# Um Sistema Acurado de Detecção de Ameaças em Tempo Real por Processamento de Fluxos*

**Antonio Gonzalez Pastana Lobato,**
**Martin Andreoni Lopez,**
**Otto Carlos Muniz Bandeira Duarte**

[1]Grupo de Teleinformática e Automação – GTA
Universidade Federal do Rio de Janeiro – UFRJ
Rio de Janeiro – RJ – Brasil

{antonio, martin, otto}@gta.ufrj.br

***Abstract.*** *The late detection of security threats causes a significant increase of the risk of irreparable damages, disabling any defense attempt. Although very hard to analyze, attacks always leave traces that can be detected. This paper proposes a real time streaming threat detection system based on machine learning algorithms. The system architecture combines the advantages of real time stream processing with the batch processing over a historical database that does not require manual interaction of security specialists. The proposed system allows anomaly-based detection of known and zero-day attacks. The system was developed and evaluated from a data set constructed with the capture of legitimate and malicious network traffic. Results show that the proposed system presents an accurate threat detection with low processing time, permitting prompt defense strategies.*

***Resumo.*** *A detecção tardia de ameaças provoca o aumento substancial dos riscos de danos irreparáveis e inviabiliza qualquer tentativa de defesa. Embora sejam difíceis de serem descobertos, os ataques deixam vestígios que podem ser detectados. Este artigo propõe um sistema de detecção de ameaças em tempo real por processamento de fluxos baseado em algoritmos de aprendizado de máquinas. A arquitetura do sistema combina as vantagens do processamento em tempo real de fluxos com as do processamento em lotes sobre uma base histórica e não requer a intervenção de especialistas de segurança para a análise e readaptações às ameaças. O sistema proposto permite tanto a detecção de ataques conhecidos quanto de novos ataques, ainda desconhecidos, através de métodos automatizados de classificação de ataques e de detecção de anomalias. O sistema foi desenvolvido e avaliado a partir de um conjunto de dados construído com a captura de tráfego de rede legítimo e malicioso. Os resultados mostram que o sistema apresenta uma acurada detecção de ameaças com baixo tempo de processamento, o que possibilita estratégias de defesa.*

## 1. Introdução

Os riscos e os ataques de segurança são atualmente muito altos e tendem a aumentar significativamente no futuro com a introdução da Internet das coisas (*Internet of Things*- IoT), uma vez que estima-se em mais de 80 bilhões os dispositivos interconectados até 2025 [Clay 2015]. Esse cenário exibe uma alta complexidade no gerenciamento e na proteção das redes de comunicação, gerando desafios na segurança e na privacidade dos dados. Os bilhões de dispositivos proveem grandes massas de dados (*Big Data*) [Costa et al. 2012], gerados em forma

*Este trabalho foi realizado com recursos da CNPq, CAPES, FAPERJ e FINEP.

de fluxos, que precisam ser gerenciados, processados, transferidos e armazenados de forma segura em tempo real [Porto e Ziviani 2014]. Esse cenário introduz novos desafios tecnológicos, uma vez que as tecnologias atuais não foram projetadas para essa demanda. Além disso, a virtualização, amplamente utilizada em sistemas de computação em nuvem, introduz novas ameaças. Assim, tanto o uso da tecnologia de virtualização quanto a velocidade, o volume e a variedade das grandes massas de dados são fatores que geram novas vulnerabilidades.

O tempo de detecção de uma ameaça é crucial para manter a segurança nos sistemas de comunicação [Wu et al. 2014]. A detecção de ameaças de forma eficiente exige o monitoramento, o processamento e o gerenciamento das grandes massas de dados, que permitem extrair informações da caracterização do tráfego. No entanto, detectar ameaças em grandes massas de dados requer o desenvolvimento de técnicas modernas de análise. Os atuais sistemas projetados para coletar dados de diferentes fontes e gerenciar em um ponto único as informações de segurança, como o *S*ecurity Information and Event Management (SIEM), não são eficazes, pois 85% das intrusões na rede são detectadas depois de semanas de sua ocorrência [Ponemon e IBM 2015]. Além disso, a reação às ameaças é muito lenta, em média 123 horas depois de ocorridas, e a detecção no vazamento de informações demora em média 206 dias [Clay 2015]. Portanto, o longo tempo de detecção dessas ameaças impossibilita qualquer tipo de defesa. Logo, as técnicas de analítica[1] para processamento de fluxo em tempo real permitem o imediato processamento e análise de diferentes tipos de dados e, consequentemente, a detecção de ameaças. Um outro problema conhecido é a enorme quantidade de alarmes gerados pelos sistemas atuais de monitoramento. Esses alertas levam a falsos positivos, que são, na maior parte, ignorados por analistas de segurança, junto com eventuais reais positivos que representam graves ameaças, por não conseguirem lidar com a quantidade de dados que chegam.

Este artigo propõe e desenvolve um sistema para a detecção de ameaças em tempo real, utilizando diversas plataformas de código aberto. A integração das diversas plataformas permite a análise de grandes volumes de dados pelo processamento por fluxo (*stream processing*). O sistema proposto utiliza técnicas de aprendizado de máquina para a classificação de ataques e a detecção de anomalias. Além disso, para a avaliação do sistema, foi criado um conjunto de dados com as classes marcadas, contendo o uso normal e ataques, a partir da captura de tráfego de rede, abstraídos em fluxos. A arquitetura do sistema utiliza o conceito da arquitetura lambda [Marz e Warren 2013], na qual é combinado o processamento tradicional em lotes (*batch*) sobre uma base histórica, com o processamento em tempo real por análise de fluxos. Assim, o sistema proposto é capaz de detectar tanto ataques conhecidos quanto os novos ataques, ainda desconhecidos, através de métodos automatizados de classificação de ataques e de detecção de anomalias. Um protótipo do sistema foi desenvolvido e avaliado. Os resultados mostram uma alta acurácia do sistema desenvolvido para detectar ameaças. Além do sistema proposto detectar rapidamente ameaças de segurança, o processamento de fluxos é associado ao processamento em tempo diferenciado de dados armazenados historicamente para adaptar os algoritmos de detecção em tempo real. Isto resulta em um sistema com um valor baixo de falsos positivos, possibilitando a implementação de estratégias de defesa.

O restante do artigo está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. O sistema proposto é apresentado na Seção 3. O conjunto de dados para a avaliação do sistema é apresentado na Seção 4. Na Seção 5 são discutidos os métodos para a detecção de ameaças e os resultados obtidos. Por fim, a Seção 6 conclui o artigo.

[1]Analítica é a ciência da análise lógica, isto é, a análise de enormes conjuntos de dados, usando matemática, estatística e software de computação, para achar padrões de dados que proveem conhecimento, informação útil ou visões de dados em bruto.

## 2. Trabalhos Relacionados

No aprendizado de máquina existem diversas técnicas ou algoritmos que podem ser classificadas de acordo com o procedimento aplicado em supervisionadas ou não-supervisionadas. Na análise supervisionada o conjunto de dados é totalmente etiquetado. Este tipo de análise é usado para a classificação de ataques. Entre os exemplos mais conhecidos de técnicas usadas na classificação estão as redes neurais, as árvores de decisão e as máquinas de vetores de suporte (*Support Vector Machine* - SVM) [Buczak e Guven 2015, Stroeh et al. 2013]. Esses artigos utilizam o conjunto de dados KDD 99, um dos poucos conjuntos de dados que existem na área de pesquisa relacionada à detecção de Intrusão. Na análise não-supervisionada, o conjunto de dados é utilizado sem informação dos tipos de classes às quais eles pertencem. Este tipo de análise é aplicado na detecção de padrões. No entanto, nenhuma destas técnicas foi ainda aplicada em tempo real.

O Snort e o Bro são as ferramentas de *Software* Livre mais populares que realizam a Detecção de Intrusão em tempo real [Rai e Devi 2013]. O Snort utiliza apenas o método de detecção por assinatura, não detectando pequenas variações dos ataques e requer atualizações constantes na base de dados das assinaturas. O Bro, por sua vez, apresenta uma linguagem para a detecção por anomalias, mas as técnicas para a detecção devem ser criadas pelo próprio usuário. Estas ferramentas podem ser usadas para melhorar a prevenção de ataques, em conjunto com outros mecanismos, como as redes definidas por *software* (*Software Defined Networks*-SDN), as quais são usadas para realizar a detecção e prevenção de intrusão em redes virtuais [Andreoni Lopez et al. 2014, Lobato et al. 2014].

Holtz *et al.* propõem um sistema de detecção de intrusão para grandes massas de dados utilizando técnicas de aprendizado de máquinas *Map-Reduce* [Holtz et al. 2011]. Embora a técnica do *Map-Reduce* seja factível para analisar grandes massas de dados, ela não possui bom desempenho para detecção de ameaças em tempo real.

Outro aspecto importante abordado por diversos pesquisadores é a tarefa de criação de conjuntos de dados para analisar, avaliar e comparar propostas de sistemas de detecção de ameaças. WebClass [Ringberg et al. 2008] é uma ferramenta *web* que permite a contribuição manual dos usuários na etiquetagem do tráfego de redes. No entanto, esta proposta não chamou a atenção dos pesquisadores e o projeto foi abandonado. Sperotto *et al.* [Sperotto et al. 2009] criaram um conjunto de dados com mais de 14.2 milhões de fluxos etiquetados. Algumas pesquisas criaram seus conjuntos de dados para avaliar as suas propostas [Amini et al. 2006, Sangkatsanee et al. 2011]. Sangkatsanee *et al.* utilizam diferentes algoritmos para realizar a classificação de ataques, enquanto Morteza Amini *et al.* detectam anomalias em tempo real utilizando redes neurais não-supervisionadas. Estes artigos não avaliam suas propostas em cenários que apresentem uma topologia com um grande número de máquinas com seu tráfego sendo analisado.

O SAND [Liu et al. 2014] é uma ferramenta para o monitoramento de redes através do processamento por fluxos, no entanto esta proposta só caracteriza o tráfego de redes sem a detecção de ataques. A detecção de anomalias no desempenho de máquinas virtuais usando técnicas de inteligência de máquinas é proposta em [Solaimani et al. 2014]. O trabalho [Du et al. 2014] não apresenta possibilidade de análises de dados provenientes de diferentes fontes, e no trabalho de Zhao *et al.* [Zhao et al. 2015] os resultados ainda são preliminares para obtenção de maiores conclusões.

Diferentemente dos artigos citados anteriormente, este artigo propõe o uso de plataformas de software aberto em uma arquitetura específica que permite o processamento e a

análise de fluxos de dados em tempo real ao mesmo tempo que se apoia em uma base de dados histórica.

## 3. O Sistema Proposto

A análise de grandes conjuntos de dados estáticos é comumente realizada por processamento em lotes (*batch*). No entanto, esta técnica apresenta grandes latências, com respostas superiores a 30 segundos, enquanto algumas aplicações requerem processamento em tempo real com respostas da ordem de sub-segundo [Rychly et al. 2014]. Por sua vez, a técnica de processamento de fluxo de dados (*streaming processing*) analisa uma sequência massiva de dados ilimitados que são continuamente gerados. Estes paradigmas são combinados na arquitetura lambda para obter analíticas de grandes massas de dados em tempo real.

O sistema proposto é baseado na arquitetura lambda, mostrada na Figura 1, que permite a manipulação e a análise em tempo real de quantidades massivas de informação. A arquitetura lambda [Marz e Warren 2013] é composta por três camadas: a camada de processamento de fluxo de dados, a camada de processamento em lotes e a camada de serviço. A camada de processamento de fluxo de dados trata o fluxo de dados em tempo real. A camada de processamento em lotes analisa grande quantidade de dados armazenados através de técnicas de computação distribuída como *map-reduce*. Finalmente, a camada de serviço agrega as informações obtidas das duas camadas anteriores para criar as saídas dos dados analisados. Assim, o objetivo da arquitetura lambda é analisar, em tempo real e de forma acurada, os fluxos de dados e *logs* de diferentes fontes nas velocidades que são gerados.

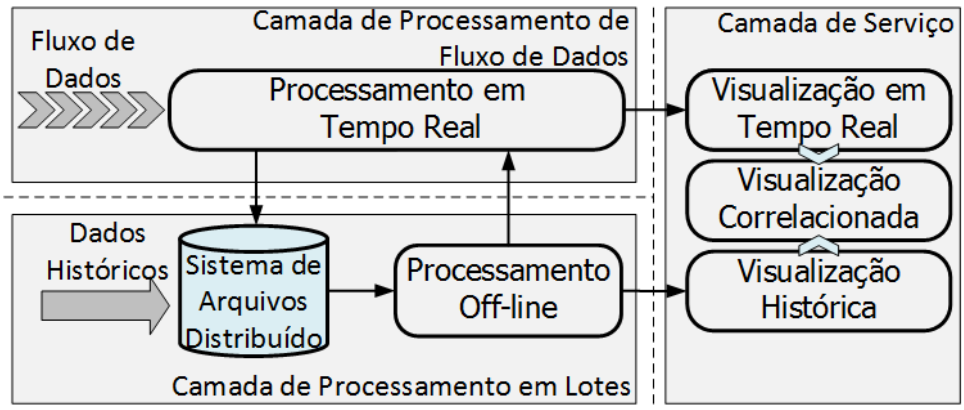



**Figura 1: As três camadas da arquitetura *lambda* de processamento simultâneo em *tempo real* e em tempo diferenciado: processamento de fluxo de dados, processamento em lotes (batch) e serviço.**

A análise dos dados é dividida em três etapas: captura, normalização e processamento. Primeiro, os dados são capturadas no local de coleta. Em seguida, todas essas informações adquiridas são enviadas para o processo de normalização, onde os dados são formatados e enriquecidos a altíssimas taxas de processamento com informações externas, tais como informações geográficas, correlações, etc. Assim, cria-se uma qualidade mais rica de informação sobre um ataque. Com a correlação e o enriquecimento de dados de distintas fontes, ao se detectar uma ameaça, é possível responder a perguntas, tais como: quem foi o atacante, a sua localização geográfica, os locais dos quais as informações vazaram, o destino para o qual as informações foram enviadas, entre outras. Finalmente, os dados normalizados são processados em tempo real, adquirindo os padrões necessários para a segurança. Neste processo, uma vez obtidos os resultados, eles são visualizados e armazenados.

O protótipo de detecção de ameaças do sistema proposto é composto de ferramentas de *software* de código aberto. A configuração do sistema consiste na integração das seguintes ferramentas de código aberto:

**Bro**[2]**:** é uma ferramenta de monitoramento e análise em tempo real do tráfego de rede. O `Bro` possui uma linguagem de programação orientada a redes, o que facilita a abstração dos fluxos;

**Flume**[3]**:** é um serviço distribuído para a coleta, agregação e movimentação de grandes quantidades de dados em formato de *logs*;

**Kafka**[4]**:** é um mediador de mensagens (*Broker*) que funciona como um serviço de Produtor/Consumidor. O `Kafka` abstrai o fluxo de mensagens em tópicos. Os Produtores gravam os dados em tópicos e os consumidores leem os dados a partir desses tópicos;

**Storm**[5]**:** é um processador de eventos em tempo real , também conhecido como *Data Streaming Processor* (DSP). O `Storm` é a ferramenta central da proposta. Este processador é composto por topologias que formam Grafos Acíclicos Dirigidos (GAD) compostos por elementos de entradas, os *spouts*, e elementos de processamento, os *bolts*. É possível usar o paralelismo dos *spouts* e dos *bolts*, fazendo com que as diferentes amostras dos fluxos possam ser processadas em tempo real e de forma paralela. Uma topologia funciona como um canal de dados, que são processados em forma de fluxo à medida que os dados avançam;

**HBase**[6]**:** é uma base de dados distribuída que armazena dados não-relacionados. O `HBase` não suporta pedidos *query* do tipo SQL, como ocorre nas bases de dados estruturadas. No entanto, ela permite o armazenamento de grandes massas de dados dispersos e o acesso aos dados em tempo real.

A Figura 2 mostra como os programas de código aberto são interligados para compor o sistema proposto. A coleta de dados é obtida de diferentes fontes. Essas fontes são *logs* do sistema operacional e também de aplicações, além dos dados de fluxo de redes que são coletados do `Bro` em diferentes pontos da rede. Logo, no sistema proposto, o `Bro` atua como um caracterizador de fluxos, sintetizando as informações dos pacotes que entram pelas interfaces de rede do sistema e agrupando as informações em janelas de tempo.

O sistema visa garantir a segurança de diversos componentes da rede. Para isso, as coletas de dados ocorrem em locais distintos da rede e são agrupadas para serem processadas. Tanto as informações de fluxo geradas pela ferramenta `Bro`, como os *logs* das aplicações, são transportados até o local de análise pela ferramenta Apache `Flume` que atua como um túnel ligando as fontes de dados à parte de processamento do sistema. Como a geração de dados de segurança do sistema possui uma taxa variável em relação ao uso, pode haver momentos em que a plataforma de processamento não consiga processar todos os dados imediatamente. Por isso, os dados de segurança são recebidos pela ferramenta Apache `Kafka` no local de análise. Basicamente, esta ferramenta serve como um *buffer* para a ferramenta de processamento, adequando taxas distintas de geração e processamento. O Apache `Storm` é o núcleo de processamento em tempo real da plataforma, para detecção de ameaças. O `Storm` oferece um método de computação processamento de fluxos distribuído e tolerante falhas. Assim, o `Storm` realiza o processamento em memória, garantindo baixa latência em tempo real.

[2]Bro IDS: https://www.bro.org/
[3]Apache Flume: https://flume.apache.org/
[4]Apache Kafka: http://kafka.apache.org/
[5]Apache Storm: http://storm.apache.org/
[6]Apache HBase: https://hbase.apache.org/

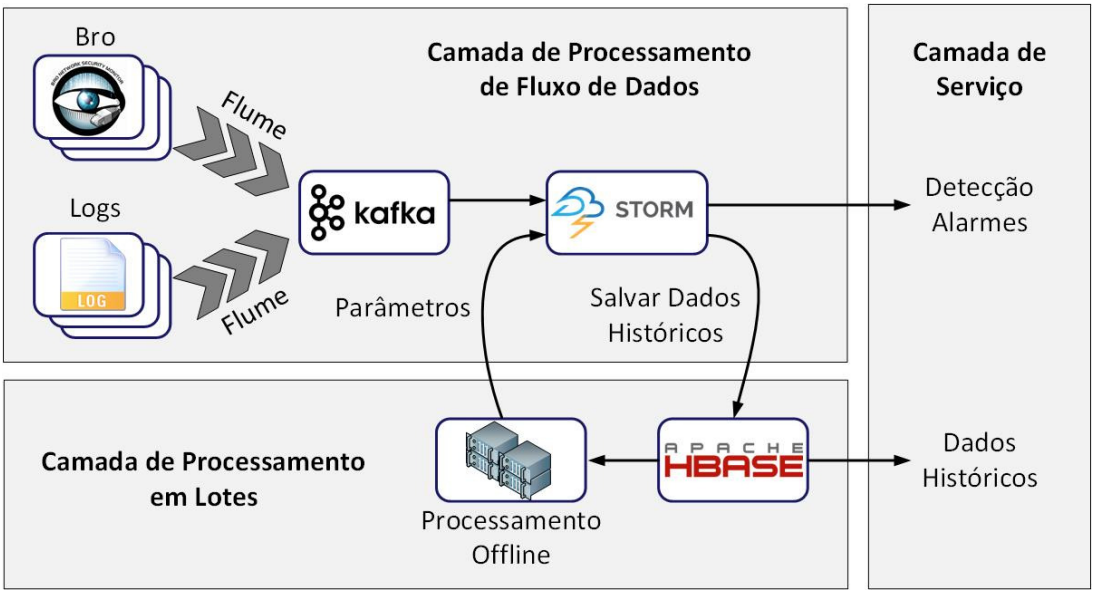



**Figura 2: Exemplo de configuração do sistema proposto para análise de dados por processamento de fluxos e processamento de lotes com uso de ferramentas de *Software* livre.**

Uma vez que as analíticas são extraídas dos dados por fluxos, os resultados são armazenados para uma análise posterior em uma base de dados dinâmica. A base de dados utilizada na proposta é a Apache `HBase`. Os dados armazenados contêm as informações obtidas da detecção e podem ser processados de maneira *off-line* a calcular parâmetros que são utilizados no modelo em tempo real. Neste ponto existe uma realimentação do sistema, já que os parâmetros calculados no processamento *off-line* com os dados históricos servem para ajustar o modelo de processamento em tempo real. Dessa forma, o sistema possui uma característica adaptativa, pois os parâmetros podem ser atualizados, se ajustando a novos padrões de uso.

## 4. Criação do Conjunto de Dados

Poucos conjuntos de dados (*datasets*) de segurança de redes são públicos para a avaliação de mecanismos de detecção de ataques. O principal motivo para a não liberação de dados de segurança é a privacidade, pois o tráfego de rede pode conter informações confidenciais. Os dois principais conjuntos de dados disponíveis são o DARPA [Lippmann et al. 2000] e o KDD 99 [Lee et al. 1999]. O DARPA foi criado com dados de tráfego (TCP/IP) e informações de sistema operacional obtidos através de uma rede de computadores simulada. Durante a coleta dos dados, também foram simulados ataques e marcados no conjunto de dados. Já o KDD 99 foi feito a partir da seleção e agrupamento de características do DARPA. Contudo, esses conjuntos de dados apresentam várias limitações [Tavallaee et al. 2009]. Uma delas é que o tráfego não corresponde ao cenário de uma rede real, por ser simulado. Além disso, há dados redundantes, o que influencia nos resultados dos algoritmos. Outro problema é que esses conjuntos de dados possuem mais de 15 anos e não representam ataques atuais [Sommer e Paxson 2010].

Uma contribuição deste trabalho é a criação de um conjunto de dados com tráfego real de rede para a avaliação da proposta[7]. Seguindo o mesmo tipo de procedimento usado na elaboração do conjunto de dados DARPA, um conjunto foi elaborado a partir da captura de pacotes em máquinas do laboratório do GTA/UFRJ, contendo tanto tráfego normal das máquinas, quanto ataques reais de segurança. Após a captura dos pacotes, as informações de fluxo foram

[7]O conjunto de dados será disponibilizado contactando os autores por *e-mail*.

obtidas por meio da extração de dados dos cabeçalhos dos pacotes, agrupadas em janelas de 2 segundos, pois dessa forma agrega-se informação suficiente para a composição dos fluxos. Os fluxos são definidos pela sequência de pacotes de um mesmo IP de origem para um mesmo IP de destino. Para cada um dos fluxos, foram obtidas 24 características a partir dos cabeçalhos TCP/IP dos pacotes de cada fluxo, como por exemplo: taxa de pacotes TCP, UDP e ICMP; quantidade de portas de destino e origem; quantidade de cada um dos *flags* TCP; entre outras. Existem duas classes de ameaças que podem ser detectadas pela análise das informações dos cabeçalhos, os ataques de negação de serviço (*Denial of Service* - DoS) e varredura de portas. Logo, diversos tipos de ataques dessas duas classes foram feitos para formar o conjunto de dados. Ao todo, o conjunto de dados contém 7 tipos de DoS e 9 tipos de varredura de portas distintos. Os ataques de DoS feitos são: inundação ICMP, *land*, *nestea*, *smurf*, inundação SYN, *teardrop* e inundação UDP. Já os tipos de varredura de portas no conjunto de dados são: *TCP SYN scan*, *TCP connect scan*, *SCTP INIT scan*, *Null scan*, *FIN scan*, *Xmas scan*, *TCP ACK scan*, *TCP Window scan* e *TCP Maimon scan*. Todas as ameaças foram realizadas com ferramentas da distribuição *Kali Linux*[8], que é voltada para a segurança de computadores. Esses ataques foram marcados no conjunto de dados através de filtros de origem e destino que permitem separar o tráfego das máquinas atacantes das demais.

Ao todo, cerca de 95 GB de dados de captura de pacotes foram coletados, resultando em 154.187 fluxos entre ataques e tráfego normal. Para a classificação, o conjunto de dados foi dividido em 70% para treino e 30% para avaliação do desempenho dos algoritmos implementados, o que faz com que as estatísticas sejam obtidas em dados novos, não analisados na parte de treino, como é comumente usado nos trabalhos científicos da área. Já para a detecção de anomalia, o treino é feito com 70% dos dados de tráfego legítimo para a determinação do comportamento normal. Os outros 30% são usados para calcular a taxa de falsos positivos e os dados de ataque são usados para calcular a taxa de detecção.

### 4.1. Seleção de Características e Análise de Componentes Principais

Para aumentar a eficiência do sistema proposto na detecção de ameaças em tempo real, optou-se pela redução da dimensionalidade. Busca-se a eliminação de algumas características irrelevantes para a classificação de ataques, tais como as características que não variam de valor ou com uma distribuição uniforme, que pouco contribuem sobre o pertencimento de uma amostra a uma classe específica. Outro aspecto que deve ser considerado é uma possível correlação entre duas ou mais características, que poderiam ser combinadas em apenas uma característica, diminuindo o tempo de processamento.

A redução de dimensionalidade é obtida pelo algoritmo de Análise de Componentes Principais (*Principal Component Analysis* - PCA), que transforma um grupo de variáveis possivelmente correlacionadas em um grupo de variáveis linearmente descorrelacionadas, que estão em planos ortogonais. Essa transformação é feita levando em conta a decomposição em autovalores, de maneira que o componente associado ao maior autovalor represente a maior variância dos dados. Os demais componentes da matriz resultante também são ordenados de acordo com a variância que eles representam. Assim, as componentes associadas aos autovalores maiores possuem mais informação relevante, fazendo com que os associados aos mais baixos possam ser retirados, reduzindo a dimensionalidade dos dados e diminuindo o tempo necessário para os algoritmos. Deve ser ressaltado que o algoritmo de análise de componentes principais não leva em conta as marcações de classe para realizar a transformação dos dados e, portanto, foi usado tanto nos algoritmos supervisionados, quanto nos algoritmos não supervisionados de aprendi-

[8]Linux Kali: https://www.kali.org/

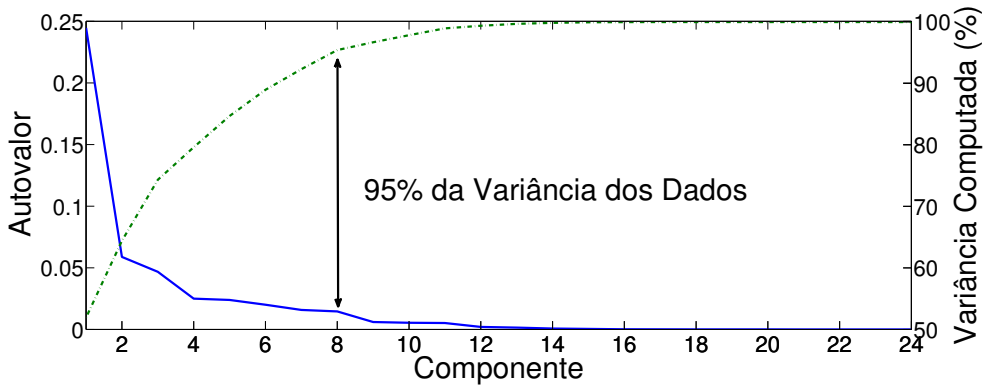



**Figura 3: Autovalor para cada uma das 24 características dos fluxos do conjunto de dados. O autovalor associado à cada característica está diretamente ligado à variância do conjunto de dados. Os oito maiores componentes principais representam 95% da variância do conjunto completo com as 24 características.**

zado de máquinas.

A Figura 3 mostra os autovalores associados ao conjunto de dados elaborado para avaliar o sistema proposto. A soma dos 8 maiores autovalores representa 95% da soma total, o que equivale a dizer que as 8 primeiras características dos dados resultantes da transformação linear calculada pela PCA representam 95% da variância total. Logo, essas 8 componentes são selecionadas e as outras componentes, que representam apenas 5% da variância, são descartadas, diminuindo assim o tempo de processamento, que é crítico em aplicações de tempo real.

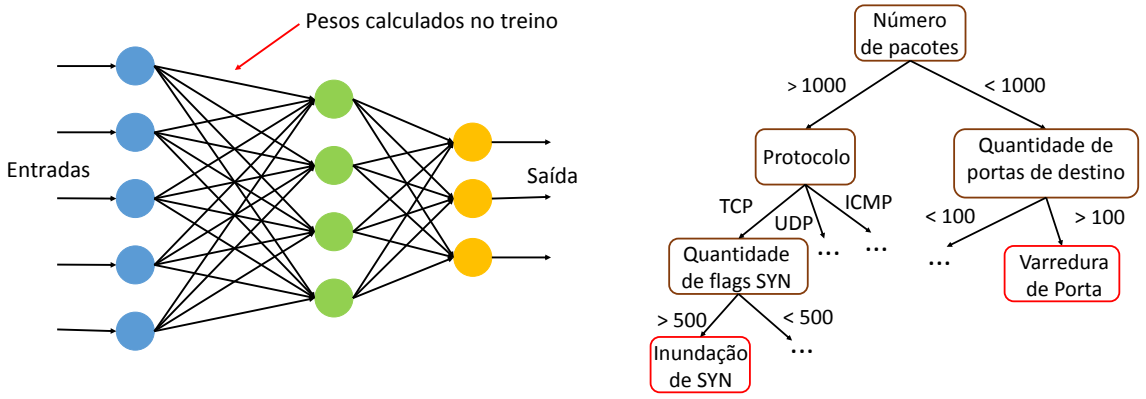
## 5. A Detecção Automática de Ameaças

Uma importante característica do sistema proposto para a obtenção da detecção em tempo real, é a eliminação completa da intervenção de especialistas de segurança para classificar e para configurar. O sistema proposto se baseia em algoritmos supervisionados de aprendizado de máquinas, como por exemplo os apresentados na Figura 4, para realizar a classificação automatizada de ameaças. Desta forma, fica sob responsabilidade dos algoritmos descobrir as características de cada classe de ataque, ao invés de ter a configuração manual como nos sistemas de segurança atuais. Além disso, a realimentação dos dados de fluxo para o processamento *off-line*, apresentada na Figura 2, permite que o sistema apresente atualizações dinâmicas, possuindo um comportamento adaptativo.

Nas Seções 5.1, 5.2 e 5.3 são apresentados os algoritmos de classificação implementados para a avaliação do sistema. Em todos os métodos são usados 70% do conjunto de dados para treino e 30% para teste e, durante a fase de treino, também é feita a validação cruzada para evitar a especialização. Na validação cruzada, partes dos dados de treino são separadas e não são usadas no cálculo dos parâmetros do modelo. Elas são usadas posteriormente para verificar se o modelo está geral o suficiente para se adaptar a novos dados, e não especializado somente para os dados de treino.

### 5.1. O Algoritmo de Árvore de Decisão

Nas árvores de decisão, as folhas representam a classe final e os ramos representam condições baseadas no valor de uma das características de entrada. Durante a fase de treino foi determinada a estrutura da árvore de classificação, usando o algoritmo de aprendizado C4.5. Na implementação do algoritmo de árvore de decisão para uso em tempo real, são usados os




**(a) Esquema de algoritmo de rede neural.** **(b) Esquema de algoritmo de árvore de decisão.**

**Figura 4: Algoritmos de classificação: a) na rede neural, cada neurônio executa uma parte do processamento b) na árvore de decisão, as características são analisadas e no fim é obtida uma classificação.**

parâmetros calculados durante a fase de treino, na qual são determinadas regras que geram um grafo semelhante ao da Figura 4b, com a diferença que as características avaliadas são as obtidas após a seleção de características com a PCA. Portanto, para a implementação no sistema em tempo real, essas regras são escritas no formato de expressões condicionais (*if-then-else*) que resultam no grafo determinado na fase de treino. Os resultados são apresentados na Seção 5.4 junto com os demais algoritmos de classificação.

## 5.2. O Algoritmo de Rede Neural

A implementação do algoritmo de rede neural segue o funcionamento do cérebro humano, em que cada neurônio é responsável por uma parte do processamento, passando o resultado para o neurônio seguinte. Na saída, é obtido um grau de pertinência a cada classe, sendo que a classe escolhida é a de maior grau. Os vetores de peso $\Theta$ são calculados durante o treino. Na Figura 4a esses pesos são representados pelas ligações entre os neurônios. Primeiro obtém-se um espaço amostral de entradas e saídas desejadas para a rede neural e posteriormente minimiza-se a parcela do erro da estimativa de cada parâmetro através do algoritmo de *Back Propagation*.

O cálculo realizado por cada camada para rede neural para determinar a classe que a amostra pertence é realizado através das seguintes equações:

$$z_{(i+1)} = \Theta_{(i)} a_{(i)} \qquad (1) \qquad a_{(i+1)} = g(z_{(i+1)}) \qquad (2) \qquad g(z) = \frac{1}{1+e^{-z}} \qquad (3)$$

onde $a_{(i)}$ é o vetor que determina a saída correspondente à camada $i$, $\Theta_{(i)}$ é o vetor de pesos que leva da camada $i$ até a camada $i+1$, e $a_{(i+1)}$ representa a saída da camada seguinte, a camada $i+1$. A função $g(z)$ é a função *Sigmoid* e ela desempenha um papel muito importante na classificação. Para valores elevados de $z$, ela vale 1 e para valores baixos, ela vale 0. Portanto, na última camada é obtida um grau de pertencimento de cada classe, entre 0 e 1, sendo a classe escolhida a de maior grau de pertencimento.

## 5.3. O Algoritmo de Máquina de Vetores de Suporte

A Máquina de Vetores de Suporte (*Support Vector Machine* - SVM) é um classificador binário baseado no conceito de planos de decisão que definem limiares de decisões. Basica-

mente, o SVM classifica através da construção de hiperplanos em um espaço multidimensional que separa casos de diferentes rótulos de classe. Um algoritmo de treinamento iterativo é usado para minimizar uma função de erro, construindo um hiperplano ótimo. O algoritmo de SVM separa os dados de treinamento em espaço de características por um hiperplano definido pelo tipo de função de *kernel* utilizado. Assim, SVM encontra o hiperplano de margem máxima, definida como a soma das distâncias do hiperplano desde o ponto mais próximo das duas classes.

A detecção em tempo real é realizada pela classificação para cada uma das classes, sendo normal e não normal; *DoS* e não *DoS*; e varredura e não varredura. Uma vez obtida a saída, é escolhida a classe que obtém a maior pontuação. A pontuação para classificar uma observação $x$, é a distância desde $x$ aos limiares de decisão variando de $-\infty$ para $+\infty$. A pontuação é computada como a resposta a função:

$$f(x) = \sum_{j=1}^{n} \alpha_j y_j G(x_j, x) + b, \tag{4}$$

onde $(\alpha_1, ..., \alpha_n.b)$ são os parâmetros estimados do SVM, e $G(x_j, x)$ é o *kernel* utilizado. Neste trabalho o *kernel* é linear, ou seja, $G(x_j, x) = x_j^{'}x$. O *kernel* linear apresenta um bom desempenho com a mínima quantidade de parâmetros de entrada.

### 5.4. Resultados da Classificação de Ataques

Os resultados obtidos de cada algoritmo supervisionado implementado são apresentados através de duas métricas: a acurácia e a matriz de confusão. A acurácia define o percentual de acerto de classificação no conjunto de dados de treino. A matriz de confusão especifica de forma clara o número de falsos positivos de cada classe. Essa matriz é uma matriz no formato Classe Real versus Classe Prevista. As linhas representam os elementos que pertencem à classe de verdade, e as colunas os elementos que foram atribuídos à classe pelo algoritmo de classificação. Portanto, os elementos na diagonal dessa matriz representam os acertos de classificação, já que os elementos pertencem à classe e foram classificados como pertencentes da mesma classe. Assim, qualquer elemento fora da diagonal representa um erro. Essa métrica foi escolhida, porque a partir dela é possível se determinar todas as outras medidas, como falso positivo e taxa de acerto de cada classe.

Os resultados mostram uma alta acurácia na classificação de ataques em cada um dos algoritmos utilizados. A Tabela 1 mostra a comparação da acurácia dos distintos algoritmos. Na tabela é possível observar que os três algoritmos utilizados no sistema apresentam uma acurácia semelhante, perto do 95%.

**Tabela 1: Comparação de Acurácia dos Distintos Algoritmos Utilizados na Classificação de Ataques.**

| | Árvore de Decisão | Rede Neural | Máquina de Vetores de Suporte |
|---|---|---|---|
| **Acurácia** | 95.9984% | 95.1271% | 95.8103% |

As Tabelas 2, 3 e 4 mostram as matrizes de confusão dos algoritmos aplicados na classificação de ataques. Na diagonal principal da matriz de confusão observa-se os acertos em cada uma das classes. Assim, é analisada com maior profundidade a acurácia de cada classificador mostrada na Tabela 1. Tanto a Tabela 2 como a Tabela 4 mostram o melhor desempenho, já que elas apresentam um menor número de Falsos Positivos (FP).

**Tabela 2: Matriz de Confusão da Árvore de Decisão**

| | Normal | DoS | Varredura |
|---|---|---|---|
| **Normal** | 29126 | 1 | 0 |
| **DoS** | 60 | 5845 | 0 |
| **Varredura** | 8 | 1782 | 9434 |

**Tabela 3: Matriz de Confusão da Rede Neural**

| | Normal | DoS | Varredura |
|---|---|---|---|
| **Normal** | 28807 | 253 | 67 |
| **DoS** | 95 | 5810 | 0 |
| **Varredura** | 1839 | 0 | 9385 |

**Tabela 4: Matriz de Confusão da Máquina de Vetores de Suporte**

| | Normal | DoS | Varredura |
|---|---|---|---|
| **Normal** | 29087 | 0 | 40 |
| **DoS** | 63 | 5842 | 0 |
| **Varredura** | 1835 | 0 | 9389 |

### 5.5. Detecção de Anomalias pela Distribuição Gaussiana

Ataques novos (*zero-day attacks*) são muito difíceis de serem detectados, pois não há dados ainda sobre como são realizados. Por isso, a proteção contra ataques desconhecidos é essencial para um maior nível da segurança de redes de computadores. A detecção de anomalias consegue assim descobrir ataques novos.

Neste artigo, a detecção de anomalias é proposta pelo distanciamento de uma distribuição gaussiana. Logo, as anomalias são detectadas a partir da média e da variância de cada característica do conjunto de dados de treino com amostras normais. Assim, as anomalias ocorrem quando é detectada uma distância para média maior do que o valor de um limiar vezes a variância em pelo menos uma das características. São analisadas as 8 características obtidas pela transformação linear da PCA.

A implementação em tempo real requer a detecção de anomalia do fluxo de dados à medida que os dados vão chegando no sistema. A anomalia é detectada se pelo menos uma das seguintes desigualdades for verdadeira para qualquer característica $j$, levando em conta as médias $\mu_j$ e as variâncias $\sigma_j^2$ calculadas no treino:

$$X_j > \mu_j + limiar * \sigma_j^2 \qquad (5) \qquad\qquad X_j < \mu_j - limiar * \sigma_j^2 \qquad (6)$$

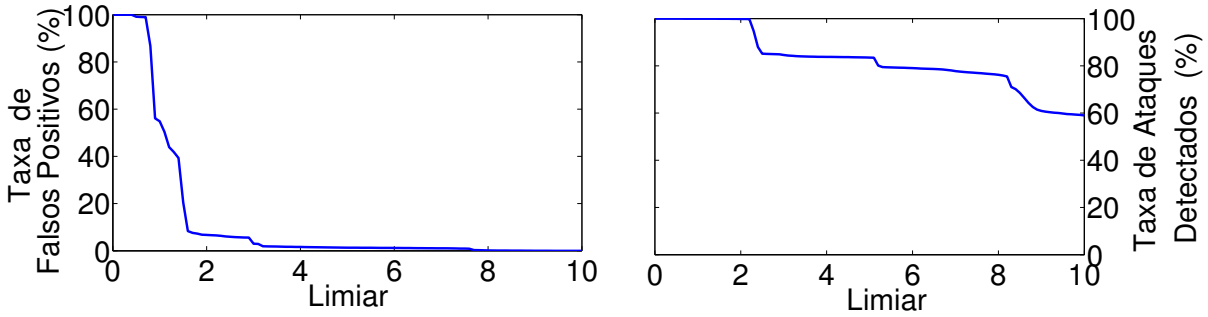
O sistema proposto permite o treino em tempo real devido à arquitetura lambda. Assim, o algoritmo torna-se adaptativo, o que é fundamental para a detecção de anomalias, pois o comportamento da rede pode mudar com o tempo. Logo, quando um fluxo novo chega ao sistema e ele não é detectado como anomalia pelas desigualdades (5) e (6), os parâmetros $\mu_j$ e $\sigma_j^2$ de cada característica são adaptados, considerando esse novo fluxo. Os parâmetros de uma distribuição normal são calculados da seguinte forma:

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} X_j \qquad (7) \qquad\qquad \sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(X_j - \mu_j)^2 \qquad (8)$$

Portanto, os valores correntes dos somatórios e da quantidade de amostras $N$ são armazenados no sistema e incrementados quando um novo fluxo chega ao sistema, considerando cada característica $X_j$ do fluxo. Assim, os parâmetros da gaussiana são sempre atualizados de acordo com os fluxos considerados benignos, garantindo adaptabilidade na detecção de anomalia.

Os resultados são mostrados na Figura 5 para diferentes valores da variável $limiar$. Os

resultados de taxa de falsos positivos foram obtidos ao se avaliar o conjunto de dados *dataset* de treino normal e os de taxa de detecção de ataque com todos os ataques que estão no conjunto de dados. Ao se escolher um valor baixo de limiar, quase todos os ataques são detectados, porém o número de falsos positivos também é maior. Já um valor elevado resulta em menos falsos positivos, mas também uma taxa de detecção menor. Com $limiar$ igual a 2, a taxa de falsos positivos é de 6,7% e a de detecção de 99,8%. Com $limiar$ igual a 3, essas taxas são de 3,0% e de 84,5%. Esse parâmetro então deve ser escolhido de acordo com a aplicação, levando em conta seus requisitos de segurança e o quanto de falsos positivos é aceitável.




**(a) Taxa de Falsos Positivos.** **(b) Taxa de Detecção de Ataques.**

**Figura 5: Taxa a) de falsos positivos e b) de detecção de ataques em função do limiar em número de variância. Quanto menor o limiar, mais ataques são detectados, porém maior é a taxa de falsos positivos.**

### 5.6. Desempenho em Tempo Real

O tempo de detecção de uma ameaça é crucial para manter a segurança em redes. Se a detecção demorar demais, não há qualquer reação que possa ser tomada para neutralizar a ameaça. Através do processamento de fluxos e da arquitetura lambda, é possível usar os parâmetros de um treino prévio para a detecção em tempo real. O protótipo do sistema foi implementado num ambiente com 4 máquinas virtuais para testar o desempenho em tempo real. As máquinas virtuais executavam sobre uma máquina com processador Intel Core i7-2600 e com 16 GB de memória RAM. A medida de desempenho foi feita com a implementação da rede neural, pois ela apresenta a maior complexidade computacional entre os algoritmos implementados. O sistema foi capaz de processar em média 3,57 milhões de fluxos por minuto com incerteza de 156 mil, medidas com intervalo de confiança de 95%. Portanto, o tempo de detecção de cada ataque é cerca de 17 microssegundos o que permite tentativas de defesas eficazes diminuindo o risco do ataque.

## 6. Conclusão

Este artigo propõe um sistema de detecção de ameaças em tempo real por processamento de fluxos. O sistema proposto usa a arquitetura lambda, que combina os paradigmas de processamento em fluxos e em lotes. O processamento em lotes, em tempo diferenciado, permite uma adaptação automática dos algoritmos de aprendizado de máquinas para serem usados em tempo real. Para avaliar o desempenho do sistema proposto, um conjunto de dados baseado em tráfego real de rede foi criado para a avaliação da proposta. Os pacotes capturados foram separados em fluxos rotulados através de 24 características. Para aumentar a eficiência do sistema a dimensionalidade do conjunto de dados foi reduzida através do algoritmo de análise de componentes principais. Foram utilizados os algoritmos de aprendizado de máquinas árvore de decisão, rede neural e máquina de vetores de suporte para a classificação em tempo real dos ataques. Os

resultados mostram uma acurada classificação de ataques dos três algoritmos com valores maiores que 95%, sendo o algoritmo de árvore de decisão foi o de melhor acurácia com 95,99%. Os algoritmos implementados também apresentaram uma baixa taxa de falsos positivos. Além disso, foi implementado um algoritmo de detecção de anomalias para detectar ataques desconhecidos que apresentou um bom compromisso entre as taxas de falsos positivos e detecção de ataques de acordo com o limiar escolhido. Por exemplo, pode-se obter uma taxa de detecção de ataques de 99,8% com 6,7% de taxa de falso positivo ou, com um limiar diferente, uma taxa de detecção de 84,5% e apenas 3,0% de falsos positivos. O processo de detecção de anomalias é adaptativo, já que os parâmetros são atualizados em tempo real. O protótipo implementado utiliza ferramentas de processamento de fluxos, como o Apache `Storm` e o Apache `Kafka` e, por isso, consegue um baixo tempo na detecção das ameaças, possibilitando estratégias de defesa. O sistema apresenta um bom desempenho em tempo real analisando até 3,57 milhões de fluxos por minuto.

A implementação do sistema proposto como serviço em nuvem será realizada em um trabalho futuro. Esse serviço deve apresentar uma característica elástica no fornecimento dos recursos. O desenvolvimento de detecção de outros tipos de ataques, como ataques na camada de aplicação também será um trabalho futuro.

## Referências


Amini, M., Jalili, R. e Shahriari, H. R. (2006). RT-UNNID: A practical solution to real-time network-based intrusion detection using unsupervised neural networks. *Computers & Security*, 25(6):459 – 468.

Andreoni Lopez, M., Figueiredo, U. d. R., Lobato, A. G. P. e Duarte, O. C. M. B. (2014). BroFlow: Um sistema eficiente de detecção e prevenção de intrusão em redes definidas por software. *XII WPerformance (XXXIV CSBC)*, páginas 1919–1932.

Buczak, A. e Guven, E. (2015). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys Tutorials*, (99):1–26.

Clay, P. (2015). A modern threat response framework. *Network Security*, 2015(4):5–10.

Costa, L. H. M. K., de Amorim, M. D., Campista, M. E. M., Rubinstein, M., Florissi, P. e Duarte, O. C. M. B. (2012). Grandes massas de dados na nuvem: Desafios e técnicas para inovação. Em *SBRC 2012 - Minicursos*.

Du, Y., Liu, J., Liu, F. e Chen, L. (2014). A real-time anomalies detection system based on streaming technology. Em *Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, volume 2, páginas 275–279. IEEE.

Holtz, M. D., David, B. M. e de Sousa Júnior, R. T. (2011). Building scalable distributed intrusion detection systems based on the mapreduce framework. *Revista Telecommun*, 13(2).

Lee, W., Stolfo, S. J. e Mok, K. W. (1999). Mining in a data-flow environment: Experience in network intrusion detection. Em *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 114–124. ACM.

Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K. et al. (2000). Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. Em *Proceedings of DARPA Information Survivability Conference and Exposition. DISCEX'00.*, volume 2, páginas 12–26. IEEE.

Liu, Q., Lui, J., He, C., Pan, L., Fan, W. e Shi, Y. (2014). SAND: A fault-tolerant streaming architecture for network traffic analytics. Em *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, páginas 80–87. IEEE.

Lobato, A. G. P., da Rocha Figueiredo, U., Andreoni Lopez, M. e Duarte, O. C. M. B. (2014). Uma arquitetura elástica para prevenção de intrusao em redes virtuais usando redes definidas por software. *Anais do XXXII SBRC 2014*, páginas 427–440.

Marz, N. e Warren, J. (2013). *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning Publications Co., Greenwich, CT, USA, 1st edition.

Ponemon, I. e IBM (2015). 2015 cost of data breach study: Global analysis. `www.ibm.com/security/data-breach/`. Acessado: 27/12/2015.

Porto, F. e Ziviani, A. (2014). Ciência de dados. Em *3o. Seminário de Grandes Desafios da Computação no Brasil*. SBC.

Rai, K. e Devi, M. S. (2013). Intrusion detection systems: A review. *Journal of Network and Information Security Volume*, 1(2).

Ringberg, H., Soule, A. e Rexford, J. (2008). Webclass: adding rigor to manual labeling of traffic anomalies. *ACM SIGCOMM Computer Communication Review*, 38(1):35–38.

Rychly, M., Koda, P. e Smrz, P. (2014). Scheduling decisions in stream processing on heterogeneous clusters. Em *Eighth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, páginas 614–619.

Sangkatsanee, P., Wattanapongsakorn, N. e Charnsripinyo, C. (2011). Practical real-time intrusion detection using machine learning approaches. *Computer Communications*, 34(18):2227 – 2235.

Solaimani, M., Khan, L. e Thuraisingham, B. (2014). Real-time anomaly detection over VMware performance data using storm. Em *IEEE 15th International Conference on Information Reuse and Integration (IRI)*, páginas 458–465. IEEE.

Sommer, R. e Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. Em *IEEE Symposium on Security and Privacy (SP)*, páginas 305–316. IEEE.

Sperotto, A., Sadre, R., Van Vliet, F. e Pras, A. (2009). A labeled data set for flow-based intrusion detection. Em *IP Operations and Management*, páginas 39–50. Springer.

Stroeh, K., Madeira, E. R. M. e Goldenstein, S. K. (2013). An approach to the correlation of security events based on machine learning techniques. *Journal of Internet Services and Applications*, 4(1):1–16.

Tavallaee, M., Bagheri, E., Lu, W. e Ghorbani, A.-A. (2009). A detailed analysis of the KDD CUP 99 data set. Em *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications*. IEEE.

Wu, K., Zhang, K., Fan, W., Edwards, A. e Yu, P. S. (2014). RS-Forest: A rapid density estimator for streaming anomaly detection. Em *IEEE International Conference on Data Mining (ICDM)*, páginas 600–609.

Zhao, S., Chandrashekar, M., Lee, Y. e Medhi, D. (2015). Real-time network anomaly detection system using machine learning. Em *11th International Conference on the Design of Reliable Communication Networks (DRCN)*, páginas 267–270. IEEE.