo da ação adaptável apresentasse um consumo médio próximo de 25%. Entretanto, foi possível constatar que a implementação do algoritmo adaptativo tem seu efeito mais próximo do desejado conforme o tempo do ciclo é aumentado. A Tabela 7.2 sumariza os resultados do teste.

Tempo de ciclo(segs)	1	3	5	10	15	30	60	90
Consumo do alvo(%)	8,2	19,2	21,0	21,3	22,7	23,0	23,7	24
Consumo do simulador(%)	75,0	74,8	74,7	74,9	74,7	75,5	75,2	75,0
Período Ocioso(%)	16,5	5,8	4,0	3,6	1,7	1,2	0,4	0,5
Intervalo observado(segs)	90	90	90	90	90	90	180	300

Tabela 7.2: Resultados obtidos com o algoritmo adaptativo.

7.2.2.2 conectionlimit(processID,max)

A implementação da primitiva `conectionlimit(processID,max)`, por estar relacionada com o tráfego de rede, também utiliza o filtro de pacotes iptables. Similarmente às ações do nível de rede, somente a regra que implementa a inserção da ação é mostrada como segue:

iptables -I OUTPUT -p tcp -m owner -pid-owner `processID` *-m iplimit -iplimit-above* `max` *-j REJECT -reject-with tcp-reset*

Como exemplo, foi utilizado o programa octopus.c, encontrado em [29], que abre um grande número de conexões destinadas a uma porta, de modo a causar uma negação de serviço.

Como teste, o programa foi executado utilizando o serviço de ssh da máquina 10.1.1.40 como alvo do ataque. A regra foi então aplicada para limitar o processo em três conexões. A restrição imposta ao processo pode ser verificada no trecho de exemplo a seguir.

```
[root@jaba resp]# ./octopus 10.1.1.40 22
pid: 3541, desc 0
pid: 3541, desc 1
pid: 3541, desc 2
connect 3 failed.
connect: Connection refused
closing connection.
closed 2
closed 1
closed 0
```

7.2.2.3 fslimit(processID,policy)

A ação que limita o acesso ao sistema de arquivos foi implementada como um módulo de *kernel*. A idéia básica da implementação consiste em interceptar chamadas ao sistema relacionadas ao acesso do sistema de arquivos. A interceptação tem como objetivo verificar a política de controle de acesso do mecanismo de resposta antes de executar a chamada original. De forma simplificada, somente as chamadas `sys_open` e `sys_fork` foram interceptadas. A chamada `sys_open` foi interceptada para prover o controle de acesso do mecanismo. Já a chamada `sys_fork` foi interceptada para estender a política de controle de acesso a processos filhos, quando o processo criador é monitorado. Além disso, também foi adicionada uma chamada ao sistema para limitar um processo com a política de controle desejada.

No momento em que o módulo de *kernel* é carregado, as políticas de acesso são lidas de um arquivo. O processamento desse arquivo é realizado através de um *parser* que lê as políticas, armazenando-as em uma lista, onde cada elemento é uma estrutura do seguinte tipo:

```
struct tpolicy
{
  char* name; // nome da política
  struct trule* rules; // regras da política
  unsigned num_rules;  // número de regras
};
```

Cada regra, representada pela estrutura `trule`, contém um caminho no sistema de arquivos e o tipo de acesso permitido aos arquivos inclusos no caminho.

```
struct trule
{
  char  *file; // caminho
  short atype; // tipo de acesso
};
```

A ação é aplicada a um processo através da chamada ao sistema `msys_monproc`, inserida no sistema quando o módulo é carregado. Essa chamada ao sistema insere um processo em uma lista que contém os processos monitorados. Os elementos dessa lista são representados pela estrutura `tmonproc` apresentada a seguir:

```
struct tmonproc
{
  pid_t  pid; // identificação do processo
  struct tpolicy* policy; // a política de controle de acesso aplicada
  char *root_dir; // diretório raiz temporário
};
```

Depois de inserido na lista, sempre que algum processo monitorado tenta abrir algum arquivo, a chamada `msys_open` é executada. Essa chamada, antes de invocar a `sys_open` original, verifica se a política de controle de acesso associada ao processo permite que o arquivo seja acessado. A chamada `msys_open`, mostrada a seguir, intercepta a chamada `sys_open` que passa a ser invocada como `o_sys_open`;

```
asmlinkage long msys_open(const char* filename, int flags, int mode)
{
  int ret;
  struct tmonproc* monproc;
  MOD_INC_USE_COUNT;

  monproc = find_current_monproc();

  if( monproc != NULL && monproc->policy != NULL )
  {
    char atype = flags_to_atype(flags);

    if( deny(filename, monproc->policy, atype) )
```

```
    {
      monproc->denied_acc++;
      pproc = find_task_by_pid(monproc->pid);
      monproc = find_monproc(pproc->p_pptr->pid);
      while(monproc != NULL)
      {
        monproc->denied_acc++;
        pproc = pproc->p_pptr;
        monproc = find_monproc(pproc->p_pptr->pid);
      }
      MOD_DEC_USE_COUNT;
      return -1;
    }
  }
  ret = o_sys_open(filename, flags, mode);
  MOD_DEC_USE_COUNT;
  return ret;
}
```

Para implementação do algoritmo adaptativo que altera dinamicamente a política aplicada, proposto na seção 6.3.1, foi necessário adicionar uma chamada ao sistema. A chamada ao sistema `msys_dacccount` recebe como argumento a identificação de um processo e retorna a quantidade de acessos negados para o processo e seus processos filhos. Com base na chamada ao sistema adicionada, o algoritmo adaptativo proposto foi implementado.

Para a validação da implementação é apresentado um caso onde a política de acesso aplicada restringe um processo a ter somente acesso de leitura aos arquivos do sistema. A reação do sistema na tentativa de um acesso é apresentada como segue.

```
[root@jaba tmp]# ls -la
total 8
drwxr-xr-x    2 root     root         4096 Jan 20 23:20 .
drwxr-x---   39 root     root         4096 Jan 20 23:20 ..
-rw-r--r--    1 root     root            0 Jan 20 23:20 test1
[root@jaba tmp]# echo test > test1
bash: test1: Operation not permitted
```

O algoritmo adaptativo também foi implementado e reage baseado nas políticas de controle de acesso associadas aos possíveis níveis de risco.

7.2.2.4 kill(processID) e stop(processID, status)

As primitivas `kill(processID)` e `stop(processID, status)`, que representam as duas ações de emergência, são implementadas com o comando *kill* do Unix. A implementação das duas primitivas é diferenciada somente pelos sinais passados como argumento. Para isso, a primitiva `kill` invoca o comando *kill* com o sinal SIGKILL, enquanto que a primitiva `stop` invoca o comando *kill*, para parar e continuar a execução dos processos, respectivamente com os sinais SIGSTOP e SIGCONT.

7.3 Conclusão

O presente capítulo abordou a implementação do modelo do mecanismo de resposta automático descrito no capítulo anterior. O objetivo principal da implementação foi demonstrar a exeqüibilidade do modelo. Para isso, a implementação abrangeu o componente de controle/comunicação, representado por um conjunto de agentes móveis, e parcialmente o componente local de reação, que abrangeu algumas das ações de contenção propostas. Esse protótipo capacitou a observação na prática da atuação do mecanismo sobre diferentes atividades consideradas como suspeitas.

Capítulo 8

Conclusão e Trabalhos Futuros

O presente trabalho desenvolveu um mecanismo automático de resposta para um sistema de segurança baseado no sistema imunológico humano. A motivação principal dessa inspiração é que o sistema imunológico humano apresenta um bom modelo de defesa, pois é capaz de garantir a sobrevivência de um ser humano por cerca de 70 anos, mesmo diante de bactérias e vírus potencialmente mortais. Além disso, esse modelo de defesa apresenta algumas características interessantes para um sistema de detecção e resposta, como adaptatividade e tolerância a falhas.

Diversos trabalhos que relacionam o sistema imunológico humano à segurança de computadores já foram desenvolvidos. Entretanto, tais trabalhos adotam o sistema imunológico humano apenas como um modelo para o desenvolvimento de sistemas de detecção de anomalia. O sistema de segurança desenvolvido, que engloba o mecanismo de resposta, é inovador, pois extrai do sistema imunológico humano características tanto de anomalia como de mau uso. Adicionalmente, também foi assimilada a capacidade de aprendizado e adaptação. Como resultado, foi modelado um sistema de detecção e resposta híbrido capaz de responder automaticamente a ataques, podendo também elaborar assinaturas e respostas específicas para um ataque previamente não encontrado. Nesse modelo, as detecções de anomalia e mau uso operam concorrentemente, tornando a detecção mais perspicaz.

O presente trabalho abordou os componentes de resposta do sistema de segurança, desenvolvendo um mecanismo que executa automaticamente as respostas primária e secundária, modeladas de acordo com as respostas primária e secundária do sistema imunológico. Seu principal objetivo foi demonstrar a viabilidade de um sistema de resposta automático. Para isso, ao contrário da maioria dos trabalhos da área que respondem estaticamente a todas as intrusões, foi proposto um modelo de resposta adaptativo e distribuído.

A seguir estão descritos sucintamente os pontos mais relevantes da pesquisa.

- A resposta primária é disparada pelo sistema de detecção de anomalia e é capaz de limitar as extensões de um ataque, ainda que de modo não específico, pois é baseada no possível risco que um ataque oferece ao sistema. Sendo assim, o sucesso de uma resposta depende diretamente da configuração da política de resposta, onde está definido cada nível.

- A resposta secundária é ativada pelo sistema de detecção de mau uso e, portanto, é capaz de responder especificamente ao ataque detectado. Nesse caso, o mecanismo obtém a estratégia de contenção na base de dados a partir da assinatura do ataque. O sucesso desta resposta depende das estratégias associadas às assinaturas.

- As respostas providas, independentemente de seu tipo, são distribuídas. A tecnologia de agentes móveis foi empregada para viabilizar tal característica. Por ser distribuída, a resposta capacita que cada nó da rede reaja ao ataque detectado.

- Além de distribuída, a resposta provida também é adaptável, pois regula dinamicamente sua intensidade de acordo com estímulos recebidos. Essa característica é fundamental, pois permite diminuir o impacto de uma resposta no sistema quando as restrições impostas prejudicam usuários ou processos legítimos. A adaptabilidade é encontrada no nível de confiança, que procura avaliar o impacto de uma ação no sistema, e em algumas ações da resposta primária.

- A implementação realizada neste trabalho representa parcialmente o modelo proposto. Entretanto, apesar dessa restrição, pôde-se constatar que o modelo é factível de implementação e aplicável na prática. As ações de contenção implementadas permitiram a verificação de como as atividades de uma intrusão podem ser contidas, bem como a implicação que uma resposta causa no restante do sistema.

8.1 Trabalhos futuros

Essa seção abrange alguns tópicos de extensão ao trabalho descrito. De um modo geral, as extensões citadas sugerem funcionalidades adicionais ao mecanismo de resposta.

8.1.1 Recuperação

Outra funcionalidade que pode ser agregada ao mecanismo de resposta é o restabelecimento do sistema, que tem como objetivo principal recuperar, quando possível, os danos causados pelo ataque. As ações de recuperação são importantes principalmente em ataques que comprometem a integridade do sistema. Um exemplo comum desse tipo de

atividade intrusa é encontrado em ataques que procuram viabilizar um meio de acesso um meio de acesso no sistema atacado através da instalação de *backdoors*[73].

Entretanto, o restabelecimento de um sistema computacional após uma intrusão é uma tarefa complexa. A complexidade é justificada pelo fato de um sistema computacional manipular e depender de um grande número de informações que podem ser modificadas quando um ataque obtém o controle do sistema. Essa complexidade ainda é ampliada pelo fato do mecanismo de resposta do sistema de segurança estar projetado como um mecanismo de resposta automático, onde as respostas são formuladas e executadas rapidamente sem a intervenção do administrador.

Em geral, um esquema de recuperação envolve a identificação das modificações efetuadas no sistema e a recuperação das alterações. A identificação dos arquivos alterados pode ser implementada, por exemplo, através de um verificador de integridade[57], que varre o sistema de arquivos a procura de modificações.

Outra funcionalidade que pode ser agregada às ações de restauração é a correção da vulnerabilidade explorada pelo ataque. Essa não é uma tarefa simples de ser implementada, pois geralmente envolve a instalação de *patches* ou de novas versões de programas. Entretanto, é possível utilizar um analisador de vulnerabilidades[10, 14] para, ao menos, tentar encontrar a vulnerabilidade explorada.

8.1.2 Auto defesa

Como relatado na seção 6.8, um assunto pouco considerado na área é o fato do próprio mecanismo de resposta ser alvo de um atacante. Nesse caso, o mecanismo é manipulado de modo a lançar ações de contenção para prejudicar usuários e processos legítimos.

Observando isso, o mecanismo de resposta pode apresentar uma estratégia de auto defesa que procura por reações prejudiciais à segurança do sistema. Essa estratégia pode ser modelada através de um método simples de detecção de ações de contenção nocivas, desempenhada por um agente que analisa os registros das atividades dos agentes do mecanismo de resposta. Quando alguma atividade nociva é encontrada, o agente notifica os responsáveis pelas atividades para finalizá-las.

8.1.3 Nível de confiança para as ações do nível de processo

O cálculo do nível de confiança, além de contribuir para as ações do nível de rede, também pode ser estendido para as ações do nível de processos. Nesse caso, pode ser adotado um método de avaliação que compara o comportamento atual do processo alvo da resposta com registros do comportamento de outras instâncias normais da aplicação. Alguns parâmetros de comparação podem ser: consumo médio do processador, consumo de memória e número de conexões estabelecidas. Quanto mais distante o comportamento

atual do processo estiver dos registros, menor o seu nível de confiança. Essa relação supõe que um processo sob ataque geralmente apresenta comportamento diferente do normal.

8.1.4 Rastreamento

Existe ainda uma terceira funcionalidade, além da contenção e restauração, que pode ser incorporada em mecanismos de resposta. Essa funcionalidade é a de rastrear a origem do ataque, quando o mesmo for originado remotamente. Essa tarefa, muitas vezes, não é simples de ser desempenhada[91, 101]. Dentre as dificuldades encontradas para seu desenvolvimento, duas são de maior destaque:

1. o atacante pode ter falsificado seu endereço de origem;
2. a origem do ataque pode estar escondida através de vários nós de rede intermediários.

A principal motivação para esse funcionalidade é que torna o mecanismo capaz de atribuir a responsabilidade sobre os eventos ocorridos.

8.1.5 Geração de respostas

Outro possível trabalho futuro é o desenvolvimento do componente *response generator* do sistema de segurança imunológico. Esse componente é responsável pela geração automática de respostas secundárias, com base na assinatura do ataque. Um possível esquema de geração consiste em relacionar um processo de resposta primária com a assinatura fornecida e então convertê-lo em um processo de resposta secundária. Entretanto, deve ser considerado que as respostas primárias contém ações adaptativas e, por isso, reagem de acordo com o estado específico do sistema. Essa dependência do estado específico do sistema pode implicar em uma estratégia de resposta de secundária inadequada.

8.1.6 QoS

Algumas ações de contenção propostas tem como objetivo restringir parcialmente a utilização dos recursos computacionais disponibilizados pela rede e pelo sistema operacional. Tais ações, além de serem aplicadas à detecção de intrusão e resposta, podem ser exploradas para o desenvolvimento de um mecanismo de controle de recursos. Esse mecanismo tem como objetivo distribuir os recursos computacionais entre os processos do sistema, provendo assim uma generalização para as pesquisas em *QoS* que é restrita aos recursos de rede. Exemplos de recursos passíveis de controle são: processador, memória, dispositivos de armazenamento e a banda passante da rede.

Referências Bibliográficas

[1] T. Aivazian. Linux 2.4 internals, Agosto de 2002. Disponível *online* em setembro de 2002 na URL http://kernel.pe.kr/data/doc/lki/lki.html.

[2] Debra Anderson, Thane Frivold, Ann Tamaru, e Alfonso Valdes. Next-generation intrusion detection expert system (nides), software users manual, beta-update release. Relatório Técnico SRI–CSL–95–07, Computer Science Laboratory, SRI International, maio de 1994. Disponível *online* em agosto de 2002 na URL citeseer.nj.nec.com/anderson94next.html.

[3] James P. Anderson. Computer security technology planning study vol i/ii. Relatório técnico, US Air Force, Eletronic Systems Division, Hanscom Field, Bedford, 1972. Disponível em setembro de 2002 na URL http://seclab.cs.ucdavis.edu/projects/history/papers/ande72.pdf.

[4] James P. Anderson. Computer security threat monitoring and surveillance. Relatório técnico, James P. Anderson Co., Fort Washington, Pennsylvania, 1980. Disponível em setembro de 2002 na URL http://seclab.cs.ucdavis.edu/projects/history/papers/ande80.pdf.

[5] Ross Anderson e Abida Khattak. The use of information retrieval techniques for intrusion detection. Em *First Workshop on the Recent Advances in Intrusion Detection*, Louvain-la-Neuve, Belgium, Setembro de 1998.

[6] K. Arnold, J. Gosling, e D. Holmes. *The Java Programming Language*. Addison-Wesley, Reading, Massachusetts, 3ª edição, 2000.

[7] M. Asaka, S. Okazawa, A. Taguchi, e S. Goto. A method of tracing intruders by use of mobile agents. Em *INET'99*, 1999. Disponível *online* em janeiro de 2002 na URL citeseer.nj.nec.com/asaka99method.html.

[8] Stefan Axelsson. Research in intrusion detection systems: A survey. Relatório técnico, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, Dezembro de 1998.

[9] Rebecca Bace. An introduction to intrusion detection assessment. Disponível *online* em agosto de 2001 na URL http://www.iss.net/prod/whitepapers, 1999. Infidel, Inc.

[10] Rebecca Bace e Peter Mell. Intrusion detection systems. NIST Special Publication on Intrusion Detection System.

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

[12] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, Eugene H. Spafford, e Diego Zamboni. An architecture for intrusion detection using autonomous agents. Em *ACSAC*, pp. 13–24, 1998.

[13] J. Balthrop, S. Forrest, e M. Glickman. Revisiting lisys: Parameters and normal behavior. Em *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.

[14] M. Bishop. Vulnerabilities analysis. Em *Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection (RAID'99)*, 1999.

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

[16] Curtis Carver, John Hill, John Surdu, e Udo Pooch. A methodology for using intelligent agents to provide automated intrusion response. Em *Proceedings of the 2000 IEEE Workshop on Information Assurance and Security*, West Point, NY, Junho de 2000. United States Military Academy.

[17] Fred Cohen. Computer viruses. *Computers & Security*, 6:22–35, 1987.

[18] Mark Crosbie, Bryn Dole, Todd Ellis, Ivan Krsul, e Eugene Spafford. IDIOT - user guide, Setembro de 1996. Disponível *online* em agosto de 2002 na URL citeseer.nj.nec.com/crosbie96idiot.html.

[19] Dipankar Dasgupta. Immunity-based intrusion detection systems: A general framework. Em *Proceedings of the* 22^{nd} *National Information Systems Security Conference (NISSC)*, Outubro de 1999.

[20] Dipankar Dasgupta e Hal Brian. Mobile security agents for network traffic analysis. Em *Proceedings of DARPA Information Survivability Conference and Exposition II (DISCEX II)*, California, Junho de 2001.

[21] Dipankar Dasgupta, Nivedita Majumdar, e Fernando Nino. Artificial immune systems: A bibliography. Relatório técnico, Computer Science Division, University of Memphis, 2002.

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

[23] Diego de Assis Monteiro Fernandes, Fabrício S. de Paula, Marcelo A. Reis, e Paulo Lício de Geus. Adenoids: A hybrid ids based on the immune system. Em *ICONIP'02-SEAL'02-FSKD'02*, Singapore, Novembro de 2002.

[24] Diego de Assis Monteiro Fernandes, Marcelo A. Reis, Fabrício S. Paula, e Paulo L. Geus. A hybrid ids architecture based on the immune system. Em *Anais do Wseg2002: Workshop em Segurança de Sistemas Computacionais*, Búzios, RJ, Maio de 2002. Workshop realizado durante o SBRC2002: Simpósio Brasileiro de Redes de Computadores.

[25] Leandro Nunes de Castro e Fernando José Von Zuben. Artificial immune systems: Part 2 - a survey of applications. Relatório Técnico DCA-RT 02/00, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, SP, Brazil, fevereiro de 2000.

[26] Leandro Nunes de Castro e Fernando Von Zuben. Artificial immune systems: Part 1 - basic theory and applications. Relatório técnico, Universidade Estadual de Campinas, Dezembro de 1999.

[27] D. E. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, Fevereiro de 1987.

[28] Patrik D'haeseleer, S. Forrest, e P. Helman. An immunological approach to change detection: Algorithms, analysis, and implications. Em *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*, Los Alamitos, CA, 1996. IEEE Press.

[29] Cotse-security-denial of service. Disponível *online* em dezembro de 2002 na URL http://www.cotse.com/dos.htm.

[30] Marcelo Abdalla dos Reis. Forense computacional e sua aplicação em segurança imunológica. Tese de Mestrado, Universidade Estadual de Campinas, 2003.

[31] Bruce Eckel. *Thinking in Java.* Prentice Hall Professional Technical Reference, 2002.

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

[33] W Finnset, J Grav, G Rogde, J Zoric, e J Aarbø. Intelligent mobile agents - status and four promising application domains. Relatório técnico, Telenor FoU, 1999. FoU N 24/99.

[34] S. Forrest e S. Hofmeyr. Immunology as information processing. Em *Design Principles for Immune Systems and Other Distributed Autonomous Systems.* Oxford University Press., 2000.

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

[36] Stephanie Forrest e Steven A. Hofmeyr. John Holland's invisible hand: An artificial immune system. Presented at the Festschrift held in honor of John Holland, 1999. Disponível *online* em agosto de 2002 na URL citeseer.nj.nec.com/forrest99john.html.

[37] Stephanie Forrest, Steven A. Hofmeyr, e Anil Somayaji. A sense of self for unix processes. Em *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pp. 120–128, Los Alamitos, CA, 1996. IEEE Computer Society Press.

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

[39] Alfonso Fuggetta, Gian Pietro Picco, e Giovanni Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.

[40] Steven M. Furnell, George B. Magklaras, Maria Papadaki, e Paul S. Dowland. A generic taxonomy for intrusion specification and response. Em *Proceedings of Euromedia 2001*, Valencia, Spain, Abril de 2001.

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

[42] Robert S. Gray, George Cybenko, David Kotz, Ronald A. Peterson, e Daniela Rus. D'Agents: Applications and performance of a mobile-agent system. *Software—Practice and Experience*, 32(6):543–573, Maio de 2002.

[43] Object Management Group. Mobile agent system interoperability facilities specification. OMG TC Document orbos/98-03-09.pdf, 1998. Disponível *online* em agosto de 2002 na URL ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf.

[44] G. Helmer, J. Wong, V. Honavar, e L. Miller. Intelligent agents for intrusion detection. Em *IEEE Information Technology Conference*, pp. 121–124, Syracuse, NY, 1998.

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

[46] Steven A. Hofmeyr e Stephanie Forrest. Architecture for an artificial immune system. *Evolutionary Computation*, 7(1):45–68, 1999.

[47] Steven A. Hofmeyr e Stephanie Forrest. Immunity by design: An artificial immune system. Em *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pp. 1289–1296, Orlando, Florida, USA, 13-17 de 1999. Morgan Kaufmann.

[48] Steven A. Hofmeyr, Stephanie Forrest, e Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 1998.

[49] Koral Ilgun, Richard A. Kemmerer, e Phillip A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, Março de 1995.

[50] Sun Microsystems Inc. RPC: Remote procedure call protocol specification, Abril de 1988. RFC 1050.

[51] Iptables web page. Disponível *online* em dezembro de 2002 na URL htpp://www.iptables.org.

[52] W. Jansen, P. Mell, T. Karygiannis, e D. Marks. Mobile agents in intrusion detection and response. Em *12th Annual Canadian Information Technology Security Symposium*, Ottawa, Canada, Junho de 2000.

[53] Wayne Jansen, Peter Mell, Tom Karygiannis, e Don Marks. Applying mobile agents to intrusion detection and response. Relatório técnico, National Institute of Standards and Technology, Computer Security Division, Outubro de 1999. NIST Interim Report - 6416.

[54] J. Kephart, G. Sorkin, e M. Swimmer. An immune system for cyberspace. Em *Proceedings of the IEEE SMC'97*, pp. 879–884, 1997.

[55] J. Kephart, G. B. Sorkin, M. Swimmer, e S. R. White. Blueprint for a computer immune system. Em D. Dasgupta, editor, *Artificial Immune Systems and Their Applications*, pp. 241–261. SpringerVerlag, 1999.

[56] J. O. Kephart. A biologically inspired immune system for computers. Em R. A. Brooks e P. Maes, editores, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Sysnthesis and Simulation of Living Systems*, pp. 130–139, Cambridge, MA, 1994. MIT Press.

[57] G. H. Kim e E. H. Spafford. The design and implementation of tripwire: A file system integrity checker. Em *Proceedings of the* 2^{nd} *ACM Conference on Computer and Communications Security*, 1994.

[58] J. Kim e J. P. Bentley. Towards an artificial immune system for network intrusion detection: An investigation of clonal selection with a negative selection operator. Em *Congress on Evolutionary Computation (CEC-2001)*, Seoul, Korea, Maio de 2001.

[59] J. Kim e P. Bentley. An artificial immune model for network intrusion detection. 7th European Conference on Intelligent Techniques and Soft Computing (EUFIT'99), Aachen, Germany, 1999.

[60] J. Kim e P. Bentley. The human immune system and network intrusion detection. 7th European Conference on Intelligent Techniques and Soft Computing (EUFIT'99), Aachen, Germany, 1999.

[61] J. Kim e P. Bentley. Towards an artificial immune system for network intrusion detection: An investigation of dynamic clonal selection. Em *Congress on Evolutionary Computation (CEC-2002)*, pp. 1015 – 1020, Honolulu, 2002.

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

[63] Danny Lange e Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets.* Addison Wesley, Reading, Massachusetts, 1998.

[64] Danny B. Lange e Mitsuru Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, 1999.

[65] Linda Lankewicz e Mark Benard. Real–Time Anomaly Detection Using A Nonparametric Pattern Recognition Approach. Em *Proceedings of Seventh Annual Computer Security Conference*, dezembro de 1991. 2-6th, San Antonio, TX.

[66] Wenke Lee, Salvatore J. Stolfo, e Kui W. Mok. A data mining framework for building intrusion detection models. Em *IEEE Symposium on Security and Privacy*, pp. 120–132, 1999.

[67] G. Liepins e H. Vaccaro. Intrusion detection: Its role and validation. *Computers & Security*, 11:247–355, 1992.

[68] Ulf Lindqvist e Erland Jonsson. How to systematically classify computer security intrusions. Em *Proceedings of the 1997 IEEE Symposium on Security & Privacy*, pp. 154–163, Oakland, CA, Maio de 1997. IEEE Computer Society Press.

[69] Ludovic Mé e Cédric Michel. Intrusion detection: A bibliography. Relatório Técnico SSIR-2001-01, SUPÉLEC, France, Setembro de 2001.

[70] John McHugh. Intrusion and intrusion detection, Julho de 2001. CERT Coordination Center, Software Engineering Institute, Carneggie Mellon University.

[71] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, e J. White. Masif, the omg mobile agent system interoperability facility. Em *Mobile Agents, Proceedings of the Second International Workshop, MA'98*, pp. 14 – 15, Setembro de 1998.

[72] David Moore, Geoffrey M. Voelker, e Stefan Savage. Inferring internet Denial-of-Service activity. Em *Usenix Security Symposium*, pp. 9–22, 2001. Disponível *online* em dezembro de 2002 na URL http://citeseer.nj.nec.com/moore01inferring.html.

[73] Nelson Murilo e Klaus Steding-Jessen. Métodos para detecção local de rootkits e módulos de kernel maliciosos em sistemas unix. Em *Anais do SSI2001: 3º Simpósio Segurança em Informática*, pp. 133–139, Instituto Tecnológico de Aeronáutica-ITA, São José dos Campos, SP, Outubro de 2001.

[74] P. Neumann e P. Porras. Experience with emerald to date. Em *Proceedings of 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, pp. 73–80, Santa Clara, California, Abril de 1999.

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

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

[77] Michael G. Noblett, Mark M. Pollitt, e Lawrence A. Presley. Recovering and examining computer forensic evidence. *Forensic Science Communications*, 2(4), Outubro de 2000. U.S. Department of Justice, FBI.

[78] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(2):205–244, 1995. Disponível *online* em fevereiro de 2002 na URL citeseer.nj.nec.com/nwana96software.html.

[79] ObjectSpace, Inc. *ObjectSpace Voyager - Core Technology User Guide*, 1997. Version 1.0.0.

[80] P.D. O'Brian e R.C. Nicol. Fipa – towards a standard for software agents. *BT Technology Journal*, 16(3):51 – 59, Julho de 1998.

[81] Maria Papadaki, Steven M. Furnell, Benn Lines, e Paul L. Reynolds. A response-oriented taxonomy of it system intrusions. Em *Proceedings of Euromedia 2002*, pp. 87–95, Modena, Italy, Abril de 2002.

[82] Maria Papadaki, George B. Magklaras, Steven M. Furnell, e Abdulaziz Alayed. Security vulnerabilities and system intrusions - the need for automatic response frameworks. Em *Proceedings of the IFIP 8th Annual Working Conference on Information Security Management & Small Systems Security*, Las Vegas, Setembro de 2001.

[83] Holger Peine e Torsten Stolpmann. The architecture of the Ara platform for mobile agents. Em Radu Popescu-Zeletin e Kurt Rothermel, editores, *First International Workshop on Mobile Agents MA'97*, Berlin, Germany, 1997.

[84] Mats Persson. Mobile agent architectures. Relatório técnico, Defence Research Establishment/Division of Command and Control Warfare Technology - Sweden, Dezembro de 2000.

[85] P. Porras e P. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. Em *Proceedings of the 20th National Information Systems Security Conference*, pp. 353–365, 1997.

[86] Portsentry - psionic technologies. Disponível *online* em janeiro de 2002 na URL http://www.psionic.com/products/portsentry.html.

[87] Daniel J. Ragsdale e et al. Adaptation techniques for intrusion detection and intrusion response systems. Em *IEEE International Conference on Systems, Man, and Cybernetics*, Nashville, TN, Outubro de 2000.

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

[89] D. Rusling. The linux kernel, 1999. Disponível *online* em agosto de 2002 na URL citeseer.nj.nec.com/article/rusling99linux.html.

[90] D. Russell e Sr. G. T. Gangemi. *Computer Security Basics.* O'Reilly & Associates, Inc., 1991.

[91] D. Schnackenberg, K. Djahandari, e D. Sterne. Infrastructure for intrusion detection and response. Em *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*, Hilton Head, South Carolina, Janeiro de 2000.

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

[93] Stephen E. Smaha. Haystack: An intrusion detection system. Em *Fourth Aerospace Computer Security Applications Conference*, pp. 37–44, Austin, TX, Dezembro de 1988. Tracor Applied Science Inc.

[94] Anil Somayaji. *Operating System Stability and Security through Process Homeostasis.* Tese de Doutorado, University of New Mexico, Albuquerque, New Mexico, Julho de 2002.

[95] Anil Somayaji e Stephanie Forrest. Automated response using system-call delays. Em *Proceedings of the* 9^{th} *USENIX Security Symphosium*, Agosto de 2000.

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

[97] James W. Stamos e David K. Gifford. Remote evaluation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(4):537–564, 1990.

[98] Thomas Toth e Christopher Kruegel. Evaluating the impact of automated intrusion response mechanisms. Em *Annual Computer Security Applications Conference (ACSAC)*, USA, Novembro de 2002. IEEE Computer Society Press.

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

[100] Nelson Uto e Ricardo Dahab. Survey sobre segurança de sistemas de agentes móveis. Relatório Técnico IC-01-08, Instituto de Computação - Unicamp, Julho de 2001.

[101] Xinyuan Wang, Douglas S. Reeves, S. Felix Wu, e Jim Yuill. Sleepy watermark tracing: An active network-based intrusion response framework. Em *Proceedings of the 16th International Conference on Information Security: Trusted Information*, pp. 369–384, 2001.

[102] Christina Warrender, Stephanie Forrest, e Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. Em *Proceedings of the 1999 IEEE Sysmposium on Security and Privacy*. IEEE Computer Society Press, Maio de 1999. verificar se foi peblicado.

[103] Gregory B. White, Eric A. Fisch, e Udo W. Pooch. Cooperating security managers: A peer-based intrusion detection system. Em *IEEE Network*, pp. 20–23, Fevereiro de 1996.

[104] David Wong, Noemi Paciorek, Tom Walsh, Joe DiCelie, Mike Young, e Bill Peet. Concordia: An infrastructure for collaborating mobile agents. In Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany, Abril de 1997.

[105] E. D. Zwicky, S. Cooper, e D. B. Chapman. *Building Internet Firewalls*. O'Reilly & Associates, Sebastopol, California, 2ª edição, 2000.