Uma proposta de Sistema de Detecção de Intrusão em Redes de Computadores Utilizando Vetores de Conexão e Máquinas de Vetores de Suporte (SVM)

Heitor Scalco Neto[1], **Wilian Soares Lacerda**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Lavras (UFLA)
Lavras – MG – Brazil

heitorscalco@hotmail.com, lacerda@dcc.ufla.br

Abstract. *This paper present an metodology to build an Network Intrusion Detection System (NIDS), using Support Vector Machines (SVM) and processing the data like Connection Vectors. To transform raw packets in Connection Vectors we proposed an framework, Open-source. The results obtained with the utilization of Support Vector Machines (SVM) were quite satisfactory, obtaining a mean score around* ***XX.XX%****, thus affirming the effectiveness of the proposed methodology.*

Resumo. *Este artigo apresenta uma metodologia para a construção de um Sistema de Detecção de Intrusão em Redes de Computadores, utilizando Máquinas de Vetores de Suporte (SVM) e realizando a formatação dos dados em Vetores de Conexão. Para transformar pacotes de rede em Vetores de Conexão foi proposto um framework, Open-source. Os resultados obtidos com a utilização do método e Máquinas de Vetores de Suporte foram bastante satisfatórios, obtendo-se uma média de acertos em torno de* ***XX.XX%****, afirmando assim a eficácia da metodologia proposta.*

1. Introdução

Este artigo propõe um método para a detecção de ataques em redes de computadores com a implementação de Máquinas de Vetores de Suporte (SVM) para classificar o tráfego normal e anômalo. De acordo com Salem et al. 2014, um método que torna um Sistema de Detecção de Intrusão em Redes (NIDS) mais eficaz é a utilização de Vetores de Conexão, o qual faz com que um fluxo inteiro de dados seja analisado, ao invés de pacotes separadamente. Esta metodologia auxilia na detecção de ataques, principalmente de negação de serviço, o qual representa uma parte significativa dos ataques reportados ao CERT.BR 2016.

Para que seja possível obter os vetores de conexão, faz-se necessário o desenvolvimento de um *framework* que extrai as informações relevantes dos pacotes de rede e as transforma em vetores de conexão. Após realizar o pré-processamento e a captura de todos os pacotes pertencentes ao fluxo, o *framework* encaminha os vetores de conexão para o motor de classificação (Máquinas de Vetores de Suporte - SVM), conforme será apresentado no decorrer das próximas seções.

O artigo está organizado da seguinte forma: na seção 2 é apresentada a revisão bibliográfica e na seção 3 são apresentados os materiais e métodos utilizados. O fechamento do artigo se dá pelas seções 4 e 5, as quais apresentam os resultados e discussões oriundas deste trabalho, juntamente com as devidas conclusões.

2. Revisão Bibliográfica

Esta seção apresenta, sucintamente, uma revisão bibliográfica sobre Sistemas de Detecção de Intrusão (Seção 2.1) e Máquinas de Vetores de Suporte (Seção 2.2) . A seção foi subdividida para melhor compreensão.

2.1. Sistemas de Detecção de Intrusão

A implementação de tecnologias anti-maliciosas nas redes de computadores são muito importantes para a manutenção da segurança dessas redes. A utilização de Sistemas de Detecção de Intrusão é bem-vinda para que seja possível realizar o monitoramento e análise (através da implementação de *honeypots* [1]) do tráfego que ingressa na rede, passando pelo *firewall* [2] [Wang 2009]. Este processo é de extrema importância, pois permite que o administrador da rede consiga estabelecer regras mais concisas nos ativos de segurança.

O termo Detecção de Intrusão define-se como o processo de monitorar eventos que estão ocorrendo em um sistema computacional ou em uma rede de computadores, buscando tráfego intrusivo. Incidentes de segurança possuem várias causas, como a propagação de um *malware* (ex.: *worms, spyware, trojan*), atacantes tentando elevar privilégios para acessar sistemas não autorizados, entre outros [Karen Scarfone 2007]. Dessa forma, uma tecnologia tornou-se aliada aos administradores de rede e segurança, são os Sistemas de detecção de Intrusão (IDS), sua função é reconhecer um comportamento anômalo ou uma ação intrusiva e reportar ao administrador da rede para que as medidas necessárias sejam tomadas [Laureano et al. 2003].

A forma na qual os Sistemas de Detecção de Intrusão (IDS) reconhecem uma ação intrusiva pode ser baseada em assinaturas ou baseado em anomalias. Sistemas de detecção baseados em assinaturas identificam ataques através da análise de assinaturas, previamente configuradas, sobre o comportamento padrão de algum tipo de ataque. No caso do IDS baseado em anomalias, é analisado o tráfego da rede, classificando o tráfego em anômalo ou normal [de Azevedo 2012]. A principal vantagem de um IDS baseado em anomalia é que é possível detectar ataques desconhecidos, o que não acontece na detecção por assinatura [Wang 2009]. Os modos de reconhecimento de intrusão serão apresentados mais detalhadamente a seguir:

- **Sistemas de Detecção baseados em assinaturas, ou abuso**, são bastante utilizados em virtude do baixo custo computacional exigido. A detecção de intrusões é feita através de regras, bem definidas, geralmente obtidas através de *honeypots*, onde o tráfego é analisado e comparado com um ataque. Dessa forma, o IDS é bastante eficiente e rápido, porém, é incapaz de identificar novos ataques [Wang 2009].
- **Sistemas de Detecção baseados em anomalia** fazem a supervisão do comportamento do sistema, onde assume-se que atividades anormais são classificadas como intrusões [SILVA 2011]. Este método pode ser implementado com várias técnicas, entre elas Inteligência Computacional [Sen et al. 2014].

[1] Sistema para capturar metodologias de ataque.

[2] Dispositivo para bloquear/liberar determinado tipo de tráfego.

2.1.1. Classificação dos Sistemas de detecção de Intrusão

De acordo com IATAC 2009 e Wang 2009, Sistemas de detecção de intrusão podem ser classificados em três diferentes categorias: IDS baseado em *hosts* (*HIDS - Host Based Intrusion Detection System*), IDS baseado em redes (*NIDS - Network Intrusion Detection System*) e ainda IDS híbrido. Um HIDS analisa o comportamento de um *host* em específico, geralmente através de informações obtidas por SNMP [3]. Um NIDS analisa somente o tráfego de rede (é o modelo mais utilizado). Por fim, o IDS híbrido faz a união das características do HIDS e NIDS [de Azevedo 2012].

Um Sistema de Detecção de Intrusão em Redes (NIDS) trabalha analisando o tráfego de rede, geralmente utilizando uma interface em modo promíscuo, funcionando como um *sniffer* [4] [Uchoa 2009]. Esse tipo de sistema geralmente atua com um ou mais sensores na rede e uma estação de monitoramento. Quando um sensor detecta uma atividade anormal na rede, um alerta é transmitido para a estação de monitoramento que informará ao administrador da rede sobre a situação. A Figura 1 apresenta como deve ser realizada a implementação de um NIDS na rede [Scalco Neto 2015].

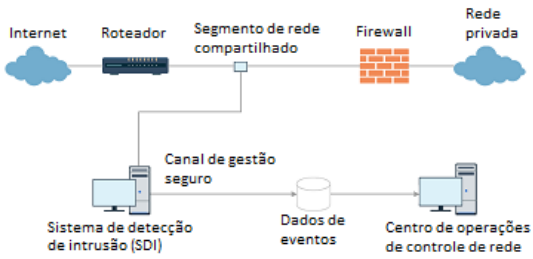

Figura 1. Posicionamento de um NIDS na rede [Scalco Neto 2015]

As principais vantagens da utilização de um NIDS são: baixo custo, possui boa taxa de acertos para uma série de ataques, não requer alterações em servidores e/ou *hosts* em produção, não causa gargalo ou indisponibilidade na rede (pois é um equipamento que funciona de forma passiva). Algumas desvantagens também podem ser citadas: incapacidade de detectar ataques oriundos de protocolos criptografados (SSL, IPsec, SSH) e ataques fragmentados [Wang 2009].

Como exemplo de NIDS, utilizando tanto o método por assinaturas quanto por anomalias, é possível citar o *Snort* (HIDS e NIDS) [SNORT.ORG 2016]. Um método para o reconhecimento de intrusão em redes de computadores pode ser a utilização de técnicas de inteligência computacional, como Máquinas de Vetores de Suporte (SVM). A técnica de SVM é descrita na próxima seção.

2.2. Máquinas de Vetores de Suporte

De acordo com Rodrigues et al. 2007, *Support Vector Machines* (SVM) ou Máquinas de Vetores de Suporte, é uma técnica utilizada para o treinamento de classificadores, baseado no conceito da minimização do risco estrutural. Essa técnica foi proposta através de

[3] Protocolo de aplicação para gerenciamento de redes de computadores
[4] Ferramenta de captura de tráfego de rede

fundamentos na teoria de aprendizagem estatística [LIMA 2014]. Nota-se um crescente interesse por pesquisadores, desde a década de 90, em problemas de reconhecimento de padrões [Rodrigues et al. 2007], tais como detecção de intrusão.

Máquinas de Vetores de Suporte destacam-se por duas características: possuem uma fundamentação teórica plausível e podem alcançar alto desempenho em certas aplicações [dos Santos 2002], também tem um grande poder de generalização [Rodrigues et al. 2007]. De acordo com [Jalil et al. 2010], em aplicações nas quais exijam precisão em classificação ou regressão, SVM é um método eficaz para tarefas de aprendizagem de máquina.

De acordo com Rodrigues et al. 2007, SVMs são classificadores lineares que tem como objetivo separar os dados em duas ou mais classes, através da criação de um hiperplano de separação. Um hiperplano ótimo separa os dados de forma que a margem seja a maior possível, sendo que a mesma é definida através da soma das distâncias entre os pontos positivos e negativos mais próximos do hiperplano, o qual é criado através de um conjunto finito de dados de treinamento. Esses pontos compõe os vetores de suporte (pontos demarcados na Figura 2) [Rodrigues et al. 2007].

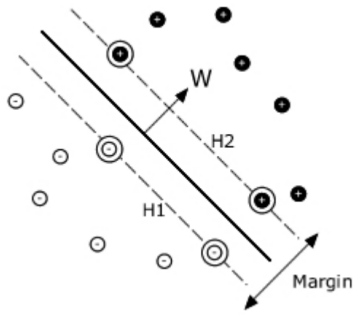

Figura 2. Classificação de um conjunto de dados utilizando SVM linear [Rodrigues et al. 2007]

Algumas propostas utilizando Máquinas de Vetores de Suporte em Sistemas de Detecção de Intrusão são encontradas na literatura, tais como: Jha and Ragha 2013, Mulay et al. 2010, Rao et al. 2003 e Li et al. 2003. Estas propostas afirmam a viabilidade da implementação de SVM para solucionar problemas de detecção de intrusão, porém não são aplicadas a um NIDS.

2.2.1. Tipos de Kernel de SVM

De acordo com Ben-Hur and Weston 2010, um *kernel* é um algoritmo que depende do produto escalar de um determinado conjunto de dados. Em outras palavras, um *kernel* é uma função de similaridade que é utilizada, dependendo do conjunto de dados, juntamente com o algoritmo de aprendizado.

Em muitos casos classificadores (*kernels*) não lineares proporcionam melhores taxas de acertos. Porém, conforme comentado em Hsu et al. 2003, em alguns casos específicos um classificador linear pode se sobressair, são eles:

- Número de amostras << Número de Características;
- Número de amostras e Número de Características muito grandes;
- Número de amostras >> Número de Características.

Os tipos de *kernel* mais comumente utilizados, de acordo com [Hsu et al. 2003], são o linear, polinomial, RBF e sigmóide. Hsu et al. 2003 recomenda fortemente que o treinamento inicial do SVM seja realizado com *kernel* linear, o qual pode trazer algumas vantagens, tais como a economia de tempo e simplicidade do modelo. Caso o aprendizado, utilizando *kernel* linear, não resulte em um classificador eficaz, aconselha-se a tentativa com outros modelos de *kernel*.

Para que o ajuste (treinamento) do classificador (SVM) seja realizado, faz-se necessário a utilização de alguma biblioteca que implemente o algoritmo. Para este artigo, a biblioteca LibSVM foi escolhida, a qual será apresentada na próxima seção.

2.2.2. Biblioteca LibSVM

É uma biblioteca desenvolvida desde 2000 e tem sido a biblioteca para SVMs mais utilizada nos últimos 10 anos. O principal objetivo é facilitar a implementação de Máquinas de Vetores de Suporte em aplicações reais. LibSVM tem compatibilidade com várias linguagens, entre elas: Python, JAVA, Matlab, Perl, Ruby, Weka, C#, PHP, entre outras [Chang and Lin 2011].

3. Metodologia

Esta seção descreve a metodologia utilizada para a realização deste trabalho, abordando as técnicas e equipamentos necessários para o desenvolvimento de um Sistema de Detecção de Intrusão em Redes de Computadores (NIDS) utilizando Máquinas de Vetores de Suporte (SVM) e Vetores de Conexão.

3.1. Base de Dados ISCX 2012

Várias bases de dados com tráfego de rede já foram propostas, porém a base de dados ISCX 2012 teve como objetivo suprir algumas deficiências encontradas nas bases anteriormente propostas (CAIDA, DARPA e KDD) [Shiravi et al. 2012]. Onde, no caso da CAIDA, foram retirados todos os *payloads* dos pacotes e não há registros sobre quais conexões são normais ou intrusivas. Já com as bases da DARPA e KDD, de acordo com Uchoa 2009, não é possível reproduzir um ambiente de rede atual. Contudo, a base de dados ISCX 2012 oferece os registros das conexões (normais ou intrusivas), todo o tráfego capturado (sem remoção dos *payloads*), além da captura ser realizada em ambiente real. Um resumo das características de cada base de dados é apresentada na Tabela 1.

A capacidade de reprodução dos pacotes coletados pelos autores da ISCX 2012 motivou a escolha desta base de dados para a proposta deste trabalho. A base de dados conta com o tráfego capturado durante 1 semana completa, totalizando 2.450.324 conexões. Durante a captura do tráfego diversos serviços foram inicializados, tais como: FTP, HTTP, HTTPS, DNS, Netbios, POP3, SMTP, SNMP, SSH, Messenger e outros.

Para representar os ataques foram criados 4 cenários, os quais foram reproduzidos separadamente, são eles [Shiravi et al. 2012]:

Tabela 1. Comparação das Características das Bases de Dados. Modificado de: [Shiravi et al. 2012]

	Config. realista da Rede	Tráfego Realístico	Classif. das Conexões	Arq. de Captura Completo	Diversos cenários de Ataque
CAIDA	Sim	Sim	Não	Não	Não
Internet Traffic Archive	Sim	Sim	Não	Não	Não
LBNL	Sim	Sim	Não	Não	Não
DARPA-99	Sim	Não	Sim	Sim	Sim
KDD-99	Sim	Não	Sim	Sim	Sim
DEFCON	Não	Não	Não	Sim	Sim
ISCX 2012	Sim	Sim	Sim	Sim	Sim

- **Exploração de dentro da rede:** Consiste em explorar vulnerabilidades em alvos que estão na mesma rede e que estão executando alguma aplicação em específico;
- **Negação de Serviço HTTP (DoS):** Consiste na inundação de pacotes dos mais diversos tipos, com objetivo de prejudicar o correto funcionamento de serviços como o HTTP (ou qualquer outro);
- **Negação de Serviço usando Botnet (DDoS):** Também consiste em inundação de pacotes, porém de forma distribuída (utilizando uma *Botnet* [5]));
- **Força Bruta SSH:** Ataque que visa a descoberta da senha do serviço de SSH por meio de força bruta.

A base de dados da ISCX 2012 será utilizada para realizar o treinamento do SVM, fazendo com que a técnica de Inteligência Computacional reconheça padrões de tráfegos normais e anômalos e, após o treinamento, tenha a capacidade de classificar novos tráfegos de rede de forma *online*. Para que esse processo seja possível, é necessário realizar o processamento da base de dados, o qual será discutido na próxima seção.

3.2. *Framework* para Tratamento de Pacotes e Conexões

Para que seja possível a extração de características de pacotes de rede brutos, assim como são apresentados na base de dados ISCX 2012 [Shiravi et al. 2012], faz-se necessário o desenvolvimento de uma aplicação que captura os pacotes e realiza o pré-processamento, transformando um conjunto de pacotes de um mesmo fluxo em vetores de conexão. Desta forma, um *framework*, *Open-source*, foi proposto. Cabe ressaltar que a união do *framework*, o motor de classificação (Técnica de Inteligência Computacional) e o módulo de notificação, tem como objetivo formar um NIDS.

O desenvolvimento de um *framework* torna possível o pré-processamento de bases de dados existentes, a criação de novas bases de dados e a classificação *online* do tráfego. O *framework* será disponibilizado para a comunidade (*Open-source*), com o objetivo de permitir que outros pesquisadores consigam gerar as suas próprias bases de dados, de forma simplificada e, principalmente, com a capacidade de alterar os parâmetros e a extração de características dos pacotes a qualquer momento. Outro objetivo é fazer com que pesquisadores possam testar este método com diferentes motores de classificação,

[5] Vários computadores *zumbis* realizando uma tarefa específica

sem precisar alterar o *framework*. Algumas propostas semelhantes foram analisadas ([Salem et al. 2014] e [Salem and Buehler 2013]), porém, a indisponibilidade do código-fonte e algumas limitações, motivaram o desenvolvimento de um *framework* deste tipo.

Um fluxo de dados, ou uma conexão — Esta definição não tem relação com o conceito de conexão do protocolo TCP — é definido por um *Unique_id* , o qual é composto por protocolo, endereço IP de origem, porta de origem, endereço IP de destino e porta de destino. Alguns exemplos da formatação do *Unique_id* são mostrados a seguir:

- *TCP-177.105.60.1:5800-177.60.20.30:80*;
- *UDP-177.105.60.1:44000-177.60.23.31:6505*;

Desta forma, todos os pacotes que tiverem as combinações apresentadas nos tópicos a seguir, serão adicionados em suas respectivas conexões. Caso ainda não exista um *Unique_id* para a conexão, uma chave é criada.

- {***PROTOCOLO - IP_DE_ORIGEM:PORTA_DE_ORIGEM - IP_DE_DESTINO:PORTA_DE_DESTINO***};
- {***PROTOCOLO - IP_DE_DESTINO:PORTA_DE_DESTINO - IP_DE_ORIGEM:PORTA_DE_ORIGEM***};
- {***ICMP (PROTOCOLO) - IP_DE_DESTINO - IP_DE_ORIGEM - ICMP_ID***}.

Cabe ressaltar que apenas alguns campos do cabeçalho do pacote são armazenados na conexão, isso evita a sobrecarga do sistema. No caso do protocolo ICMP, onde o cabeçalho não faz a utilização de portas, os campos *PORTA_DE_ORIGEM* e *PORTA_DE_DESTINO* foram substituídos pelo campo ICMP_ID, presente no cabeçalho do ICMP e que possibilita a identificação de um determinado fluxo ICMP.

Cada conexão é tratada de forma independente, e é importante resolver alguns impasses para que seja possível estabelecer corretamente características como, por exemplo, a quantidade de *bytes* trafegados da origem para o destino e vice-versa. Caso a direção da conexão não for definida de forma correta, os valores podem ficar invertidos, prejudicando a qualidade dos dados. A Tabela 2 apresenta um método para a definição do sentido da conexão, baseado no primeiro pacote recebido em relação a cada protocolo.

Tabela 2. Definição do sentido da Conexão (Modificado de: [Salem and Buehler 2013])

Primeiro Pacote Recebido	**Ação**
Flag SYN Ativada	O *host* que enviou o pacote é a Origem
Flag SYN + ACK Ativada	O *host* que enviou o pacote é o Destino
Flag ACK Ativada e *Payload* vazio	Último pacote do *3-way-handshake*, o *host* que enviou o pacote é a Origem
UDP ou ICMP	O *host* que enviou o pacote é a Origem
Nenhuma das situações, com *Payload* preenchido	Quem tiver o maior número de porta é a Origem

Outro fator é a definição das *flags* do estado da conexão TCP. Este é um parâmetro importante para identificar, por exemplo, um ataque de inundação de pacotes com *flag* SYN (*SYN Flood*), em que o estado da conexão será definido como *handshake*. Os estados de conexão foram definidos de uma forma simplificada, em relação a apresentada

em Bing et al. 2009, na qual são definidos, de forma ordenada, 5 estados de conexão: *Handshake*, *Established*, *Termination*, *Closed* e *Unknown*. Cabe ressaltar que o estado de conexão segue um fluxo contínuo, ou seja, uma conexão nunca passará de *Established* para *Handshake*, por exemplo. No caso dos protocolos UDP e ICMP, os estados de conexão iniciam e terminam como *Closed* (já que não são protocolos orientados a conexão). As *Flags* dos estados de conexão são apresentadas na Tabela 3.

Tabela 3. *Flags* do estado da conexão (Modificado de: [Bing et al. 2009])

Flag	**Situação**
S0	Tentativa de Conexão, porém ainda sem resposta (SYN)
S1	Conexão estabelecida, ainda ativa
S2	Conexão estabelecida e requisição para fechamento pela Origem, porém ainda sem resposta do Destino
S3	Conexão estabelecida e requisição para fechamento pelo Destino, porém ainda sem resposta da Origem
SF	Conexão normalmente estabelecida e fechada
RSTO	Conexão estabelecida, porém a Origem fechou a conexão enviando um RST
RSTR	Conexão estabelecida, porém o Destino fechou a conexão enviando um RST
RSTOS0	Origem enviou um SYN seguido de um RST, nunca recebeu um SYN+ACK do Destino
RSTRH	Destino enviou um SYN+ACK seguido de um RST
SH	Origem enviou um SYN seguido de um FIN, nunca recebeu um SYN+ACK do Destino
SHR	Destino enviou um SYN+ACK seguido de um FIN
OTH	Outro tipo de tráfego

Para que uma conexão que não foi terminada possa ser analisada pelo NIDS, faz-se necessário o estabelecimento de *timeouts*. Dessa forma, conexões que foram perdidas ao longo do tráfego, ou até mesmo conexões não finalizadas propositalmente [6], poderão ser analisadas em um tempo pré-definido. Os *timeouts* foram definidos em relação ao último pacote de cada conexão e também ao estado em que a conexão encontra-se. Os valores são apresentados na Tabela 4. A implementação de valores de *timeouts* altos (tais como os apresentados na Tabela 4) faz com que o NIDS torne-se menos eficiente, tendo como resultado um atraso de alguns segundos na classificação. É importante afirmar que os valores foram escolhidos em virtude do embasamento teórico apresentado por NIDS já conhecidos [SNORT.ORG 2016], e também utilizados nas propostas de Salem et al. 2014 e Salem and Buehler 2013, ficando a critério do usuário do *framework* a definição de outros valores.

Tabela 4. *Timeouts* (Modificado de: [Salem and Buehler 2013])

Protocolo	**Estado**	***Timeout*** (em segundos)
UDP	-	180
ICMP	-	180
TCP	*Handshake*	20
TCP	*Established*	720
TCP	*Termination*	675
TCP	*Closed*	240

[6]Geralmente ataques de negação de serviço, tais como *SYN Flood*, esgotam os recursos de um alvo por meio da abertura de diversas conexões, sem finalizá-las

Grande parte das características utilizadas foram definidas baseando-se nas propostas de Moustafa and Slay 2016, Salem et al. 2014, Zhang and Zulkernine 2005 e Tavallaee et al. 2009. O conjunto de características de conexão pode ser dividido em 3 tipos, são eles: Características da Conexão (Tabela 5), Características do *Buffer* de Tempo (Tabela 6) e Características do *Buffer* de Conexões (Tabela 7).

Tabela 5. Características da Conexão

Num.	Característica	Descrição	Tipo
1	*Duração*	Tempo (em segundos) da conexão	contínuo
2	*Protocolo*	tipo do protocolo (Ex.: TCP, UDP, ICMP)	discreto
3	*Serviço*	Serviço utilizado (determinado pela porta)	discreto
4	*Flag da Conexão*	Estado da Conexão (Ex.: *Handshake, Established, Termination, Closed*)	discreto
5	*SourceToDest*	Quantidade de *bytes* enviados da Origem para o Destino	contínuo
6	*DestToSource*	Quantidade de *bytes* enviados do Destino para a Origem	contínuo
7	*Land*	1 se a conexão é de/para o mesmo destino/porta, 0 o inverso	discreto
8	*Wrong*	Número de pacotes com erro de *checksum*	contínuo
9	*Urgent*	Número de pacotes TCP com a *Flag* Urgent	contínuo
10	*STTL*	TTL do primeiro pacote da Origem	contínuo
11	*DTTL*	TTL do primeiro pacote do Destino	contínuo
12	*SourceToDestPkts*	Contador dos pacotes enviados da Origem para o Destino	contínuo
13	*DestToSourcePkts*	Contador dos pacotes enviados do Destino para a Origem	contínuo

Tabela 6. Características do *Buffer* de Tempo (Padrão: 2 segundos)

Num.	Característica	Descrição	Tipo
14	*CountSameHost*	Contador de conexões para o mesmo *host*	contínuo
15	*CountSameService*	Contador de conexões com o mesmo serviço	contínuo
16	*Serror_rate*	% de conexões para o mesmo *host*, com erros de *SYN* — Aplicável apenas ao TCP	contínuo
17	*Srv_serror_rate*	% de conexões para o mesmo serviço, com erros de *SYN* — Aplicável apenas ao TCP	contínuo
18	*Same_srv_rate*	% de conexões para o mesmo serviço	contínuo
19	*Diff_srv_rate*	% de conexões para serviços diferentes	contínuo
20	*Srv_diff_host_rate*	% de conexões para o mesmo serviço com *host* diferente	contínuo

Tabela 7. Características do *Buffer* de Conexões (Padrão: 100 conexões)

Num.	Característica	Descrição	Tipo
21	*Count*	Contador de conexões para o mesmo *host*	contínuo
22	*Srv_count*	Contador de conexões com o mesmo serviço	contínuo
23	*Same_srv_rate*	% de conexões para o mesmo *host*, com o mesmo serviço	contínuo
24	*Diff_srv_rate*	% de conexões para o mesmo *host*, com serviços diferentes	contínuo
25	*Same_src_port_rate*	% de conexões com a mesma porta de origem	contínuo
26	*Srv_diff_host_rate*	% de conexões para o mesmo serviço, com *host* diferente	contínuo
27	*Serror_rate*	% de conexões para o mesmo *host*, com erro de SYN — Aplicável apenas ao TCP	contínuo
28	*Srv_serror_rate*	% de conexões para o mesmo serviço, com erro de SYN — Aplicável apenas ao TCP	contínuo

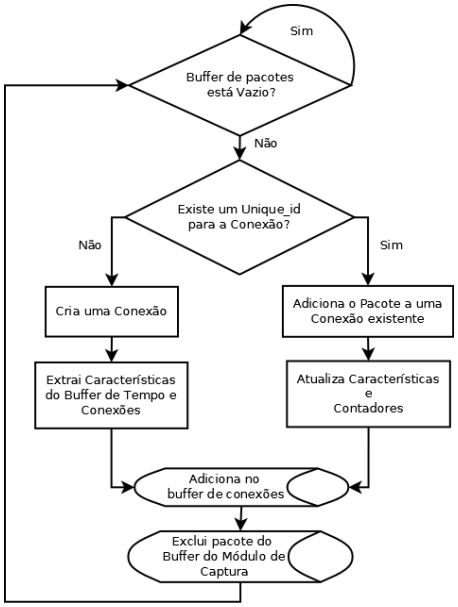

Figura 3. Fluxograma do Módulo de Gerenciamento

O funcionamento do sistema consiste em 5 módulos, um módulo de Captura de Pacotes, um módulo de Gerenciamento de Conexões (Figura 3), um módulo de Monitoramento de *timeouts*, um módulo de Classificação e um módulo de Notificação, todos operando de forma concorrente. A definição de cada módulo é apresentada nos tópicos a seguir:

- **Módulo de Captura de Pacotes:** É responsável por deixar a interface de rede em modo promíscuo e, utilizando um filtro, capturar todos os pacotes que entram ou saem nessa interface. Esse módulo também é responsável por obter as características necessárias do cabeçalho dos pacotes e despachá-las para o módulo de Gerenciamento de Conexões;
- **Módulo de Gerenciamento de Conexões:** Tem como principal função determinar o destino de cada pacote, baseando-se no *Unique_id*. Também é responsável por realizar todo o tipo de atualização e obtenção das características da conexão, tais como: quantidade de dados da origem para o destino, número de *flags* do TCP, número de erros de *checksum*, atualização da *Flag* de conexão, atualização do estado da conexão, ajuste do *timeout*, entre outros;
- **Módulo de Monitoramento de *Timeouts*:** Analisa, continuamente, as conexões que estão em *buffer*, verificando se o *timeout* de cada conexão expirou. Caso afirmativo, faz uma cópia da conexão e despacha para o Módulo de Classificação, utilizando *socket*. Logo após, a conexão é excluída do *buffer*.;
- **Módulo de Classificação:** É um módulo a parte do *framework*, é constituído pela Técnica de Inteligência Computacional, recebe os dados utilizando *sockets* e realiza a classificação. Caso uma conexão seja classificada como Intrusão, os dados são despachados para o módulo de notificação;
- **Módulo de Notificação:** Caso o sistema esteja funcionando *Online*, um *log* será gerado a cada intrusão detectada, contendo todos as informações da conexão capturada;

O *framework* foi desenvolvido em Java, utilizando o ambiente de programação *Eclipse Mars*. Para fazer o uso da aplicação e/ou modificá-la existem alguns pré-requisitos, são eles: OpenJDK 8, Libpcap 0.8, JnetPcap 1.4.

Já o formato dos dados de saída da aplicação, que são disponibilizados para as técnicas de Inteligência Computacional, via *socket*, são apresentados seguindo o modelo dos tópicos abaixo:

- 63.0,TCP,HTTP,S0,280,[...],30,100.0,0.0,0.0,93.0,100.0,6.0,normal;
- 0.0,UDP,DNS,SF,16,[...],100.0,0.0,4.0,66.0,0.0,0.0,ataque;

Por fim, o *framework* é capaz de operar de três formas distintas, com intuito de suprir as necessidades dos pesquisadores interessados. As formas de operação são:

- **Online:** Em conjunto com os módulos de Inteligência Computacional e Notificação, esse modo de operação permite que o *framework* opere como um NIDS, classificando o tráfego *Online* na rede de computadores. Os dados pré-processados são disponibilizados para o motor de classificação utilizando *sockets* ou arquivo texto;
- **Base de dados:** Este modo de operação consiste em realizar a leitura de uma base de dados e seus respectivos *labels*, permitido assim a extração de características de pacotes brutos. Ao final do processamento, o *Unique_id*, juntamente com o *timestamp* das conexões, são comparados aos *labels* e definidos como Normal ou Intrusão (Caso o *label* não seja encontrado, a conexão é descartada). Os dados pré-processados são gravados em um arquivo texto;
- **Construção de Base de Dados:** Este modo de operação consiste em realizar a construção de uma nova base de dados. Os pacotes são pré-processados, extraindo todas as características necessárias, sendo classificados por uma *tag default* (Normal ou Intrusão). Portanto é inevitável que algumas poucas conexões sejam classificadas de modo incorreto. Cabe ao usuário realizar o isolamento da rede para capturar tráfegos normais e intrusivos.

Os diferentes modos de operação permitem aos pesquisadores realizar diversos experimentos, sem precisar alterar o *framework*. Cabe ressaltar que, havendo a necessidade de um novo modo de operação, é possível adicionar novos recursos, de forma simplificada, ao *framework*.

3.3. Máquinas de Vetores de Suporte

Após os dados serem processados pelo *framework* com modo de operação ”Base de dados”, é possível aplicá-los ao treinamento do SVM. Porém, antes disso é necessário realizar o pré-processamento dos dados, conforme os requisitos da biblioteca LibSVM.

3.3.1. Pré-Processamento dos dados

Os processos de normalização e pré-processamento são, sem dúvida, fases muito trabalhosas e importantes para garantir a eficácia de qualquer método de inteligência computacional. Como os dados processados possuem algumas entradas não numéricas, é necessário realizar o pré-processamento para transformá-las em entradas numéricas, sendo assim é possível realizar o treinamento do SVM.

Alguns fatores precisam ter uma atenção especial para o tratamento dos dados, por exemplo: os protocolos, as *Flags* de conexões e os nomes dos serviços. Como esses parâmetros não são uma grandeza, onde é possível definir valores de distância entre si, é necessário implementar um algoritmo que realiza a codificação dos dados de nominal para binário, permitindo assim o processamento pela LibSVM. Os tópicos abaixo exemplificam o algoritmo, onde n é o número de serviços contabilizados.

- http = [1 0 ... n];
- smtp = [0 1 ... n];

Como os resultados são caracterizados como normal ou ataque, os valores resultantes para normal e intrusão foram normalizados para +1 e -1, respectivamente.

3.3.2. Treinamento do SVM

O treinamento do SVM foi realizado a partir de uma ferramenta (*easy.py*) da biblioteca LibSVM, a qual encontra os melhores parâmetros de treinamento para um determinado conjunto de dados. Os melhores parâmetros, de acordo com a ferramenta, são os seguintes:

- ***Kernel*:**
- **Custo:**
- ***Gamma*:**

4. Resultados e Discussões

5. Conclusões

6. Agradecimentos

Os autores agradecem a FAPEMIG pelo apoio financeiro para a publicação deste trabalho de pesquisa.

Referências

Ben-Hur, A. and Weston, J. (2010). A user's guide to support vector machines. *Data mining techniques for the life sciences*, pages 223–239.

Bing, X., Xiaosu, C., and Ning, C. (2009). An efficient tcp flow state management algorithm in high-speed network. In *Information Engineering and Electronic Commerce, 2009. IEEC'09. International Symposium on*, pages 106–110. IEEE.

CERT.BR (2016). Centro de estudos, resposta e tratamento de incidentes de segurança no brasil.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

de Azevedo, R. P. (2012). Detecção de ataques de negação de serviço em redes de computadores através da transformada wavelet 2d.

dos Santos, E. M. (2002). *Teoria e Aplicação de Support Vector Machines à Aprendizagem e Reconhecimento de Objetos Baseado na Aparência*. PhD thesis, Universidade Federal da Paraíba.

Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). A practical guide to support vector classification.

IATAC, T. M. W. (2009). Information assurance technology analysis center (iatac).

Jalil, K. A., Kamarudin, M. H., and Masrek, M. N. (2010). Comparison of machine learning algorithms performance in detecting network intrusion. In *Networking and Information Technology (ICNIT), 2010 International Conference on*, pages 221–226, Fac. of Comput. e Math. Sci., Univ. Teknol. MARA, Shah Alam, Malaysia. IEEE.

Jha, J. and Ragha, L. (2013). Intrusion detection system using support vector machine. *International Journal of Applied Information Systems (HAIS)-ISSN*, pages 2249–0868.

Karen Scarfone, P. M. (2007). *Guide to Intrusion Detection and Prevention Systems (IDPS)*. National Institute of Standards and Technology Gaithersburg.

Laureano, M. A. P., Maziero, C. A., and Jamhour, E. (2003). Detecção de intrusão em máquinas virtuais. *5º Simpósio de Segurança em Informática–SSI. São José dos Campos*, pages 1–7.

Li, H., Guan, X.-H., Zan, X., and HAN, C.-Z. (2003). Network intrusion detection based on support vector machine. *Journal of Computer Research and Development*, 6:799–807.

LIMA, R. A. G. (2014). *Utilização de Sistemas Inteligentes para Classificação de Tráfego Malicioso*. PhD thesis, Universidade Federal Viçosa.

Moustafa, N. and Slay, J. (2016). The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. *Information Security Journal: A Global Perspective*, pages 1–14.

Mulay, S. A., Devale, P., and Garje, G. (2010). Intrusion detection system using support vector machine and decision tree. *International Journal of Computer Applications*, 3(3):40–43.

Rao, X., Dong, C.-X., and Yang, S.-Q. (2003). An intrusion detection system based on support vector machine. *Journal of Software*, 4(14).

Rodrigues, R. C. B., Viana, R., Pasquali, A., Pistori, H., and Alvarez, M. A. (2007). Máquinas de vetores de suporte aplicadas à classificação de defeitos em couro bovino. *Qualificação de Bacharel em Engenharia de Computação na Universidade Católica Dom Bosco Campo Grande-MS-Brasil*.

Salem, M. and Buehler, U. (2013). Reinforcing network security by converting massive data flow to continuous connections for ids. In *Internet Technology and Secured Transactions (ICITST), 2013 8th International Conference for*, pages 570–575. IEEE.

Salem, M., Reissmann, S., and Buehler, U. (2014). Persistent dataset generation using real-time operative framework. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 1023–1027. IEEE.

Scalco Neto, H. (2015). Reconhecimento de intrusão em redes de computadores utilizando pybrain. *12 Congresso Brasileiro de Inteligência Computacional*.

Sen, N., Sen, R., and Chattopadhyay, M. (2014). An effective back propagation neural network architecture for the development of an efficient anomaly based intrusion detection system. In *Computational Intelligence and Communication Networks (CICN), 2014 International Conference on*, pages 1052–1056.

Shiravi, A., Shiravi, H., Tavallaee, M., and Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374.

SILVA, J. R. C. D. (2011). *Sistemas de Detecção de Intrusão com técnicas de inteligencia artificial*. PhD thesis, Universidade Federal de Viçosa.

SNORT.ORG (2016). Snort ids.

Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A.-A. (2009). A detailed analysis of the kdd cup 99 data set.

Uchoa, J. Q. (2009). Algoritmos imunoinspirados aplicados em segurança computacional: utilizaçao de algoritmos inspirados no sistema imune para detecçao de intrusos em redes de computadores.

Wang, J. (2009). *Computer network security: theory and practice*. Springer Publishing Company, Incorporated, University of Massachusetts.

Zhang, J. and Zulkernine, M. (2005). Network intrusion detection using random forests. In *PST*. Citeseer.