Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# NERD (Network Exfiltration Rootkit Detector): Um Sistema Imune Artificial Multi-Agente para Detecção de Rootkits por Exfiltração em Rede

Mateus Berardo de Souza Terra

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. João José Costa Gondim

Brasília
2021

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# NERD (Network Exfiltration Rootkit Detector): Um Sistema Imune Artificial Multi-Agente para Detecção de Rootkits por Exfiltração em Rede

Mateus Berardo de Souza Terra

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. João José Costa Gondim (Orientador)
CIC/UnB

Prof. Dr. Marcos F. Caetano
Departamento de Ciência da Computação

Prof. Dr. Robson de Oliveira Albuquerque
Departamento de Engenharia Elétrica

Prof. Dr. João José Costa Gondim
Coordenador do Curso de Engenharia da Computação

Brasília, 26 de Outubro de 2021

# Agradecimentos

Agradeço à minha mãe pelo constante apoio e pela disposição de revisar um trabalho tão distante de sua área de atuação.

Agradeço ao meu pai pela empolgação ao ver o projeto produzir resultados e por ceder roteadores para instalação de um sistema operacional alternativo.

Agradeço à minha noiva, Larissa, que me apoiou em todos os momentos que a carga de trabalho parecia ser maior do que eu aguentaria.

Agradeço à Universidade de Brasília pelas oportunidades de crescimento que a graduação me proporcionou, em especial aos professores Guilherme Ramos, Carla Koike, Daniel Café, Maristela de Holanda e João Gondim, que marcaram minha graduação com suas aulas excepcionais.

Agradeço ao meu orientador João Gondim, que teve muita paciência durante nossa busca por um tema, que conseguiu me proporcionar liberdade na medida certa e que sempre me apoiou na busca por soluções quando o sistema não compilava ou não funcionava.

Agradeço ao professor André Ricardo Abed Grégio, do Departamento de Informática da Universidade Federal do Paraná, pelo apoio na busca do espécime de rootkit e pelas sugestões valiosas de análise da ferramenta.

Agradeço também à NovaWeb, na pessoa do Celso Souza, pela flexibilidade que me foi dada, essencial para que o projeto fosse concluído no prazo correto, pelas incontáveis oportunidades de aprendizado e pelo acesso a plataformas de conhecimento, muitas vezes caras.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

# Resumo

Desde sua primeira aparição nos anos 1990 *rootkits* se apresentam como grandes ameaças à computação. Estes *malwares* são capazes de ocultar a presença e a atividade de um agente malicioso e seus associados sendo especialmente difíceis de detectar devido à sua capacidade de subverter o Sistema Operacional. Geralmente, *rootkits* são ferramentas utilizadas em campanhas de ataque cibernético, como APT's, que visam exfiltrar informações confidenciais ou propriedade intelectual, para facilitar a manutenção e a ocultação da presença do adversário. Neste contexto, diversas ferramentas foram propostas na literatura para detecção destes programas por meio de diferentes técnicas. Entre elas a arquitetura MADEX é um Sistema Imune Artificial para análise de fluxo capaz de detectar obfuscação de tráfego. Este trabalho desenvolve a proposta do MADEX com o objetivo de elevar sua tolerância à carga de rede sem comprometer a capacidade de detecção. A arquitetura NERD resultante é capaz de suportar tráfego acima do limite da implementação original do MADEX e com melhor performance de detecção. A acurácia obtida foi de 99,996% e a taxa de falso positivo, abaixo de 0,08%.

**Palavras-chave:** Segurança da Informação, Sistemas Imunes Artificiais, Sistemas Multi-Agente, Análise de Fluxo, Rootkits, Detecção de Malware

# Abstract

Since it's first appearance in the 1990's, rootkits have posed severe threats to computing. These malware are capable of hiding the presence and activity of a malicious agent and it's associates and are especially hard to detect due to it's capacity to subvert the Operating System. Usually rootkits are tools for cyber attacks, like APT campaings, whose objective is to exfiltrate sensitive information or intellectual property, used to maintain and conceal the adversary presence. In this context, several tools were proposed in the literature for the detection of such programs through various techniques. Among them, MADEX is a Artificial Immune System that uses flow analysis to detect traffic obfuscation. This work enhances MADEX so that it could handle more load without impact to its detection capabilites. The resulting NERD architecture is capable to process trafic at higher rates than those of MADEX with a better detection performance. The obtained accuracy was 99.996% and false positive rates were below 0.08%.

**Keywords:** Information Security, Artificial Immune Systems, Multi-Agent Systems, Flow-based Analysis, Rootkits, Malware detection

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Algoritmos

# Lista de Abreviaturas e Siglas

**AA** Agente Auditor.

**AIS** *Artificial Immune System*.

**APC** *Antigen Presenting Cell*.

**API** *Application Programming Interface*.

**APT** Ameaça Persistente Avançada(*Advanced Persistent Threat*).

**CA** Agente Coletor.

**CPU** *Central Processing Unit*.

**DC** Células Dendríticas (*Dendritic Cells*).

**DCA** Algoritmo de Células Dendrítica (*Dendritic Cell Algorithm*).

**DT** Teoria do Perigo (*Danger Theory*).

**HTTP** *Hypertext Transfer Protocol*.

**HTTPS** *Hypertext Transfer Protocol Secure*.

**ICMP** *Internet Control Message Protocol*.

**IDS** *Intrusion Detection System*.

**IP** *Internet Protocol*.

**IS** Sistema Imune (*Immune System*).

**iSS** Sinais Ilícitos Similares (*illicit Similar Signals*).

**LKM** *Loadable Kernel Module*.

**PID** *Process ID*.

**RAM** *Random Access Memory*.

**REST** *Representational State Transfer*.

**SDK** *Software Development Kit*.

**SO** Sistema Operacional.

**TCP** *Transmission Control Protocol*.

**TTL** *Time-To-Live*.

**UDP** *User Datagram Protocol*.

# Capítulo 1

# Introdução

No século XXI, a Internet se consolidou como meio principal de transmissão de dados sensíveis para empresas e governos, contudo, ainda é escassa a regulamentação do seu uso [6]. Além disso, o seu número de usuários cresce exponencialmente, chegando a 82 milhões de novos usuários únicos entre Janeiro de 2019 e Janeiro de 2020 [7]. Este contexto se apresenta como uma oportunidade para o crime cibernético, que pode chegar a custar mais de 10 trilhões de dólares para os negócios até 2025 [8].

Neste cenário, diversas ameaças cibernéticas surgiram e afetam cada vez mais o ambiente. Recentemente foi percebido um aumento no número de campanhas de Ameaça Persistente Avançada(*Advanced Persistent Threat*) (APT), caracterizadas por métodos sofisticados e progresso lento, com objetivos de exfiltração de dados sensíveis e propriedade intelectual [9]. Esse tipo de ameaça, usualmente, depende de um conjunto de *malwares* ou *rootkits* que permitam a manutenção da permanência e se estende durante um longo período de tempo com baixa atividade, características que contribuem para a furtividade destas campanhas.

Os *rootkits*, cuja primeira aparicação registrada foi na década de 1990, são ameaças severas à computação devido a sua evolução e adaptação constante [1]. Estes são *malwares* que se diferenciam por sua capacidade de ocultar sua presença maliciosa e a de outros *malwares* relacionados [1]. Considerando este cénario de APT's e outras ameaças que utilizam *rootkits* para garantir a furtividade de seus ataques, é imprescindível o desenvolvimento de ferramentas para sua detecção.

Embora diversas técnicas de detecção tenham sido propostas academicamente [4, 10, 11] e outras estejam disponíveis na Internet como LKRG [12], Rootkit Hunter [13] e Chkrootkit [14], os resultados obtidos ainda possuem, em especial nas soluções comerciais, altas taxas de falsos positivos e baixa sensibilidade, como evidenciado pelos resultados obtidos por [15]. Neste trabalho, é desenhada uma nova arquitetura baseada no MADEX proposto em [4] para detecção de *rootkits* a partir da análise de fluxo da rede.

## 1.1 Justificativa

A arquitetura MADEX, de Marques *et al.* [4], se apresenta como uma solução multi-agente baseada em sistemas imunes artificiais para análise de fluxo e detecção de *rootkits* por meio de exfiltração de dados. A proposta foi capaz de atingir taxas de erro baixas, 1,21% e alta sensibilidade, 99,97%, contudo, possui baixa capacidade de processamento, o que a torna incompatível com o uso em cenários reais. Com três nós na rede analisada realizando requisições HTTP/HTTPS a cada segundo a arquitetura classificou, incorretamente, todo o tráfego como ilícito devido ao mau funcionamento em função da carga [4].

Neste sentido, este trabalho contribui para o refinamento da arquitetura proposta, alcançando uma performance superior em face de alto fluxo de dados sem comprometimento da capacidade de detecção obtida.

## 1.2 Objetivo

A partir da proposta MADEX de [4], o objetivo deste trabalho é projetar e testar uma arquitetura de detecção de *rootkits* por meio de exfiltração de dados baseada em *Artificial Immune System* (AIS)'s que seja capaz de produzir resultados de detecção acurados com alta sensibilidade e de tolerar elevadas cargas de rede.

### 1.2.1 Objetivos Específicos

Os objetivos específicos deste trabalho são desenvolver um sistema que:

- seja capaz de classificar corretamente tráfego lícito e ilícito em uma rede com 3 nós gerando requisições HTTP a cada segundo;
- seja capaz de atingir acurácia igual ou superior àquela obtida pela arquitetura MADEX;
- seja capaz de atingir taxa de falso positivo inferior àquela obtida pela arquitetura MADEX.

## 1.3 Organização do Documento

Este documento explora o projeto e análise da arquitetura NERD comparada ao MADEX [4]. No Capítulo 2 é apresentada uma revisão dos conceitos relevantes, em especial do Sistema Imune (*Immune System*) (IS) humano, que serve de base para a elaboração dos componentes do sistema. No Capítulo 3 o desenvolvimento da arquitetura NERD é

descrita em detalhes, incluindo os ambientes utilizados para teste e a descrição dos experimentos realizados. No Capítulo 4 é realizada uma análise comparativa dos resultados obtidos neste trabalho e em [4]. Finalmente, o Capítulo 5 apresenta as conclusões obtidas por este projeto.

# Capítulo 2

# Revisão Conceitual

Este capítulo trata dos principais conceitos utilizados e dos trabalhos relacionados.

## 2.1 *Rootkits*

*Rootkits* são um tipo especial de *malware* desenvolvido para ter capacidade de ocultar sua própria atividade e a de outros e de facilitar a manutenção de acesso, desta forma dificultando ainda mais a sua detecção [10, 15]. O termo *rootkit* foi criado baseado no usuário *root* de sistemas Unix, que possui todos os direitos na máquina por ser administrador [1]. Ou seja, o termo se refere a um kit que permite acesso, escalação de privilégios e manutenção de controle do sistema [1].

Desde a aparição de tais programas em 1990, estes foram ameaças constantes à computação devido a sua adaptabilidade e melhoramento constante [1].

Para fornecer suas funcionalidades ao atacante, este *malware* utiliza diferentes técnicas de interceptação e manipulação de chamadas de sistema [1]. Dentre essas técnicas, a técnica utilizada pelo *rootkit* Nuk3Gh0st [16], que será utilizado neste trabalho, é o *hooking* de funções, em que é inserido um pulo incondicional para a função maliciosa como primeira instrução [10]. Geralmente, esta abordagem necessita privilégios de superusuário para ser implementada e é utilizada por *rootkits* de *kernel* [15]. Na Figura 2.1 é possível visualizar um esquema simplificado do funcionamento de *hooks* utilizando a biblioteca Detours, do Windows. No fluxograma representado, uma função realiza uma chamada para uma função alvo, contudo, esta chamada é capturada pela *Detour Function*, que pode realizar diferentes ações e então encaminhar a chamada para o alvo e, depois do retorno desta, pode realizar alterações no resultado para finalmente encaminhar a resposta ao chamador.

*Rootkits* que executam em espaço de *kernel*, como módulos carregáveis ou LKM's, com privilégios de superusuário são conhecidos como *rootkits* de *kernel* [11]. Este tipo de

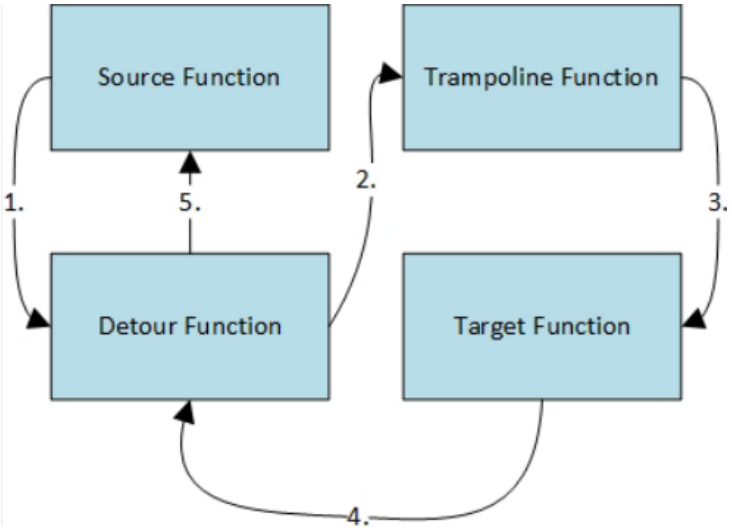



Figura 2.1: Esquema simplificado de *hooking* com a biblioteca Detour **Fonte:** [1]

*malware* é ainda mais difícil de detectar e remover devido ao seu domínio profundo da máquina [10].

Este tipo de ferramenta foi utilizada em mais de um ataque recente para possibilitar a dispersão de *malwares* [11]. Além disso, são frequentemente utilizados em campanhas de Ameaça Persistente Avançada(*Advanced Persistent Threat*) (APT) [17], um tipo de ameaça de longa duração e alta furtividade caracterizada por ter como objetivo a exfiltração de dados confidenciais ou propriedade intelectual [9].

*Rootkits* podem fornecer diferentes utilidades para furtividade e acesso. O *rootkit* Nuk3Gh0st utilizado neste trabalho fornece as seguintes funcionalidades [16]:

1. Fornecer permissões de *root* a um processo do usuário;
2. Esconder um processo da tabela de processos por meio do *Process ID* (PID);
3. Esconder arquivos e diretórios por nome;
4. Esconder portas TCP/UDP abertas para IPv4 ou IPv6;
5. Esconder tráfego TCP por endereço IP;
6. Esconder a presença do módulo de *kernel*;
7. Bloquear a remoção do *rootkit*.

Neste trabalho, a detecção visa identificar o item 5 das capacidades do *rootkit* na rede por meio da comparação do tráfego percebido por dois diferentes agentes em pontos distintos da rede.

### 2.1.1 Detecção

Diferentes técnicas de detecção de *rootkits* foram propostas durante os anos visando combater este tipo de ameaças [11]. Estas podem ser categorizadas como métodos estáticos, que analisam código fonte, ou métodos dinâmicos, que analisam o comportamento do *malware* em execução [11]. Além disso, podem ser divididos em cinco categorias: assinatura, comportamento, *cross-view*, integridade e baseado em hardware [18]. Cada mecanismo possui vantagens e desvantagens e é comum que ferramentas associem diferentes técnicas [15].

Em seu trabalho, Junnila [15] comparou a efetividade de cinco ferramentas *open source* disponíveis: OSSEC, AIDE, Rootkit Hunter, Chkrootkit e LKRG. Nos experimentos propostos, com 75 execuções de detecção, apenas 28 indicações de compromisso foram emitidas por estas ferramentas [15], desta forma, mais de 50% das infecções não puderam ser identificadas. Neste experimento, o *rootkit* Nuk3Gh0st [16], que será avaliado neste trabalho, só foi identificado pela LKRG [15].

A ferramenta proposta inicialmente por Marques *et al.* [4] e modificada neste trabalho, se enquadra como um detector baseado em comportamento, uma vez que utiliza o tráfego de rede em um canal oculto para realizar a detecção da infecção.

## 2.2 Sistemas Imunes Artificiais (AIS)

A natureza é, há bastante tempo, uma inspiração para cientistas da computação que criam algoritmos como Redes Neurais Artificiais, Sistemas Imunes Artificiais e outros baseados nela [2].

Neste trabalho, o interesse reside no Sistema Imune (*Immune System*) (IS), devido especialmente à sua capacidade de detectar um grande número de padrões desconhecidos com recursos limitados e com retenção de memória de forma distribuída [2].

As principais características do sistema imune são [2]:

- unicidade: cada indivíduo possui um sistema único;
- detecção distribuída: não há uma "central de comando" para detecção;
- autorregulação: a interação entre os diversos agentes promove o controle da resposta;
- reconhecimento de padrões aproximados: a detecção é baseada na identificação de pequenos pedaços de um antígeno, reduzindo a complexidade de detecção;
- diversificação: existem inúmeras populações diferentes de células detectoras com diversas mutações;

- detecção de anomalias: padrões desconhecidos são diferenciados de padrões conhecidos;
- protege a si mesmo: sua intenção é defender apenas o corpo, portanto deve ser autotolerante e autodefensivo; e
- memória e aprendizado: respostas passadas podem otimizar respostas futuras baseadas em padrões que foram detectados ao menos uma vez.

Foram propostos diferentes paradigmas do IS, e pelo menos quatro deles foram utilizados na computação, são eles: Seleção Negativa, Seleção Clonal, Rede Imune e Teoria do Perigo (*Danger Theory*) (DT) [2]. Estas teorias se diferenciam pelo entendimento do funcionamento geral do IS proposto. Na seleção negativa e clonal, o problema central do sistema é a diferenciação de si e não-si, isto é, as células que o compõe são especificamente selecionadas de forma a diferenciar o que faz parte do corpo e o que não faz. Já a teoria da Rede Imune busca uma abordagem mais matemática para modelar a interação entre as células que compõem o IS como uma rede complexa de estimulação e inibição [19]. Já a DT compreende que o problema central é identificar sinais de perigo e impedir o dano [20].

### 2.2.1 Seleção Negativa

Algoritmos de seleção negativa assumem que existe um dicionário de tamanho fixo que representa *si* e que deve ser protegido de objetos que não compõe este dicionário [2]. Estes algoritmos possuem um período de treinamento e sensibilização para criar detectores capazes de descrever e diferenciar *si* e *não-si*. Durante este período, detectores que reagem a *si* são eliminados, deste funcionamento foi criado o nome Seleção Negativa.

Na Figura 2.2 é possível visualizar um exemplo no espaço-forma do resultado de um treinamento. Cada detector, representado por um círculo, detecta uma parcela de *si* e compõe um espaço que representa o dicionário. Na Figura, é possível perceber que não há uma combinação exata entre o *si* real e o estimado, o que ocasiona erros.

### 2.2.2 Seleção Clonal

Os algoritmos de seleção clonal compreendem o IS de forma semelhante à seleção negativa, contudo, propõem mecanismos diferentes para seleção e multiplicação de detectores [2]. Neste paradigma são incluídos mecanismos de clonagem, variação e seleção que favorecem o aumento da afinidade de detecção.

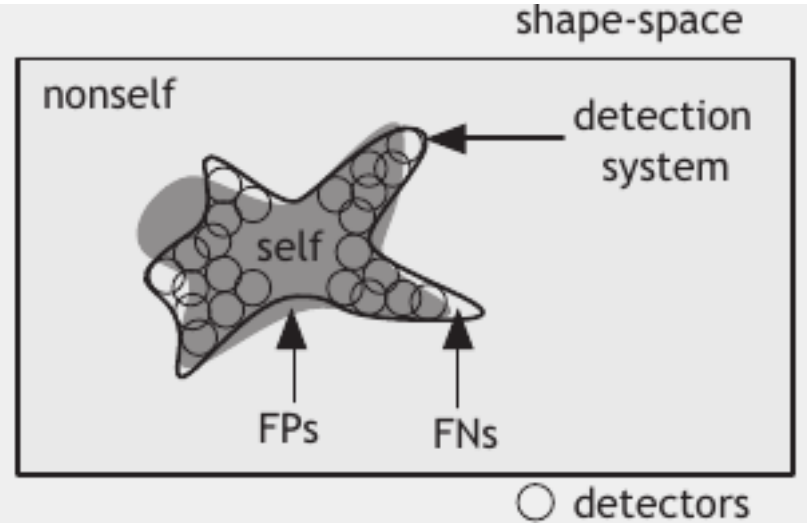



Figura 2.2: Ilustração de espaço-forma com detectores. **Fonte:** [2]

### 2.2.3 Rede Imune

Essa teoria foi inicialmente proposta por Jerne em [21] e revisitada pelo autor no mesmo ano visando desenvolver uma descrição matemática por meio de equações diferenciais do funcionamento do Sistema Imune (*Immune System*) como uma rede [19]. Neste paradigma, o funcionamento do IS é visto como resultado de interações complexas entre diversas células que constantemente estimulam e inibem umas às outras.

### 2.2.4 Teoria do Perigo (*Danger Theory*) (DT)

A Teoria do Perigo (*Danger Theory*) proposta por [20] propõe uma mudança na visão do objetivo do Sistema Imune (*Immune System*). Para o autor, o IS não visa distinguir *si*/*não-si*, mas sim detectar perigo e impedir destruição. Para [20] é possível explicar o motivo pelo qual o corpo tolera partes que não o compõem e por que é possível que ele desenvolva uma resposta a uma parte de *si* a partir dessa nova perspectiva. Em [22], os autores afirmam que esta teoria é essencial para revelar o real potencial de AIS's e permitir a produção de sistemas comercialmente viáveis capazes de lidar com problemas reais.

Para a detecção de perigo, as células passam a necessitar de dois sinais para que se construa uma resposta imune [20]. Estes sinais são antígenos, pedaços de proteínas que compõem um organismo invasor, e citocinas, proteínas marcadoras de perigo sintetizadas por células em sofrimento ou células que detectaram algum sinal de perigo específico [2].

Este será o paradigma adotado neste trabalho e será a partir desta visão que as células que compõe o IS serão descritas. Na Figura 2.3, está representado um esquema que demonstra o desenvolvimento de uma resposta imune em função do tempo de infecção.

Nesta Figura, extraída de [3] é possível também visualizar as principais células do IS. Para este trabalho, nosso foco será no funcionamento adaptativo do IS.

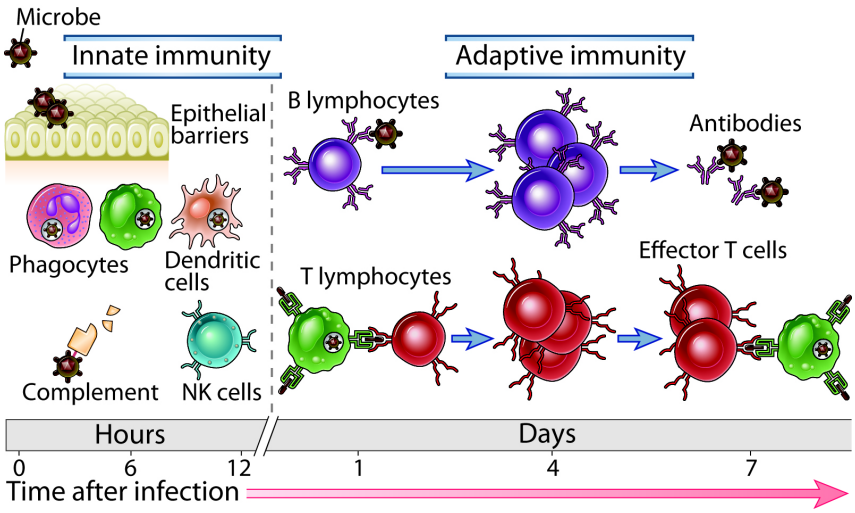



Figura 2.3: Esquema simplificado do desenvolvimento da resposta imune **Fonte:** [3]

### 2.2.5 Células do Sistema Imune

De forma a garantir a objetividade deste trabalho, apenas as células que compõe nossa analogia serão detalhadas. São elas: as Células Dendríticas (*Dendritic Cells*) e os linfócitos B e T.

#### Células Dendríticas (*Dendritic Cells*) (DC)

As Células Dendríticas (*Dendritic Cells*) são células apresentadoras de antígenos(APC's, do inglês *Antigen Presenting Cells*), isto é, são células responsáveis por capturar e processar antígenos, que são pedaços de patógenos [2]. Embora existam outras APC's, as DC's possuem um papel especial devido à sua capacidade singular de ativar linfócitos T virgens, o que lhes rende o título de APC's profissionais [20].

Estas células patrulham o corpo, em especial a pele, a procura de patógenos que elas processam em antígenos que serão apresentados nos linfonodos aos linfócitos, que a partir do reconhecimento deste padrão poderão seguir com a resposta imune adequada [2]. Geralmente estas células são a base de algoritmos baseados na teoria do perigo que utilizam um agente responsável por capturar e processar sinais chamado de célula dendrítica artificial [2].

#### Linfócitos T

Os linfócitos T são células responsáveis pela modulação da resposta imune adaptativa [2]. Estas células possuem receptores capazes de identificar padrões de antígenos e, quando ativadas por meio da ligação com esta molécula, produzem citocinas [2]. Estas substâncias indicam sofrimento celular e recrutam outras células responsáveis por reagir à presença de patógenos como macrófagos, fagócitos, linfócitos T matadores e linfócitos B [2].

Em seu trabalho, Matzinger [20] diferencia linfócitos T experientes e virgens, isto é, células que já tiveram contato com antígenos ou não. Cada uma destas deve ter um caminho de ativação diferente de forma a garantir que o sistema imune será capaz de tolerar *si* [20]. Linfócitos T virgens são ativados quase exclusivamente por DC's, enquanto células experientes podem ser ativadas por outras APC's como linfócitos B e macrófagos.

Os linfócitos T, dependendo da associação entre os dois sinais que os sensibilizam podem estimular ou suprimir uma resposta imune [2]. Caso encontre antígenos, mas não encontre sinais de sofrimento celular, ou seja, citocinas associadas, esta célula irá morrer e suprimir esta resposta imune [20]. Caso os sinais sejam associados, estas células entram em um estado de ativação temporário em que a presença do antígeno isoladamente será suficiente para a produção de uma resposta [20].

#### Linfócitos B

Os linfócitos B possuem um papel importantíssimo no desenvolvimento da memória imune sendo as células responsáveis pela produção de anticorpos, proteínas em formato de Y capazes de identificar antígenos ou patógenos [2]. Além disso, estas células podem atuar como APC's para linfócitos T experientes, contribuindo com o desenvolvimento de uma resposta imune mais rápida após o primeiro contato com um patógeno. Anticorpos são utilizados pelo sistema imune como marcadores. Estas proteínas se ligam a antígenos pelos braços do Y e a macrófagos pela cauda, o que conduz uma resposta muito mais rápida e direcionada [2].

### 2.2.6 Trabalhos Relacionados

A primeira publicação com um algoritmo baseado na Seleção Negativa foi publicado por Forrest *et al.* [23] para detecção de vírus de computadores por meio da análise de arquivos com *strings* de tamanho aleatório. Este trabalho utilizou fases de treinamento em que apenas conteúdo lícito era apresentado para eliminar detectores reativos à *si*.

Diversas analogias de antígenos, com diferentes complexidades de detecção e de geração de detectores foram propostas na literatura [23, 24, 25, 26, 27, 28]. Em [29], os autores

revisaram diversos destes e de outros algoritmos de Seleção Negativa buscando identificar as características fundamentais desta família.

Em [30] os autores se basearam no princípio da seleção clonal para desenvolver o algoritmo CLONALG cujo objetivo é otimização e detecção de padrões, sendo usado para problemas de *Machine Learning* e reconhecimento de padrões. Contudo, não é claro se este algoritmo possui alguma vantagem em relação a algoritmos genéticos [31].

Em seu trabalho, Garrett [31] analisou a utilidade de AIS's segundo critérios de efetividade e unicidade e concluiu que estes algoritmos possuem resultados promissores, por vezes superiores a alternativas em suas áreas. Contudo, o autor levantou o problema da cobertura do espaço-forma pelos algoritmos de Seleção Negativa, que aumenta linearmente com o tamanho do espaço e portanto pode ter limitações de escalabilidade [31].

Em [32], o autor apresenta que o uso de ferramentas dependentes de assinaturas de ameaças não é capaz de acompanhar a evolução dos atacantes, visto que a probabilidade de identificação correta está entre 30 e 50%, o que é um indicativo de que novos métodos baseados em anomalias como AIS's são necessários.

Em [33] foi proposto um framework para implementação de AIS's, ARTIS, focado em aplicações de segurança. Este sistema apresenta diversidade, automonitoramento, computação distribuída, aprendizado, entre outras características desejadas de algoritmos baseados no IS [2]. Este sistema é baseado na Seleção Negativa e apresenta também um mecanismo de co-estimulação baseado em um operador humano. Em um sistema integrado, detectores podem ser compartilhados entre diferentes nós, favorecendo a evolução dos detectores e a proteção de sistemas distribuídos.

Em [22], os autores apresentaram uma análise do uso da DT para a construção de IDS's e concluíram que esta teoria possibilita a otimização do funcionamento destas ferramentas. Em [34], os autores apresentaram o Algoritmo de Células Dendrítica (*Dendritic Cell Algorithm*) (DCA), que foi capaz de obter acurácia acima de 99% com taxas extremamente baixas de falsos positivos. Pesquisas relacionadas a este trabalho resultaram na biblioteca `libtissue` para desenvolvimento baseado no DCA e resultou em um IDS capaz de detectar escaneamento de portas com `nmap`.

Outros trabalhos relacionados podem ser verificados em [2], em que o autor apresenta uma extensa revisão de trabalhos em que AIS's foram utilizados no contexto de segurança.

## 2.3 Sistemas Multi-Agente

O uso de sistemas multi-agentes se destaca devido a sua habilidade de resolução de problemas de forma distribuída [35]. Na literatura, há exemplos de sua associação à AIS's, como em [36], em que foi desenvolvido o RTMAS-AIDS, um AIS multi-agente para de-

tecção adaptativa em tempo real de intrusão. Este trabalho atingiu acurácia de 95,86% e taxa de falso positivo de 2,13%, contudo, necessitaria de outras etapas de treinamento para detecção de ameaças desconhecidas.

Em [37], os autores propuseram o MAIS-IDS, um IDS multi-agente baseado em AIS's de Seleção Clonal capaz de reduzir taxas de falso positivo por meio da comunicação entre diferentes máquinas virtuais da rede protegida.

Um sistema multi-agente define os papéis que cada agente assume e as tarefas realizadas por cada agente, além de determinar o funcionamento da comunicação e cooperação [4]. Estes sistemas podem possuir uma divisão pré-definida do trabalho, ou podem realizar a divisão em tempo de execução [4]. Neste trabalho, com base no MADEX, a divisão será pré-definida.

## 2.4 Arquitetura MADEX

A arquitetura MADEX, proposta por Marques *et al.* [4] serviu de base para o desenvolvimento deste trabalho e tem como objetivo desenvolver um sistema multi-agente de análise de fluxo baseado em AIS's para detecção de *rootkits* com base na exfiltração de dados. Abaixo, é apresentada uma descrição do sistema e na Figura 2.4 está representado o diagrama da arquitetura inicialmente proposta.

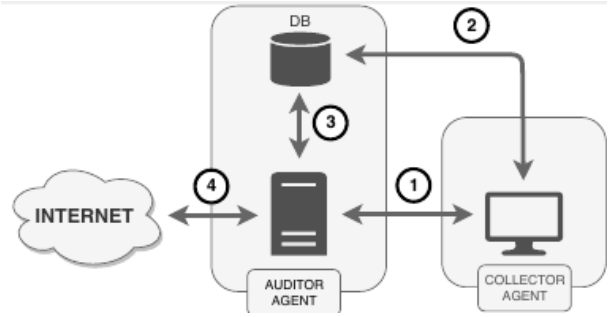



Figura 2.4: Arquitetura MADEX proposta por Marques *et al.* **Fonte:** [4]

Neste sistema o Agente Auditor (AA), ao receber tráfego de rede oriundo da máquina em que o Agente Coletor (CA) está instalado, verifica se a conexão está em uma lista de permissão previamente elaborada e, caso não a encontre, questiona o CA, indicado por 1 na Figura. Neste momento, o CA revisará todas as suas conexões conhecidas via tabela de conexões TCP e montará uma nova lista de permissão para o AA, indicado por 2 na Figura. Em seguida, o AA decidirá entre encaminhar o pacote ou marcá-lo como ilícito, indicado por 4 na Figura, com base na verificação da nova lista, indicado por 3 na Figura. O autor não afirma que os pacotes inicialmente marcados como ilícitos sejam bloqueados antes de que um nó seja identificado como comprometido e submetido a quarentena.

Vale ressaltar que a arquitetura MADEX utiliza a tabela de conexões abertas no host para montagem da lista, isto é, uma ferramenta equivalente ao `netstat` é utilizada. Isto, por um lado, facilita o trabalho do CA, uma vez que basta realizar uma chamada de sistema para listar as conexões conhecidas sem que haja nenhum componente de monitoramento do fluxo de rede propriamente dito, contudo, isto causa o problema de saturação da arquitetura quando o tempo para processamento desta tabela é maior que o *timeout* da chamada realizada pelo AA, conforme verificado por Marques *et al.* [4].

O autor realizou experimentos com esta arquitetura para determinar a carga em que o sistema passava a classificar todo o tráfego como ilícito determinando o limite de operação com três CAs gerando pacotes a cada segundo [4]. O principal objetivo deste trabalho é implementar a arquitetura proposta de forma a elevar a taxa de saturação. Em [4] não existem detalhes de implementação, portanto, toda a definição de arquitetura deste trabalho foi realizada de forma independente.

### 2.4.1 Estratégia de detecção

O sistema MADEX baseia a detecção na premissa de que o *rootkit* irá ocultar seu tráfego por meio da ocultação de portas e conexões abertas nas tabelas de *kernel*. Desta forma, o Agente Coletor verifica sempre que provocado as conexões conhecidas pelo SO, o que não incluirá as conexões que o *rootkit* está ocultando. Já o Agente Auditor irá analisar cada pacote de forma individual, comparando-o às conexões reportadas pelo CA. Uma vez que o *rootkit* não está executando no AA, não será capaz de ocultar o tráfego, desta forma, é possível identificar conexões que estão sendo removidas da tabela de conexões por um agente malicioso no nó do CA.

Esta estratégia é eficiente em detectar *rootkits* que trabalham com a ocultação da atividade de rede por meio da remoção de conexões da tabela de rede, contudo, não será capaz de detectar espécimes que não apresentem esta funcionalidade. Isto é, *rootkits* que permitam que seu canal seja registrado pelo *kernel* utilizando outros métodos como tunelamento de DNS não serão detectados pelo sistema. Esta é uma limitação do método escolhido.

## 2.5 Síntese

Neste capítulo, foi revisado o conceito de *rootkit*, *malwares* com controle profundo do sistema, utilizados para manutenção da presença. Estes serão o foco do sistema de detecção proposto, NERD, um Sistema Imune Artificial multi-agente baseado no MADEX, cujo projeto e desenvolvimento são descritos em detalhes no próximo capítulo.

O sistema proposto utiliza o roteador da rede como um linfonodo, uma região do corpo que abriga diversas células do sistema imune. Neste ambiente, os antígenos coletados por células dendríticas são apresentados aos linfócitos B e T que decidirão sobre o desenvolvimento de uma resposta imune. Na sequência, serão desenvolvidos algoritmos, descritos no próximo capítulo, que representam estas células envolvidas visando alcançar uma arquitetura que apresenta as características do sistema imune humano.

# Capítulo 3

# Metodologia

Neste capítulo é apresentada a arquitetura do sistema, seus componentes e principais algoritmos. Além disso, são discutidos os experimentos realizados com a ferramenta.

## 3.1 Arquitetura NERD

Neste trabalho serão propostas alterações na arquitetura MADEX [4] buscando viabilizar um maior desempenho sob alto volume de tráfego sem que haja degradação do desempenho de detecção. A arquitetura proposta, NERD, segue o esquema descrito na Figura 3.1, em que podemos ver o fluxo de comunicação do sistema.

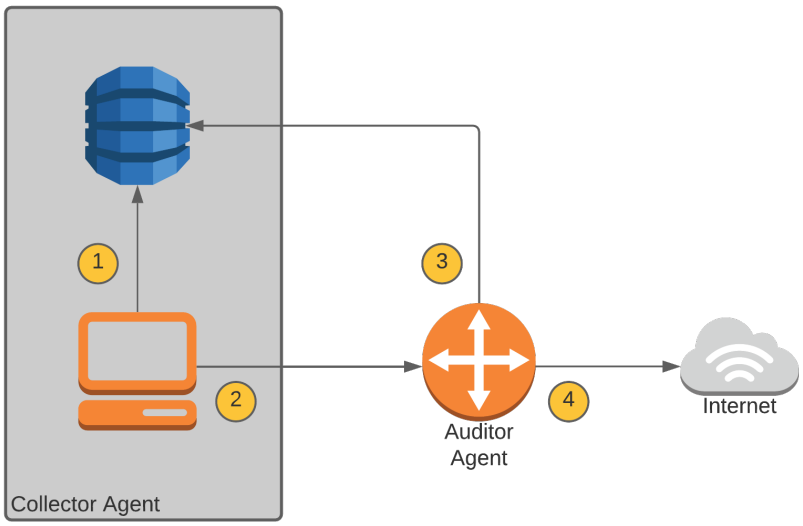



Figura 3.1: Arquitetura proposta

Inicialmente o pacote é percebido pelo CA que atualiza sua lista de conexões para incluir a conexão relativa ao pacote, como indicado pela seta 1 da Figura 3.1. Em seguida,

o pacote é encaminhado ao roteador, onde está executando o AA, como indicado por 2. A captura do CA é não bloqueante e, portanto, o pacote já foi encaminhado ao AA enquanto está sendo inserido no banco, o que influenciará o algoritmo do AA.

A partir dos cabeçalhos IP e do protocolo do pacote, o AA irá consultar o banco de dados das conexões para decidir a ação a ser tomada em relação ao pacote, indicado por 3. Em 4, o pacote é encaminhado para a Internet caso seja encontrado na lista de conexões e bloqueado, caso contrário. Essa característica de reação imediata por meio do impedimento do pacote é uma das vantagens da arquitetura proposta, uma vez que ela não dependerá da quantidade de fluxo ilícito para evitar a exfiltração de dados.

Quando a quantidade de Sinais Ilícitos Similares (*illicit Similar Signals*) (iSS) de um determinado IP da internet superar um limite, ele será banido. Esta ação representa uma ação de memória, desta forma, todos os pacotes para este IP serão bloqueados sem questionamento ao CA quanto ao conhecimento do pacote, constituindo uma resposta rápida.

### 3.1.1 Agente Coletor

Na arquitetura MADEX proposta em [4] e neste trabalho a analogia imune utilizada relaciona o Agente Coletor (CA) com as Células Dendríticas (*Dendritic Cells*) (DC). No contexto da Teoria do Perigo (*Danger Theory*) (DT) [20] estas células patrulham os tecidos e buscam por sinais de perigo emitidos por células sob ataque ou danificadas e criam uma zona de perigo ao redor deste sinal [2]. Em seguida, as DCs coletam antígenos e os apresentam para linfócitos B e T que produzem a resposta imune caso identifiquem os antígenos associados aos sinais de perigo como uma ameaça ao corpo [2].

Em [4], contudo, o CA somente apresenta os antígenos coletados quando questionado sobre um pacote identificado pelo AA, logo, ele é passivo, ao contrário das DC no sistema imune. Uma das hipóteses deste trabalho é que essa decisão arquitetural cria um gargalo no desempenho do sistema, uma vez que o CA precisa percorrer todas as conexões abertas ao buscar a conexão questionada.

Na Figura 3.2 é possível visualizar o esquema de funcionamento do CA. Pacotes gerados pelo usuário ou pelo SO serão encaminhados ao gateway e capturados pela biblioteca `libpcap` de forma que o CA será capaz de inspecionar este tráfego. Ao contrário destes, pacotes gerados pelo *rootkit* utilizado não serão capturados, evadindo portanto o monitoramento do CA e sendo encaminhados ao gateway sem registro no banco de dados.

Visando reduzir a latência e a necessidade de percorrer a lista completa de conexões, o CA irá realizar a captura contínua dos pacotes produzidos pelo *host* utilizando a biblioteca `libpcap` e a cada novo pacote irá atualizar a lista de permissão com a nova conexão. Para registro das conexões foi criada uma 5-tupla, idêntica àquela utilizada em [4], composta

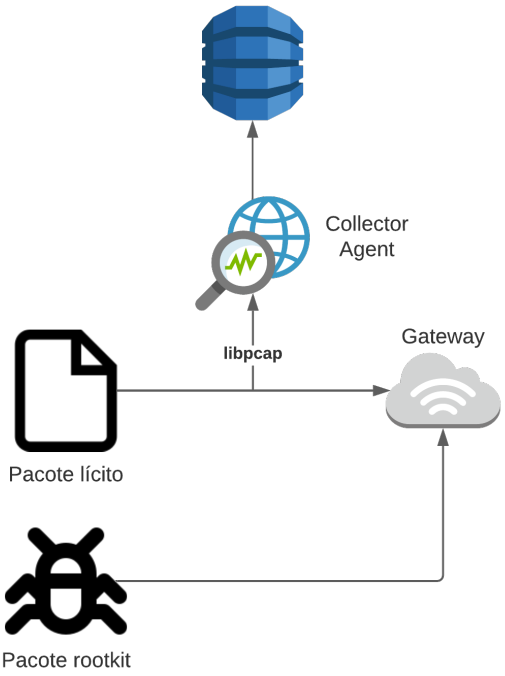



Figura 3.2: Esquema simplificado de funcionamento do Agente Coletor

por IP de origem e destino, porta de origem e destino e protocolo para análise de fluxo. Não é realizada nenhuma análise do conteúdo do pacote, o que favorece o desempenho do sistema e a privacidade de seus usuários.

A partir dessa 5-tupla é gerada uma chave única que identifica este fluxo de forma bidirecional e o registro da conexão é inserido com *Time-To-Live* (TTL) de 60(sessenta) segundos. A cada novo pacote no fluxo, o TTL é reiniciado. Desta forma, o sistema é tolerante a um certo estímulo durante um certo período de tempo. Esta validade garante que, mesmo que o tráfego malicioso tenha sido inicialmente percebido pelo CA, a partir do momento em que ele for ocultado, será detectado como ilícito após o TTL desta chave.

Para montar a chave de identificação da conexão, os endereços IP de origem e destino serão comparados e o menor será selecionado para iniciar a chave, esta estratégia será utilizada para garantir que a direção do fluxo não interfere na chave gerada. Para os protocolos TCP e UDP, as portas de origem e destino acompanham os endereços na formação da chave. A chave resultante é uma string com o formato abaixo.

$$"<ip_1><porta_1><ip_2><porta_2><protocolo>"$$

Para protocolos em que não há uso de portas, como o ICMP, as portas serão zeradas. Na formação da chave, os IP serão utilizados em seu formato decimal de forma a reduzir o tamanho das chaves e o esforço computacional empregado na sua definição.

Cada CA executa um cluster `etcd` de nó único na porta 4001, acessível ao AA, em que serão armazenadas as conexões conhecidas do *host*. Foi selecionado este banco de dados devido a sua alta garantia de consistência, logo, quando o AA verificar a existência da conexão, podemos garantir que, se a requisição de inserção já foi realizada, a resposta correta será enviada.

Uma vez que a captura de pacotes é realizada por meio da captura não bloqueante, a depender do fluxo de pacotes sendo gerados no CA é possível que os pacotes sejam percebidos pelo AA antes que tenha sido solicitada a sua inserção no banco, o que pode causar falsos positivos. Como mitigação deste problema, o processamento dos pacotes é realizado por três threads no CA e o AA implementa funcionalidades de repetição de consultas com espera. Para minimizar o impacto destas múltiplas tentativas, foram utilizados tempos curtos com um número moderado de repetições.

**Algoritmo**

**Algoritmo 1** Algoritmo de processamento de pacotes do CA

**Recebe:** *pacote*
**Ensure:** $ttl > 0$
  $chave.protocolo \leftarrow pacote.protocolo$
  **se** $pacote.ip\_origem < pacote.ip\_destino$ **então**
    $chave.ip_1 \leftarrow pacote.ip\_origem$
    $chave.porta_1 \leftarrow pacote.porta\_origem$
    $chave.ip_2 \leftarrow pacote.ip\_destino$
    $chave.porta_2 \leftarrow pacote.porta\_destino$
  **senão**
    $chave.ip_1 \leftarrow pacote.ip\_destino$
    $chave.porta_1 \leftarrow pacote.porta\_destino$
    $chave.ip_2 \leftarrow pacote.ip\_origem$
    $chave.porta_2 \leftarrow pacote.porta\_origem$
  **fim se**
  $registrar(chave, ttl)$

O algoritmo executado pelo Agente Coletor para cada pacote recebido pode ser verificado no algoritmo 1. Cada pacote capturado de forma assíncrona é enviado para uma fila para que o pacote seja tratado em uma *thread* separada. É possível configurar a quantidade de instâncias de forma a limitar o consumo de recursos do sistema.

Este programa foi implementado em Nim utilizando a biblioteca `libpcap` em C/C++ para captura de pacotes. Para compatibilizar a biblioteca com Nim, foi utilizado o programa `c2nim`, que converte código C para a linguagem. Esta escolha se deve a sintaxe simples e velocidade de execução da ferramenta, uma vez que, quando compilada, produz

código equivalente ao produzido por um programa em C/C++, podendo inclusive ser compilado para estas linguagens [38].

### 3.1.2 Agente Auditor

Na arquitetura proposta neste trabalho, como evidenciado na Figura 3.1, e baseado na arquitetura do sistema MADEX proposta em [4], o Agente Auditor está posicionado na borda da rede, a partir de onde poderá visualizar todo o tráfego de entrada e saída. Na analogia imune proposta, este será um linfonodo, povoado por linfócitos B, responsáveis pela resposta de memória, e linfócitos T, que irão tomar decisões sobre os pacotes em geral.

Considerando que este componente concentra o tráfego, este será também o gargalo de desempenho da rede como um todo. Ademais, roteadores são máquinas com menos recursos de processamento que servidores e especializados em encaminhar pacotes, portanto, é necessário limitar a aplicação a ser implantada neste componente. Outrossim, é importante ressaltar que a cópia de pacotes para fora da malha de encaminhamento e para a memória para processamento é um processo lento em comparação com outras atividades destas máquinas.

Neste trabalho, foi desenvolvido um componente em C para o Sistema Operacional OpenWrt [39], baseado no Linux. Foi utilizada a biblioteca `libnetfilterqueue` para enfileirar pacotes para processamento em espaço de usuário devido à necessidade de utilização da biblioteca `libcurl` para comunicação entre agentes. O AA comunica-se com o banco de dados `etcd` do CA via uma API REST.

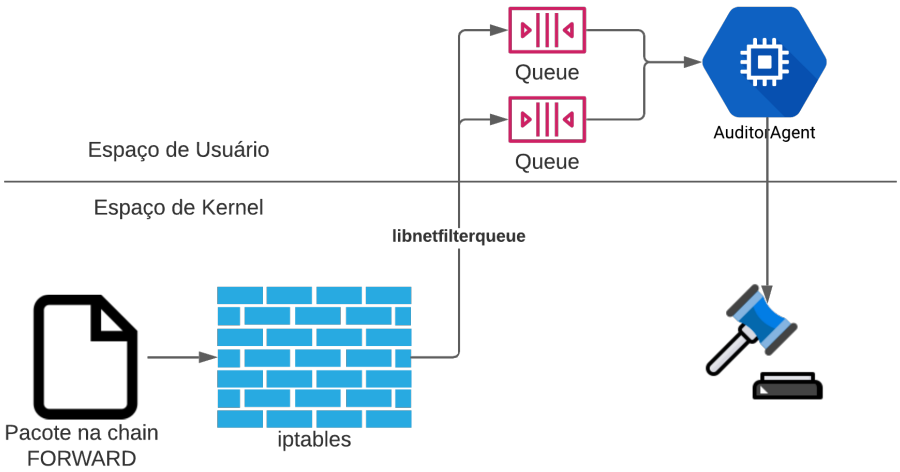



Figura 3.3: Esquema simplificado de funcionamento do Agente Auditor

Na Figura 3.3 é possível visualizar um esquema simplificado do funcionamento do Agente Auditor. Todo pacote identificado na cadeia `FORWARD` pelo `iptables` será enviado pela biblioteca `libnetfilterqueue` do kernel para uma fila de processamento em espaço de usuário que é consumida pelo AA. Para cada pacote, o AA irá emitir um veredito de `ACCEPT`, caso o pacote seja lícito e `DROP`, caso o pacote seja ilícito.

A cópia de pacotes para espaço de usuário é uma tarefa que acrescenta latência ao processamento, portanto, o pacote será copiado parcialmente, incluindo apenas o tamanho equivalente aos cabeçalhos TCP/IP associados, cujo tamanho máximo é de 120 bytes. Com o pacote em espaço de usuário, a 5-tupla utilizada será computada e a interface de entrada do pacote será extraída para que seja possível identificar o endereço do CA que originou o pacote.

A partir dessas informações, o AA gera a chave e envia uma solicitação ao banco `etcd` do CA responsável pelo pacote para identificar o tráfego como lícito ou ilícito. Caso não exista uma entrada com a chave de conexão do pacote será emitido o veredito de rejeição após a expiração das tentativas para o `iptables`, *firewall* utilizado neste sistema Linux.

Para selecionar os pacotes que devem ser inseridos na fila foi criada uma regra no `iptables` para enfileirar o tráfego na cadeia de `FORWARD`, desta forma não são capturadas as consultas aos CA's.

### Algoritmo

O algoritmo executado pelo Agente Auditor para cada pacote recebido pode ser verificado no algoritmo 2. Cada pacote recebido e interceptado pelo `iptables` é enviado para uma fila para que o pacote seja tratado em uma *thread* separada. É possível configurar a quantidade de instâncias de forma a limitar o consumo de recursos do sistema. Este componente foi implementado em C e compilado para a arquitetura do roteador utilizando o SDK do OpenWrt para o R8000.

Inicialmente, o algoritmo calcula a chave da conexão de forma análoga àquela utilizada pelo CA e descrita no algoritmo 1. Em seguida, será iniciado um laço de repetição de processamento do pacote. Primeiramente, é verificada a resposta de memória, isto é, se o endereço IP do servidor remoto está presente na lista de endereços banidos do CA. Esta etapa representa a ação rápida dos linfócitos B.

Em seguida, o Agente Coletor é questionado quanto à chave de conexão, equivalente à atuação dos linfócitos T. Caso conhecida a conexão, o pacote será encaminhado e o processamento encerra. Caso contrário, o programa irá aguardar um tempo de espera pré-determinado e irá realizar uma nova tentativa enquanto o número de tentativas estiver abaixo de um limite configurado. Esta estratégia é necessária, pois o CA possui atraso no registro das conexões em relação ao AA.

**Algoritmo 2** Algoritmo de processamento de pacotes do AA

**Recebe:** $pacote, limiteTentativas, tempoEspera, limiteBloqueios, pacotes, tamanhoCiclo$

```
pacotes ← pacotes + 1
chave.protocolo ← pacote.protocolo
se pacote.ip_origem < pacote.ip_destino então
  chave.ip_1 ← pacote.ip_origem
  chave.porta_1 ← pacote.porta_origem
  chave.ip_2 ← pacote.ip_destino
  chave.porta_2 ← pacote.porta_destino
senão
  chave.ip_1 ← pacote.ip_destino
  chave.porta_1 ← pacote.porta_destino
  chave.ip_2 ← pacote.ip_origem
  chave.porta_2 ← pacote.porta_origem
fim se
tentativas ← 0
enquanto tentativas < limiteTentativas faça
  banido ← questionarCA(pacote.ip_remoto)
  se banido então
    rejeitar(pacote)
  fim se
  conhecido ← questionarCA(chave)
  se conhecido então
    encaminhar(pacote)
  senão
    tentativas ← tentativas + 1
    esperar(tempoEspera)
  fim se
fim enquanto
bloqueios[pacote.ip_remoto] ← bloqueios[pacote.ip_remoto] + 1
se bloqueios[pacote.ip_remoto] > limiteBloqueios então
  banir(pacote.ip_remoto)
fim se
se pacotes >= tamanhoCiclo então
  pacotes ← 0
  limpar(bloqueios)
fim se
```

Caso se esgotem as tentativas e o pacote não tenha sido marcado como conhecido no banco de dados, o AA irá incrementar o contador de Sinais Ilícitos Similares (*illicit Similar Signals*) (iSS) do endereço IP remoto. Caso o iSS seja maior que o limite definido, este IP será banido, isto é, será gerada resposta de memória.

Após processar o pacote, o AA verifica se o ciclo de processamento foi encerrado e redefine os contadores iSS caso tenha alcaçado o número de pacotes por ciclo.

### 3.1.3 Banco de conexões

Para o banco de conexões, foi escolhido o banco de dados `etcd`. Esta escolha foi baseada na sua capacidade de clusterização, que poderia ser aproveitada na arquitetura. Além disso, este banco oferece garantias de consistência fortes em ambientes concorrentes, o que seria essencial para a utilização de um banco clusterizado.

Ademais, a facilidade de comunicação para inserção e consulta das chaves textuais das conexões via API REST foram um fator determinante na escolha deste banco de dados.

### 3.1.4 Estratégia de detecção

O sistema NERD baseia a detecção na premissa de que o *rootkit* irá ocultar o tráfego por meio da evasão de captura, o que é diferente da proposta do MADEX [4], que apenas verifica a tabela de conexões. Esta capacidade é mais complexa e está geralmente associada a *rootkits* mais sofisticados. Desta forma, o Agente Coletor do sistema proposto apresenta um comportamento ativo, em contraste ao comportamento passivo do CA no MADEX [4]. Já os AA's atuam de maneira similar, comparando a lista de conexões geradas pelos CA's com o pacote sendo analisado. Para o sistema NERD, pacotes que não puderam ser capturados pelo CA devido à atuação do *rootkit* e que serão percebidos pelo AA serão classificados como ilícitos.

Esta estratégia está limitada a detectar *rootkits* que possuem e utilizam ativamente sua capacidade de impedir a captura dos pacotes gerados por eles, contudo não é capaz de detectar outros tipos de atividade maliciosa. Devido a esta limitação, *rootkits* que utilizem canais não ocultos de comunicação aparentarão gerar pacotes lícitos para este sistema.

## 3.2 Hardware para Testes

Nesta Seção são descritos os equipamentos utilizados para realizar os testes propostos.

### 3.2.1 Roteador para o Agente Auditor

O agente auditor proposto é executado em um roteador Netgear R8000 v1, cujas características de hardware, extraídas de [5] podem ser verificadas na Tabela 3.1.

### 3.2.2 Vítima

Para os testes foram utilizadas até três máquinas virtuais Linux. O supervisor utilizado foi Oracle VM VirtualBox na versão 6.1.22.

A distribuição escolhida foi o Ubuntu 14.04 LTS devido a seu uso difundido tanto em computadores de uso pessoal quanto em servidores. A versão do kernel durante a execução do trabalho era `4.4.0-148-generic`.

Além do sistema operacional, já estava instalado o wireshark que será utilizado na análise. As configurações destas máquinas são idênticas e podem ser verificadas na Tabela 3.2.

## 3.3 *Rootkit*

Para os testes da ferramenta foi utilizado o *rootkit* Nuk3Gh0st [16] disponível no github como código aberto. Este espécime foi escolhido após diversas buscas devido a apresentar a funcionalidade de ocultação de tráfego baseado nos endereços IP envolvidos. Para ativar a funcionalidade de ocultação do tráfego de rede, foram necessárias alterações em parte do código fonte, que estava comentado no repositório. Para os testes, esta capacidade era essencial, uma vez que esta é a premissa de detecção do sistema.

Este espécime na comparação proposta por [15], em que a eficácia de diferentes ferramentas de detecção de *rootkits* foi comparada, só foi detectado por uma das diversas ferramentas testadas, enganando outras como `chkrootkit` e `rkhunter`.

## 3.4 Experimentos

### 3.4.1 Capacidade da Rede

Para referência da sobrecarga de rede do sistema, serão realizados cinco testes de velocidade da rede sem nenhum dos agentes em uma rede de alto desempenho e a média será utilizada como valor base da capacidade desta rede. Este teste será repetido em uma

| | |
|---|---|
| ***Switch*** | Broadcom BCM4709A0 |
| **Arquitetura** | ARM cortex A9 |
| **Núcleos** | 2 |
| ***Clock*** | 1GHz |
| **RAM** | 256MB |
| ***Flash*** | 128MB |

Tabela 3.1: Configurações de hardware do roteador R8000. **Fonte:** [5]

rede com baixa capacidade três vezes. Para realização do teste será utilizado o *Speedtest by Ookla*, que será a ferramenta de geração de carga para testes de estresse do sistema. Será medido também quantos pacotes são gerados e percebidos, em média, na rede de alto desempenho pelo teste utilizando a ferramenta `wireshark`. Este teste tem como objetivo determinar a capacidade das redes analisadas. Para as duas redes serão executados três testes com o sistema rodando para verificar o impacto do sistema na performance de ambas.

### 3.4.2 Carga do Agente Coletor (CA)

Para determinar a carga exercida no nó pelo Agente Coletor, este componente será executado na vítima descrita na Seção 3.2.2 e o uso de CPU será medido utilizando o utilitário `mpstat`. O uso de CPU será medido sem a presença do Agente Auditor na rede e sem fluxo de pacotes. Em seguida, será executado o teste de estresse, para determinar o uso de CPU do CA em situações de alto tráfego. Ademais, será aferido o tempo médio de processamento dos pacotes e a latência inserida no *ping* da rede pelo agente. Para cálculo da média, este experimento será realizado três vezes.

### 3.4.3 Carga do Agente Auditor (AA)

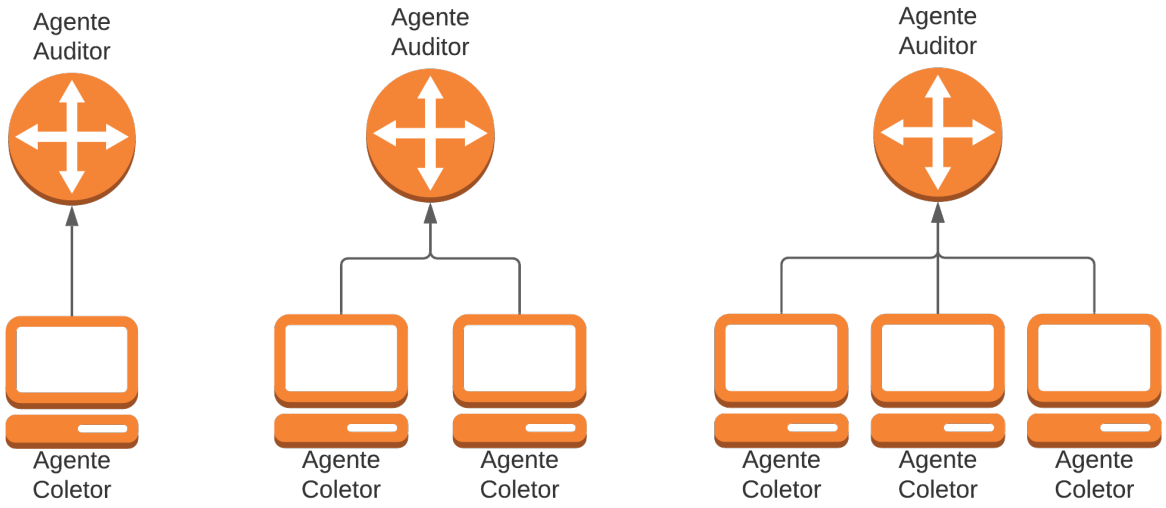
A determinação da carga gerada pelo Agente Auditor será realizada também utilizando o teste de velocidade utilizado nos outros experimentos. Os detalhes de uso de CPU serão coletados a partir da interface LuCI do sistema OpenWRT. Para este experimento será coletado um dos gráficos de uso de CPU gerados pela ferramenta de estatísticas da LuCI.

### 3.4.4 Limite de Operação do Sistema

Para medição do limite de operação do sistema, será criado um script, representado no anexo I, para geração de requisições a uma taxa aproximadamente constante de requisições por segundo. Este script será executado nas configurações propostas por [4]. Para cada configuração proposta serão processados 10.000 pacotes pelo Agente Auditor.

| | |
|---|---|
| **CPU** | 3 núcleos |
| **RAM** | 4096 MB |
| **Memória de vídeo** | 16 MB |
| **Disco** | 50 GB dinâmico |
| **Arquitetura** | 64-bit |

Tabela 3.2: Configurações da VM




(a) 1 CA (b) 2 CA's (c) 3 CA's

Figura 3.4: Configurações de rede para o teste de limite de operação

Na Figura 3.4 estão apresentadas as 3 configurações de rede utilizadas para o teste de limite de operação. Inicialmente será utilizado apenas um CA, como representado na Figura 3.4a. Em seguida serão testados os cenários das Figuras 3.4b e 3.4c, em que a rede terá 2 ou 3 CA's, respectivamente.

Em cada configuração de rede serão testados 4 intervalos entre requisições HTTP(s): 1, 3, 5 ou 10 segundos. Não há neste teste nenhum CA infectado, logo, todos os pacotes devem ser marcados como lícitos pelo sistema. Desta forma, será possível medir a taxa de falso positivo do sistema. O script utilizado para a execução deste teste pode ser visualizado no anexo I.

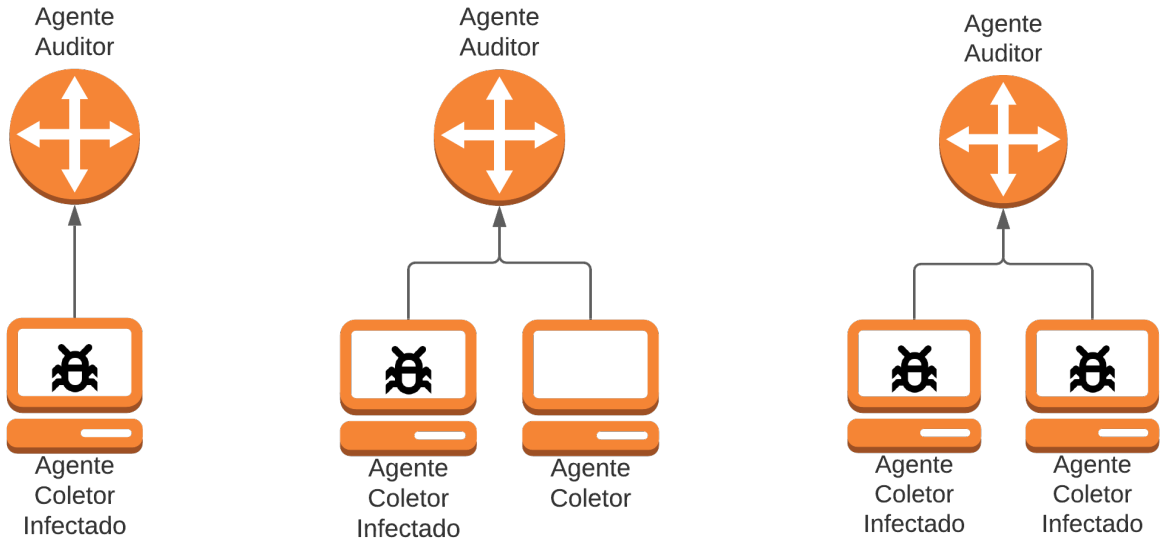
### 3.4.5 Teste de Detecção

Para a verificação da capacidade de detecção serão executados os cenários propostos por [4] e descritos na Tabela 3.3 com os respectivos objetivos. Neste experimento o tráfego lícito será gerado a cada 10 segundos e o tráfego ilícito será gerado em quatro intervalos: 5 minutos, 1 minuto, 30 segundos e 10 segundos. No total, serão inspecionados 2.000 pacotes em cada cenário. No anexo II está o algoritmo utilizado para geração de tráfego ilícito e no anexo III, o *script* utilizado para associar a geração de pacotes lícitos e ilícitos.

Na Figura 3.5 é possível visualizar as configurações de rede utilizadas no teste de detecção. Na Figura 3.5a está representado o cenário em que há apenas um nó infectado. Já na Figura 3.5b é possível observar a configuração de rede com 2 CA's sendo apenas um infectado. Por último, a Figura 3.5c representa a configuração do experimento em que há 2 nós infectados na rede.

| Cenário | Objetivo |
|---|---|
| 1 CA infectado e gerando tráfego lícito e ilícito | Verificar a capacidade de distinção de tráfego oriundo de um mesmo nó |
| 2 CA's, um infectado e outro não infectado | Verificar a capacidade do sistema de diferenciar nós comprometidos |
| 2 CA's infectados | Verificar a capacidade do sistema de diferenciar sinais maliciosos de diferentes nós comprometidos |

Tabela 3.3: Descrição dos cenário de teste




(a) 1 CA (b) 2 CA's (c) 3 CA's

Figura 3.5: Configurações de rede para o teste de limite de detecção

Durante este teste, as requisições consideradas lícitas serão todas aquelas geradas pelo script do anexo I que buscam *websites* diversos. Nestes testes, o *rootkit* foi configurado para ocultar pacotes direcionados para o IP do domínio `smartview-api.novaweb.duckdns.org`, cujo uso para testes foi autorizado pelos responsáveis. Este endereço foi escolhido devido a facilidade de associar seu IP ao domínio.

A partir desses dados será construída a matriz de confusão para determinar a acurácia, a sensibilidade e a taxa de erro do sistema. Além disso, será determinada a capacidade do sistema de gerar uma resposta de memória.

## 3.5 Síntese

Neste capítulo foi apresentado o NERD e seus algoritmos, sendo propostos experimentos para avaliar diferentes aspectos como sua capacidade de detecção, sua taxa de falsos

positivos e seu impacto na rede. No próximo capítulo serão apresentados os resultados destes experimentos e será apresentada uma análise comparativa da ferramenta em relação ao MADEX.

# Capítulo 4

# Resultados e Análise

## 4.1 Capacidade da rede

Nas Tabelas 4.1 e 4.2 é possível visualizar os resultados obtidos no teste da rede sem nenhum componente do sistema em operação. Os valores de média serão utilizados como base para verificação da sobrecarga do sistema.

| **Teste** | **Ping [ms]** | **Download [Mbps]** | **Upload [Mbps]** | **Pacotes TCP capturados** |
|---|---|---|---|---|
| 1 | 3 | 293,790 | 153,690 | 317.429 |
| 2 | 3 | 289,350 | 153,800 | 390.312 |
| 3 | 2 | 292,520 | 153,740 | 387.634 |
| 4 | 2 | 292,630 | 153,670 | 378.405 |
| 5 | 2 | 286,720 | 153,620 | 379.735 |
| Média | 2,4 | 291,002 | 153,704 | 370.703 |

Tabela 4.1: Resultados do teste de carga da rede de alta capacidade

| **Teste** | **Ping [ms]** | **Download [Mbps]** | **Upload [Mbps]** |
|---|---|---|---|
| Média | 10,33 | 50,06 | 10,44 |

Tabela 4.2: Resultados do teste de carga da rede de baixa capacidade

Na Tabela 4.3 é possível visualizar os resultados do desempenho da rede com e sem o sistema ativo. A partir dos dados, observamos que para a rede de baixo desempenho houve uma degradação de cerca de 50% na capacidade da rede, um valor significativo. Na rede de alto desempenho a degradação percebida é percentualmente maior, atingindo quase 90% da capacidade, contudo, o valor absoluto é muito próximo àquele da rede de baixo desempenho.

A banda obtida em ambos os cenários com o sistema ativo evidenciam que a capacidade percebida representa o limite de processamento do hardware utilizado e não uma degradação proporcional à capacidade original da rede. Desta forma, deve ser possível reduzir o impacto do NERD na rede utilizando um roteador de borda com melhor capacidade de processamento.

| | **NERD** | **ping [ms]** | **Download [Mbps** | **Upload [Mbps]** |
|---|---|---|---|---|
| **Alto Desemp.** | Não Ativo | 2,4 | 291,002 | 153,704 |
| | Ativo | 8 | 33,45 | 5,18 |
| **Baixo Desemp.** | Não Ativo | 10,33 | 50,06 | 10,44 |
| | Ativo | 28,33 | 24,85 | 4,47 |

Tabela 4.3: Comparativo do desempenho da rede

## 4.2 Carga do Agente Coletor (CA)

Na Tabela 4.4 é possível visualizar o uso de CPU na vítima com e sem o sistema e em níveis de carga diferentes. Para este teste, é considerado o sistema com carga, o sistema durante a execução do teste de velocidade.

A partir destes dados é evidente que o Agente Coletor apresenta um consumo de CPU considerável, o que pode restringir sua aplicabilidade em sistemas limitados. Durante o pico de carga, o consumo de CPU no nó apresentou um aumento de 78,605% e o consumo base do sistema sem carga foi de 14,311%.

| **Teste** | **Uso médio de CPU** |
|---|---|
| Nó sem sistema | 5,058% |
| Sistema sem carga | 19,369% |
| Sistema com carga | 83,663% |

Tabela 4.4: Uso de CPU do nó com CA

A partir da Tabela 4.5 é possível verificar que o CA não causou nenhum tipo de degradação da capacidade da rede. Este comportamento é esperado, uma vez que o sistema não interfere no tráfego do nó, apenas realiza uma captura utilizando a biblioteca `libpcap`, que observa o tráfego percebido pelo nó. Ademais, o tempo médio de processamento dos pacotes calculado pelo CA é informado na Tabela 4.5.

| Teste | Ping [ms] | Download [Mbps] | Upload [Mbps] | Tempo médio de processamento [ms] | Uso médio de CPU [%] |
|---|---|---|---|---|---|
| 1 | 4 | 291,510 | 148,640 | 0,41945 | 86,183 |
| 2 | 3 | 293,100 | 140,970 | 0,40844 | 83,653 |
| 3 | 3 | 286,550 | 142,880 | 0,41191 | 81,152 |
| Média | 3,3 | 290,387 | 144,163 | 0,41327 | 83,663 |

Tabela 4.5: Resultados do teste de carga da rede

## 4.3 Carga do Agente Auditor (AA)

Na Figura 4.1 é possível visualizar o uso de CPU do agente auditor com intervalo de captura de 1 segundo e tempo total de 60 segundos. Neste gráfico, é possível visualizar que, em situação de espera, isto é, sem tráfego de pacotes considerável, o sistema consome cerca de 50% de CPU. Além disso, são evidentes dois picos de consumo, referentes aos testes de download e upload, em que o uso de CPU se manteve acima de 95%, com uso expressivo de interrupções de software.

A partir desse gráfico, é coerente concluir que o impacto observado na rede decorre do limite do hardware de processamento e não da arquitetura proposta, desta forma, seria possível alcançar melhores desempenhos com roteadores melhores que o R8000 utilizado. Observa-se também que embora existam picos de consumo, o sistema rapidamente se recupera, sendo possível observar o consumo basal entre os testes de download e upload, que possuem um intervalo de menos de 5 segundos entre si.

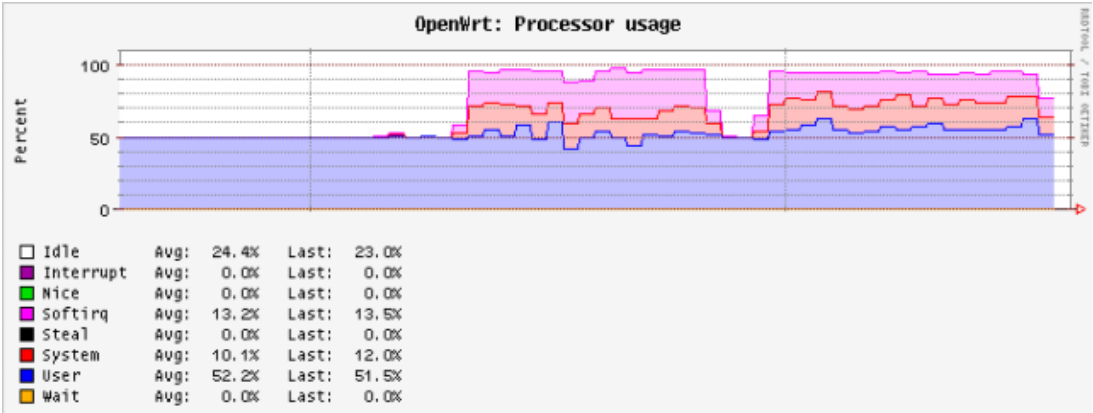



Figura 4.1: Gráfico de uso de CPU durante teste de carga

## 4.4 Limite de Operação do Sistema

Durante o teste do limite de operação do sistema, foram testados os cenários propostos em [4] e descritos na Seção 3.4.4. Os resultados obtidos para as taxas de falsos positivos estão representados na Tabela 4.6a. Os resultados obtidos pela arquitetura MADEX, proposta por Marques *et al.* em [4] estão reproduzidos na Tabela 4.6b.

Para estes testes, não era gerado tráfego ilícito, desta forma, todo alerta é considerado um falso positivo, tanto para a arquitetura MADEX, quanto para a NERD.

| | **1 CA** | **2 CA's** | **3 CA's** |
|---|---|---|---|
| **Intervalos** | ***FP*** | | |
| 10 | 0% | 0,01% | 0,01% |
| 5 | 0% | 0,04% | 0,08% |
| 3 | 0% | 0% | 0,04% |
| 1 | 0% | 0% | 0,04% |

(a) Taxas de falsos positivos do NERD

| | **1 CA** | **2 CA's** | **3 CA's** |
|---|---|---|---|
| **Intervalos** | ***FP*** | | |
| 10 | 0,04% | 0,06% | 0,11% |
| 5 | 0,06% | 0,08% | 0,09% |
| 3 | 0,04% | 0,09% | 0,10% |
| 1 | 0,01% | 0,10% | 100%[a] |

[a] Nesta taxa o sistema falha e classifica todo o tráfego como ilícito.

(b) Taxas de falsos positivos do MADEX

**Fonte:** [4]

Tabela 4.6: Taxas de falso positivo

A partir da comparação das Tabelas, é evidente que a arquitetura NERD proposta foi capaz de diminuir a quantidade de falsos positivos durante o processamento, chegando a não emitir falsos positivos em 50% das taxas testadas. Ademais, o limite de operação do MADEX era alcançado no cenário com maior carga, visto que nesta taxa o sistema classifica todo o tráfego como ilícito, e para a proposta deste trabalho, 99,96% do tráfego é classificado corretamente sob esta carga.

Em todos os testes, NERD apresentou taxas de falsos positivos inferiores àquelas do MADEX, o que mostra que foi possível atingir um dos objetivos propostos neste trabalho.

## 4.5 Teste de Detecção

Na Tabela 4.7 está representada a matriz de confusão obtida a partir dos cenários de teste de detecção descritos na Seção 3.4.5 e na Tabela 3.3. A partir da matriz de confusão da Tabela 4.7 é possível obter a taxa de erro do sistema, de 0,004%, a acurácia, de 99,996%, e a sensibilidade, de 100%. Isto é, apenas 0,004% do tráfego lícito foi marcado como ilícito e nenhum pacote malicioso foi aceito pelo NERD. Isto representa um avanço quando comparado ao MADEX de nível 1, que possui taxa de erro de 1,21%, acurácia de 98,78% e sensibilidade de 99,97%.

| | **Positivo** | **Negativo** |
|---|---|---|
| **Positivo** | 310 | 0 |
| **Negativo** | 1 | 23.689 |

Tabela 4.7: Matriz de confusão do NERD

Na Tabela 4.8 está representada a quantidade de ciclos que o NERD executou desde a primeira requisição ilícita até o momento em que o nó foi marcado como infectado em cada cenário. Na Tabela 4.9 está representado o equivalente para a arquitetura MADEX. Embora seja possível perceber que houve certa degradação nesta funcionalidade, a nova arquitetura não permitiu que nenhum pacote malicioso chegasse à internet, diferentemente da estratégia implementada pelo MADEX, em que esta atitude dependia da implementação de quarentena decorrente da confirmação de infecção.

| **Malicious traffic** | $1^{st}$ **scenario** | $2^{nd}$ **scenario** | $3^{rd}$ **scenario** |
|---|---|---|---|
| Every 5 minutes | - | - | - |
| Every 1 minute | 1 cycle | - | - |
| Every 30 seconds | 1 cycles | - | 1 cycle |
| Every 10 seconds | 1 cycle | 2 cycle | 1 cycle |

Tabela 4.8: Performance de detecção do NERD

| **Malicious traffic** | $1^{st}$ **scenario** | $2^{nd}$ **scenario** | $3^{rd}$ **scenario** |
|---|---|---|---|
| Every 5 minutes | 11 cycles | 18 cycles | 20 cycles |
| Every 1 minute | 2 cycles | 5 cycles | 6 cycles |
| Every 30 seconds | 2 cycles | 2 cycles | 1 cycle |
| Every 10 seconds | 1 cycle | 1 cycle | 1 cycle |

Tabela 4.9: Performance de detecção do MADEX **Fonte:** [4]

Esta dificuldade de geração de uma resposta de memória se deve ao ciclo de detecção limitado em 200 pacotes e à quantidade de pacotes ilícitos emitidos por tentativa, fixada em 3, número inferior ao limiar de 6 pacotes. Ao final deste ciclo, as contagens de pacotes ilícitos são reiniciadas, desta forma há diversas cargas em que não são emitidos pacotes ilícitos suficientes por ciclo. É possível melhorar a performance deste componente reduzindo a tolerância do sistema de 6 pacotes ilícitos para um valor menor. Durante testes sem tráfego ilícito, os falsos positivos emitidos não ultrapassaram dois pacotes por endereço, desta forma, seria possível adotar este valor como limite, o que favoreceria a geração de memória sem grandes impactos nos falsos positivos emitidos pelo sistema.

Ademais, esta diferença observada entre o MADEX e o NERD pode ser justificada pela capacidade do primeiro de acumular a contagem de sinais ilícitos entre ciclos de detecção, que, se implementada nesta proposta, possivelmente contribuiria também para o desenvolvimento da resposta de memória. De toda forma, este aspecto de geração de resposta duradoura requer mais investigação.

## 4.6 Considerações finais

Na Tabela 4.10 é mostrada uma comparação da arquitetura proposta com dados reportados por outras ferramentas para detecção de *rootkits*, em que é evidente que este trabalho foi capaz de reduzir os falsos positivos e melhorar a sensibilidade, cumprindo os objetivos propostos.

| **Ferramenta** | **Taxa de erro** | **Acurácia** | **Sensibilidade** |
|---|---|---|---|
| **NERD** | 0,004% | 99,996% | 100% |
| **MADEX Level 1** [4] | 1,21% | 98,78% | 99,97% |
| **VKRD Random Forest** [11] | 3,25% | 96,74% | 95,13% |
| **KRHFDV** [10] | 70,31% | 29,68% | 22,41% |

Tabela 4.10: Comparação com outras ferramentas propostas

Desta forma, é possível observar que a ferramenta proposta, NERD, foi capaz de atingir sensibilidade acima do observado na literatura, inclusive em relação à arquitetura MADEX, que inspirou este trabalho. Contudo, é possível perceber que o sistema possui um impacto no desempenho da rede, em especial devido ao roteador utilizado. É necessário mais estudo em relação à capacidade do sistema de gerar uma resposta duradoura, aspecto impactado em comparação ao MADEX. Ao considerar todos estes aspectos, avalia-se que a troca de desempenho por segurança se configura como benéfica nos resultados obtidos.

# Capítulo 5

# Conclusão

Neste trabalho foi proposto e implementado o NERD, uma arquitetura melhorada capaz de analisar tráfego de rede em busca de anomalias em cargas superiores àquelas suportadas pelo MADEX proposto em [4]. Durante os testes deste trabalho, o sistema foi capaz de interromper o fluxo de todos os pacotes maliciosos na rede, contudo, a capacidade de geração de memória foi comparativamente reduzida, uma troca aceitável dentro dos objetivos do projeto.

É evidente que a proposta foi capaz de alcançar taxas de falsos positivos abaixo de outras referências com uma maior sensibilidade. O impacto do NERD no desempenho da rede, em especial em redes de alto desempenho, pode ser considerado elevado, o que leva à conclusão de que o hardware utilizado na borda da rede foi insuficiente, embora a velocidade atingida ainda possa ser suficiente para aplicações domésticas, incluindo *streaming* de vídeo de alta qualidade.

## 5.1 Trabalhos futuros

Em projetos futuros, propõe-se que seja investigada a possibilidade de utilizar o NERD para detectar outras técnicas de ocultação de tráfego e exfiltração de dados que não se limitem a *rootkits*. Ademais, tendo em vista a baixa frequência de falsos positivos, a experimentação com menores tolerâncias deve levar a uma melhor resposta de memória com taxas de falsos positivos dentro de limiares aceitáveis. Baseado no MADEX, a utilização de memória entre diferentes ciclos de detecção também pode ter um efeito positivo na resposta do sistema e pode ser um incremento relevante.

Outra questão pendente na arquitetura atual é a implementação do AA que foi feita em espaço de usuário e poderia apresentar ganhos significativos se realizada em espaço de kernel. Possivelmente, este incremento contribuiria na redução do impacto no desempenho da rede, que foi significativo. Esforços de otimização deste algoritmo não devem ser

restritos a migração para espaço de kernel, visto que são imprescindíveis para a viabilidade do sistema em cenários reais.

# Referências


[1] Eresheim, Sebastian, Robert Luh e Sebastian Schrittwieser: *The evolution of process hiding techniques in malware-current threats and possible countermeasures.* Journal of Information Processing, 25:866–874, 2017. viii, 1, 4, 5

[2] Fernandes, Diogo AB, Mário M Freire, Paulo AP Fazendeiro e Pedro RM Inácio: *Applications of artificial immune systems to computer security: A survey.* Journal of Information Security and Applications, 35:138–159, 2017. viii, 6, 7, 8, 9, 10, 11, 16

[3] Abbas, Abul K, Andrew H Lichtman e Shiv Pillai: *Cellular and molecular immunology E-book.* Elsevier Health Sciences, 2012. viii, 8, 9

[4] Marques, Rafael Salema, Gregory Epiphaniou, Haider Al-Khateeb, Carsten Maple, Mohammad Hammoudeh, Paulo Andre Lima De Castro, Ali Dehghantanha e Kim Kwang Raymond Choo: *A flow-based multi-agent data exfiltration detection architecture for ultra-low latency networks.* 2020. viii, ix, 1, 2, 3, 6, 12, 13, 15, 16, 19, 22, 24, 25, 31, 32, 33, 34

[5] OpenWrt: *[openwrt wiki] techdata: Netgear r8000*, 2021. `https://openwrt.org/toh/hwdata/netgear/netgear_r8000`, acesso em 01/09/2021. ix, 22, 23

[6] Lee, Geraldine, Gregory Epiphaniou, Haider Al-Khateeb e Carsten Maple: *Security and privacy of things: Regulatory challenges and gaps for the secure integration of cyber-physical systems.* Em *Third International Congress on Information and Communication Technology*, páginas 1–12. Springer, 2019. 1

[7] Kemp, Simon: *Digital 2020: Global digital overview*, 2020. `https://datareportal.com/reports/digital-2020-global-digital-overview`, acesso em 2021-09-18. 1

[8] Morgan, Steve: *Cybercrime to cost the world $10.5 trillion annually by 2025.* Cybercrime Magazine, 13, 2020. 1

[9] Ussath, Martin, David Jaeger, Feng Cheng e Christoph Meinel: *Advanced persistent threats: Behind the scenes.* Em *2016 Annual Conference on Information Science and Systems (CISS)*, páginas 181–186. IEEE, 2016. 1, 5

[10] Geetha Ramani, R e S Suresh Kumar: *Nonvolatile kernel rootkit detection using cross-view clean boot in cloud computing.* Concurrency and Computation: Practice and Experience, 33(3):e5239, 2021. 1, 4, 5, 33

[11] Tian, Donghai, Rui Ma, Xiaoqi Jia e Changzhen Hu: *A kernel rootkit detection approach based on virtualization and machine learning*. IEEE Access, 7:91657–91666, 2019. 1, 4, 5, 6, 33

[12] Openwall: *Lkrg - linux kernel runtime guard*, 2021. `https://www.openwall.com/lkrg/`, acesso em 19/10/2021. 1

[13] team, Rootkit Hunter project: *The rootkit hunter project*, 2018. `http://rkhunter.sourceforge.net/`, acesso em 19/10/2021. 1

[14] Murilo, Nelson e Klaus Steding-Jessen: *chkrootkit – locally checks for signs of a rootkit*, 2021. `http://www.chkrootkit.org/`, acesso em 19/10/2021. 1

[15] Junnila, Juho: *Effectiveness of linux rootkit detection tools*. 2020. 1, 4, 6, 23

[16] ropch4ins: *Nuk3gh0st: Universal linux lkm rootkit, designed to work in any kernel version and both architectures.*, 2019. `https://github.com/ropch4ins/Nuk3Gh0st`, acesso em 05/06/2021. 4, 5, 6, 23

[17] Bhushan, Anshumali e Deepa Sharma: *Hypervisor based security*. International Journal of Engineering Research and Applications (IJERA), 10:26–30, 2020. 5

[18] Todd, A, J Benson, G Peterson, T Franz, Michael Stevens e Richard Raines: *Analysis of tools for detecting rootkits and hidden processes*. Em *IFIP International Conference on Digital Forensics*, páginas 89–105. Springer, 2007. 6

[19] Perelson, Alan S: *Immune network theory*. Immunol. Rev, 110(5), 1989. 7, 8

[20] Matzinger, Polly: *Tolerance, danger, and the extended family*. Annual review of immunology, 12(1):991–1045, 1994. 7, 8, 9, 10, 16

[21] Jerne, Niels K: *Towards a network theory of the immune system*. Ann. Immunol., 125:373–389, 1974. 7

[22] Aickelin, Uwe, Peter Bentley, Steve Cayzer, Jungwon Kim e Julie McLeod: *Danger theory: The link between ais and ids?* Em *International Conference on Artificial Immune Systems*, páginas 147–155. Springer, 2003. 8, 11

[23] Forrest, Stephanie, Alan S Perelson, Lawrence Allen e Rajesh Cherukuri: *Self-nonself discrimination in a computer*. Em *Proceedings of 1994 IEEE computer society symposium on research in security and privacy*, páginas 202–212. Ieee, 1994. 10

[24] D'haeseleer, Patrik, Stephanie Forrest e Paul Helman: *An immunological approach to change detection: Algorithms, analysis and implications*. Em *Proceedings 1996 IEEE Symposium on Security and Privacy*, páginas 110–119. IEEE, 1996. 10

[25] Wierzchon, Slawomir T: *Discriminative power of the receptors activated by k-contiguous bits rule*. Journal of Computer Science & Technology, 1, 2000. 10

[26] Ayara, Modupe, Jon Timmis, Rogerio de Lemos, Leandro N de Castro e Ross Duncan: *Negative selection: How to generate detectors*. Em *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, volume 1, páginas 89–98. University of Kent at Canterbury Printing Unit University of Kent at Canterbury, 2002. 10

[27] Hang, Xiaoshu e Honghua Dai: *Constructing detectors in schema complementary space for anomaly detection*. Em *Genetic and Evolutionary Computation Conference*, páginas 275–286. Springer, 2004. 10

[28] Dasgupta, Dipankar, Kalmanje KrishnaKumar, D Wong e Misty Berry: *Negative selection algorithm for aircraft fault detection*. Em *International Conference on Artificial Immune Systems*, páginas 1–13. Springer, 2004. 10

[29] Ji, Zhou e Dipankar Dasgupta: *Revisiting negative selection algorithms*. Evolutionary computation, 15(2):223–251, 2007. 10

[30] De Castro, Leandro N e Fernando J Von Zuben: *Learning and optimization using the clonal selection principle*. IEEE transactions on evolutionary computation, 6(3):239–251, 2002. 10

[31] Garrett, Simon M: *How do we evaluate artificial immune systems?* Evolutionary computation, 13(2):145–177, 2005. 11

[32] Fernandes, Diogo AB, Liliana FB Soares, João V Gomes, Mário M Freire e Pedro RM Inácio: *A quick perspective on the current state in cybersecurity*. Em *Emerging trends in ICT security*, páginas 423–442. Elsevier, 2014. 11

[33] Hofmeyr, Steven A e Stephanie Forrest: *Architecture for an artificial immune system*. Evolutionary Computation, 7(1):45–68, 1999. 11

[34] Greensmith, Julie, Uwe Aickelin e Steve Cayzer: *Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection*. Em *International Conference on Artificial Immune Systems*, páginas 153–167. Springer, 2005. 11

[35] Julian, Vicente e Vicente Botti: *Multi-agent systems*. Applied Sciences, 9(7), 2019, ISSN 2076-3417. `https://www.mdpi.com/2076-3417/9/7/1402`. 11

[36] Al-Yaseen, Wathiq Laftah, Zulaiha Ali Othman e Mohd Zakree Ahmad Nazri: *Real-time multi-agent system for an adaptive intrusion detection system*. Pattern Recognition Letters, 85:56–64, 2017. 11

[37] Seresht, Neda Afzali e Reza Azmi: *Mais-ids: A distributed intrusion detection system using multi-agent ais approach*. Engineering Applications of Artificial Intelligence, 35:286–298, 2014. 12

[38] Picheta, Dominik e Hugo Locurcio: *Nim programming language*, 2021. `https://nim-lang.org/`, acesso em 18/08/2021. 19

[39] OpenWrt: *Welcome to the openwrt project*, 2020. `https://openwrt.org/start`, acesso em 22/04/2021. 19

# Anexo I

# *Script* para execução de requisições lícitas em intervalos regulares

```
urls=("google.com" \
      "facebook.com" \
      "stackoverflow.com" \
      "github.com" \
      "amazon.com.br")
sleep_time=$1
RANDOM=$$$(date +%s)
while [ 1 ]
do
    selected_url=${urls[$RANDOM % ${#urls[@]}]}
    echo "Testing $selected_url"
    curl -L "https://$selected_url" --retry 2 --retry-max-time 10 \
        --connect-timeout 5 &>/dev/null || echo "Was blocked"
    sleep $sleep_time
done
```

# Anexo II

# *Script* para execução de requisições ilícitas em intervalos regulares

```
sleep_time=$1
while [ 1 ]
do
    sleep $sleep_time
    echo "Testing adversary"
    load-nuk3gh0st
    curl -L "smartview.novaweb.duckdns.org:16443" --retry 1 --retry-max-time 5 \
        --connect-timeout 5 &>/dev/null || echo "Was blocked"
    unload-nuk3gh0st
done
```

# Anexo III

# *Script* para execução de teste de detecção com requisições lícitas e ilícitas

```
./test_request.sh 10 &
sleep 3
./test_adversary.sh $1 &
```