

CURSO DE ENGENHARIA DE COMPUTAÇÃO

Jáder Friderichs Vieira

**DESENVOLVIMENTO DE UM IDS BASEADO EM TÉCNICAS DE
*DATA MINING***

Prof. Me. Charles Varlei Neu
Orientador

Santa Cruz do Sul, novembro de 2016

Jáder Friderichs Vieira

**DESENVOLVIMENTO DE UM IDS BASEADO EM TÉCNICAS DE
*DATA MINING***

Trabalho de Conclusão II apresentado ao Curso de Engenharia de Computação da Universidade de Santa Cruz do Sul, como requisito parcial para a obtenção do título de Bacharel em Engenharia da Computação.

Orientador Prof. Me. Charles Varlei Neu

Santa Cruz do Sul, Novembro de 2016

2016

“Eu não tenho ídolos. Tenho admiração por trabalho, dedicação e competência.”

-Ayrton Senna

AGRADECIMENTOS

Meu agradecimento inicial, vai para meus pais, Luís e Dirce, responsáveis por eu estar hoje aqui, concluindo uma graduação. Não é possível mencionar as tantas formas que vocês me ajudaram, seja na aula de história dada pela mãe, depois da novela, ainda na quinta série ou meu pai me buscando no colégio à tardinha, cansado do trabalho. Meu irmão Vinícius também merece o meu agradecimento, seja por ouvir as reclamações dos professores sobre mim na 8ª série, ou por me buscar na aula, já um adolescente no ensino médio. O que posso dizer é, família não se escolhe, mas se pudesse escolher, eu não escolheria uma diferente por nada no mundo. Amo vocês!

Meu agradecimento especial, para minha namorada Tainara, que na metade desse caminho, me ajudou a seguir, e sempre acreditou em mim, até quando eu não acreditava! Uma verdadeira companheira, com quem eu desejo compartilhar futuramente os frutos desta graduação. TE AMO VIDA!

Meu mais sincero agradecimento, aos amigos que fiz e os professores que me ajudaram nessa longa jornada.

RESUMO

Estamos na era digital, onde a maior parte dos dispositivos eletrônicos estão conectados, transmitindo e recebendo informações através de uma rede. Geralmente, essas informações são de extrema importância e valor, como no caso das grandes empresas que são alvos constantes de pessoas e sistemas mal-intencionados, que causam prejuízos e problemas. Dessa forma, a detecção de ataques é de extrema importância para os administradores de redes, já que muitos métodos de proteção (controle de acesso, políticas de firewall, criptografia de dados), podem não ser suficientes. Os IDS – *Intrusion Detection System* ou Sistemas de Detecção de Intrusão tornaram-se essenciais como complemento da infraestrutura das redes atuais, assim como novas técnicas de detecção integradas a essas ferramentas. Este trabalho propõe, a utilização de algoritmos de *Data Mining* para gerar padrões de detecção de ataques *SYN Flood*, com base nos dados do dataset *KDDCup '99*. Esses padrões gerados são utilizados para o desenvolvimento de um IDS, para detectar esse tipo de ataque. Os resultados mostraram que o IDS desenvolvido, foi eficaz na detecção de ataques *SYN Flood*.

Palavras-chave: IDS, SNORT, Ataques, DoS, *Syn Flood*, *Data Mining*, J48, *Random Tree*.

ABSTRACT

We are in the digital age, where most of the electronic devices are connected, transmitting and receiving information through a network. Generally, this information is of utmost importance and value, as in the case of large companies that are constant targets of people and malicious systems, which cause losses and problems. In this way, attack detection is of utmost importance to network administrators, since many methods of protection (access control, firewall policies, and data encryption), may not be sufficient. IDS - Intrusion Detection System have become essential as a complement to the infrastructure of the current networks, as well as new detection techniques integrated with these tools. This work proposes the use of Data Mining algorithms to generate SYN Flood attack detection patterns, based on data from the KDDCup'99 dataset. These generated patterns are used for the development of an IDS to detect this type of attack. The results showed that the IDS developed was effective in detecting SYN Flood attacks.

Keywords: IDS, SNORT, Attacks, DoS, *Syn Flood*, *Data Mining*, J48, *Random Tree*.

LISTA DE ILUSTRAÇÕES

Figura 1 - Ilustração de alguns protocolos de acordo com sua camada <i>TCP/IP</i>	15
Figura 2 - Relação de categoria de ataques por camada do <i>TCP/IP</i>	16
Figura 3 - Localização de um <i>HIDS</i> em uma rede <i>TCP/IP</i>	21
Figura 4 - Localização de um <i>NIDS</i> em uma rede <i>TCP/IP</i>	22
Figura 5 - Uma visão geral dos passos que constituem o processo de <i>KDD</i>	26
Figura 6 - Mecanismo de detecção do <i>Snort</i>	31
Figura 7 - Estrutura básica das regras do <i>Snort</i>	32
Figura 8 - Estrutura do cabeçalho de regra do <i>Snort</i>	32
Figura 9 - Exemplo de regra no <i>Snort</i>	33
Figura 10 - Arquitetura de geração das regras.	36
Figura 11 – Tabela attack com os atributos do <i>KDDCup'99</i>	37
Figura 12 - Atributo attack após execução na ferramenta <i>Weka</i>	39
Figura 13 - Dados prontos para o <i>Data Mining</i>	40
Figura 14 - Árvore gerada pelo algoritmo <i>J48</i>	42
Figura 15 - Árvore gerada pelo algoritmo <i>RandomTree</i> parte 1.	43
Figura 16 - Árvore gerada pelo algoritmo <i>RandomTree</i> parte 2.	44
Figura 17 - Logs de pacotes suspeitos gerados pelo <i>Snort</i>	50
Figura 18 - Interface da Aplicação <i>Java</i>	51
Figura 19 - Arquivo de logs de ataques gerado pelo <i>IDS</i> desenvolvido.	52
Figura 20 - Arquitetura de detecção.	53
Figura 21 - Ambiente de testes.	54
Figura 22 - Filtro de pacotes no <i>Wireshark</i> de acordo com o intervalo de captura.	55
Figura 23 - Execução de um arquivo <i>.pcap</i> no <i>SNORT</i>	57
Figura 24 - Execução de um ataque com a ferramenta <i>Slowloris</i>	57
Figura 25 - Execução de um ataque com a ferramenta <i>Hping3</i>	58

LISTA DE TABELAS

Tabela 1 - Ataques relacionados a categoria de Probing.....	17
Tabela 2 - Ataques relacionados a categoria DoS.....	18
Tabela 3 - Ataques relacionados a categoria U2R.....	19
Tabela 4 - Ataques relacionados a categoria R2L.....	20
Tabela 5 - Parâmetros de desempenho de um IDS.....	23
Tabela 6 - Principais tarefas de Data Mining.....	26
Tabela 7 - Comparativo entre trabalhos estudados e o trabalho proposto.....	35
Tabela 8 - Descrição dos atributos do KDDCup'99.....	38
Tabela 9 – Resultados obtidos para o algoritmo J48, utilizando datasets.....	59
Tabela 10 - Resultados obtidos para o algoritmo <i>Random Tree</i> , utilizando datasets.....	60
Tabela 11 - Resultados obtidos para o algoritmo J48 utilizando a ferramenta Slowloris.....	61
Tabela 12 - Resultados obtidos para o algoritmo J48 utilizando a ferramenta Hping3.....	61
Tabela 13 - Resultados obtidos para o algoritmo <i>Random Tree</i> utilizando a ferramenta Slowloris.....	62
Tabela 14 - Resultados obtidos para o algoritmo <i>Random Tree</i> utilizando a ferramenta Hping3.....	62
Tabela 15 - Desempenho do IDS com o Datasets para o algoritmo J48.....	64
Tabela 16 - Desempenho do IDS com o Datasets para o algoritmo <i>Random Tree</i>	64
Tabela 17 - Desempenho do IDS em tempo real para o algoritmo J48.....	65
Tabela 18 - Desempenho do IDS em tempo real para o algoritmo <i>Random Tree</i>	65

LISTA DE ABREVIATURAS

API	<i>Application Programming Interface</i>
CPU	<i>Central Processing Unit</i>
DNS	<i>Domain Name System</i>
DOS	<i>Denial of Service</i>
FTP	<i>File Transfer Protocol</i>
GNU	<i>General Public License</i>
HIDS	<i>Host Intrusion Detection System</i>
IDS	<i>Intrusion Detection System</i>
ICMP	<i>Internet Control Message Protocol</i>
IP	<i>Internet Protocol</i>
KDD	<i>Knowledge Discovery in Databases</i>
NIDS	<i>Network Intrusion Detection System</i>
RAM	<i>Random Access Memory</i>
RNA	<i>Rede Neural Artificial</i>
R2L	<i>Remote to Local Attack</i>
SMTP	<i>Simple Mail Transfer Protocol</i>
SQL	<i>Structured Query Language</i>
TCP	<i>Transmission Control Protocol</i>
UDP	<i>User Datagram Protocol</i>
U2R	<i>User to Root Attack</i>
WEKA	<i>Waikato Environment for Knowledge Analysis</i>
SNMP	<i>Simple Network Management Protocol</i>
TTL	<i>Time to Live</i>

SUMÁRIO

1.INTRODUÇÃO.....	12
2.FUNDAMENTAÇÃO TEÓRICA	14
2.1. Redes TCP/IP.....	14
2.2. Definição e tipos de Intrusão	15
2.2.1. <i>PROBING</i> (exploração).....	17
2.2.2. <i>DoS</i> (Negação de serviço).....	18
2.2.3. <i>U2R (User to Root attack)</i>	19
2.2.4. <i>R2L (Remote to local attack)</i>	20
2.3. Sistemas de detecção de intrusão.....	21
2.4. <i>Datasets</i>	23
2.5. <i>KDD e Data Mining</i>	24
2.5.1. O processo de <i>KDD</i>	25
2.5.2. Tarefas de <i>Data Mining</i>	26
2.5.3. Algoritmos de classificação	27
3. TECNOLOGIAS ENVOLVIDAS.....	28
3.1. Snort.....	28
3.1.1. Introdução ao Snort	28
3.1.2. Modos de operação	29
3.1.3. Arquitetura e funcionamento	29
3.1.4. Regras.....	31
3.2. <i>WEKA</i>	33
4. TRABALHOS RELACIONADOS	34
5. TRABALHO DESENVOLVIDO	36
5.1. Geração dos padrões do ataque	36
5.1.1. Pré-processamento	37
5.1.2. <i>Data Mining</i>	41
5.1.3. Interpretação/Avaliação.....	45
5.2. Log de pacotes suspeitos	47

5.3. Detecção do ataque	50
6. RESULTADOS	54
6.1. Ambiente de testes	54
6.2. Testes realizados	55
6.2.1. Testes com datasets	55
6.2.2. Testes em tempo real.....	57
6.3. Resultados Obtidos	59
6.4. Análise de desempenho	63
7. CONCLUSÃO E CONSIDERAÇÕES FINAIS	66
REFERÊNCIAS	67

1. INTRODUÇÃO

Os sistemas de rede atuais são alvos dos mais diversos tipos de ataques, por motivos banais, econômicos ou pelo comércio criado, onde se vende ataques a quem esteja interessado em comprar (NAFIR *et al.*, 2014). Dessa forma a detecção de ataques em redes é uma tarefa fundamental para operadores de rede na Internet (ENACHE e SGÂRCIU, 2014). O ataque a Sony Pictures em novembro de 2014, onde 100 Terabytes de dados foram roubados, contendo lançamentos de músicas e vídeos inéditos, especulando-se perdas superiores a 100 milhões de dólares (RICHWINE, 2014), é apenas mais um exemplo dos prejuízos causados pelas intrusões.

Muitas técnicas de proteção são utilizadas a fim de administrar os riscos de segurança da informação (controle de acesso, políticas de firewall, criptografia de dados). Estes métodos não são suficientes e por isso a necessidade da utilização dos IDS - *Intrusion Detection System* ou Sistemas de Detecção de Intrusão, onde tornaram-se um complemento essencial para a infraestrutura de segurança de quase todas as organizações (CHANDRASEKHAR e RAGHUVVEER, 2014; ENACHE e SGÂRCIU, 2014; LAHOTI *et al.*, 2014).

Existe uma busca incessante por parte de pesquisadores e estudiosos, por técnicas mais eficientes de detecção de intrusão para utilização pelos IDS. Por exemplo, em Silva (SILVA, 2005), o autor propõe o uso de redes neurais artificiais no auxílio a detecções em redes de pacotes *TCP/IP*. Ele acredita que a capacidade de generalização das RNA – Redes Neurais Artificiais mantém uma taxa alta de acertos para novos ataques.

Em Souza *et al.* (SOUZA e MONTEIRO, 2008), os autores acreditam que um *NIDS* – *Network Intrusion Detection System* ou Sistema de Detecção de Intrusão de Rede por anomalias, aliado a uma rede neural para aprendizagem, torna a detecção mais eficaz. Temos exemplos teóricos utilizando aprendizado de máquina com o classificador *Naive Bayes*, onde os autores baseiam-se na ideia de que a ponderação das amostras mais recentes, permite ao classificador se adaptar mais rapidamente a novos ataques (GUMUS *et al.*, 2014).

Os *IDS* monitoram dinamicamente as ocorrências em um sistema e arbitram se esses eventos são suscetíveis de um ataque. Foram criados para auxiliar os administradores de rede, quando alguma atividade suspeita ou maliciosa ocorre, dispara um aviso e registra a ocorrência (COMER, 1988; ENACHE e SGÂRCIU, 2014; ZAMAN e KARRAY, 2009; NJOGU *et al.*, 2013).

Um dos mais populares *IDS* é o Snort, que é definido como um sistema de detecção de intrusão de código aberto, capaz de analisar o tráfego em tempo real e analisar registros de

pacotes em redes *IP*. Realiza análises de protocolos, conteúdos de buscas, varredura de portas e outros, em locais que possam ser alvos para iniciar uma intrusão. (SNORT, 2016).

O Snort é uma ferramenta constantemente atualizada e bastante popular pela flexibilidade em suas regras, de forma que podem ser personalizadas de acordo com a necessidade do administrador, se tornando muito útil para a detecção de intrusões não conhecidas e não encontradas na base de dados. Permite assim que, determinado um comportamento ou característica para a intrusão, criar uma regra para que elimine essa vulnerabilidade (SNORT, 2016).

Este trabalho tem como objetivo geral, obter métodos mais eficientes de detecção de intrusão para o ataque *DoS SYN Flood* (também conhecido como Neptune), utilizando técnicas de *Data Mining* aplicadas a um *IDS* desenvolvido, e integrado ao *IDS Snort*. Dessa forma, é feita uma tentativa de tornar as redes mais seguras contra esse tipo de ataque, e contribuir com pesquisas e estudos já existentes na área.

Para alcançar o objetivo geral, foi necessário atender um conjunto de objetivos específicos para elaboração deste trabalho que são: (i) estudo da estrutura de redes *TCP/IP*, (ii) estudo e utilização do *dataset KDDCup'99*, (iii) pesquisa e utilização das etapas da *KDD*, incluindo a etapa de *Data Mining* com o auxílio da ferramenta *Weka*, (iv) integração do SNORT com o *IDS* desenvolvido, (v) testes de eficiência utilizando *datasets* e ferramentas geradoras de ataques.

Sendo assim, o restante deste trabalho foi dividido da seguinte forma: no capítulo 2 é apresentada a fundamentação teórica para a realização do trabalho, capítulo 3 são abordadas as tecnologias utilizadas no desenvolvimento, o capítulo 4 são apresentados os trabalhos relacionados, no capítulo 5 é apresentado o trabalho desenvolvido, no capítulo 6 são apresentados os resultados obtidos com o trabalho. Por fim, no capítulo 7 é feita a conclusão e as considerações finais do trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

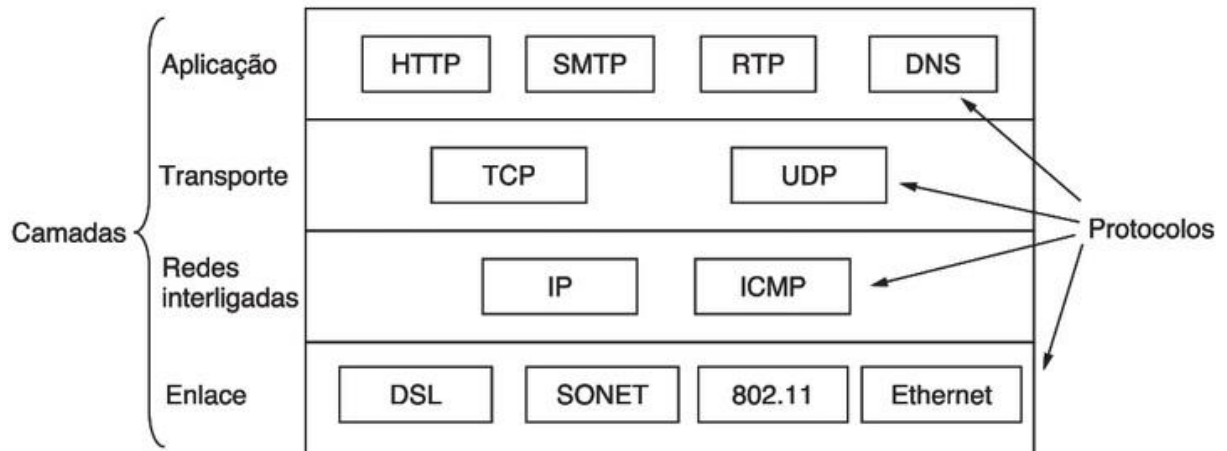
Neste capítulo são apresentados os tópicos estudados, que foram fundamentais para a conclusão deste trabalho.

2.1. Redes TCP/IP

O modelo de referência *TCP/IP* foi criado com o intuito de conectar várias redes de maneira uniforme, e fosse capaz de sobreviver à perda de hardware de sub-redes, sem interromper as conversações existentes. Segundo (TANENBAUM, 2012), o modelo TCP/IP se divide em 4 camadas:

- **Enlace:** é a camada mais baixa do modelo TCP/IP, descreve o que os enlaces como linhas seriais e a Ethernet clássica precisam para a interconexão com o serviço não orientado a conexões, não é uma camada propriamente dita, mas uma interface entre os *hosts* e os enlaces de transmissão.
- **Rede:** a tarefa da camada de rede é permitir a inserção de pacotes na rede pelo host e garantir o tráfego deles até o destino. Define dois protocolos, o *IP* que é acompanhado pelo *ICMP*. A camada de Internet é responsável pelo roteamento e entrega dos pacotes *IP* ao destino.
- **Transporte:** está localizada acima da camada de Internet e sua função é manter uma comunicação entre os *hosts* origem e destino. Define então o protocolo *TCP* que é orientado a conexões e confiável, faz controle de fluxo no destino, para entrega dos pacotes sem erros de uma determinada origem a qualquer computador na Internet. Outro protocolo definido por essa camada é o *UDP*, não orientado a conexões e também não confiável. Geralmente utilizado por aplicações que não necessitam sequência ou controle de fluxo como o do *TCP*.
- **Aplicação:** localizada acima da camada de transporte, contendo os protocolos de nível mais alto como *TELNET*, *FTP*, *SMTP*, *DNS* e outros. Recebe os dados das camadas mais baixas e fornece os serviços direto aos usuários através de seus protocolos.

Figura 1 - Ilustração de alguns protocolos de acordo com sua camada *TCP/IP*.



Fonte: Tanenbaum (2012, p. 28).

2.2. Definição e tipos de Intrusão

Uma situação onde podemos definir uma intrusão ou ataque, são ações executadas com a finalidade de comprometer a integridade, confidencialidade ou disponibilidade de dados ou recursos (JUNQI; ZHENG BING, 2008).

Bishop (2004) define esses três componentes da seguinte forma:

- **Integridade:** refere-se à confiabilidade e integridade dos dados ou recursos, garantindo que são provenientes de uma fonte confiável.
- **Confidencialidade:** ocultação de informações ou recursos. Por exemplo, manter em segredo informações decorrentes de áreas sensíveis, tais como governo ou indústria.
- **Disponibilidade:** refere-se à capacidade de usar a informação ou recurso desejado.

As intrusões ocorrem por meio de exploração de técnicas ou engenharia social. A engenharia social explora a pessoa que pode conceder acesso ao recurso, ludibriando e

enganando para que possa alcançar seus objetivos. Esse recurso pode ser uma senha ou outras informações que comprometam a segurança da rede que será atacada.

No caso da utilização de técnicas para intrusões, os atacantes exploram vulnerabilidades na implementação de sistemas, serviços, protocolos e outros. Existe também os problemas gerados por usuários e administradores, como a configuração e manutenção imprópria dos sistemas, senhas ineficientes ou ainda alguma brecha que possa ser deixada (NAKAMURA; GEUS, 2007).

Os ataques são definidos em 4 categorias principais dependendo da camada TCP/IP em que ocorrem (ZAMAN e KARRAY, 2009; CAMPOS e LIMA, 2012). As categorias principais de ataques, de acordo com a camada TCP/IP em que ocorrem podem ser vistas na Figura 2.

Figura 2 - Relação de categoria de ataques por camada do TCP/IP.

CAMADA TCP/IP	ATAQUES
APLICAÇÃO	PROBING
TRANSPORTE	DoS
INTERNET	U2R
ENLACE	R2L

Fonte: Adaptado de Zaman *et al.* (ZAMAN e KARRAY, 2009).

2.2.1. *PROBING* (exploração)

A principal característica dessa categoria de ataque é a coleta de informações sobre uma rede de computador, com o propósito de burlar os controles de segurança. Alguns ataques desta categoria, podem ser vistos na Tabela 1, assim como os serviços utilizados pelos atacantes, plataformas vulneráveis, mecanismos para efetuar o ataque e o efeito que eles exercem sobre a rede ou *host*.

Tabela 1 - Ataques relacionados a categoria de Probing.

Nome	Serviço	Plataformas vulneráveis	Mecanismo	Efeito
Ipsweep	Icmp	Todas	Abuso de funcionalidade	Encontra computadores ativos
Mscan	Vários	Todas	Abuso de funcionalidade	Descobre vulnerabilidades conhecidas
Nmap	Vários	Todas	Abuso de funcionalidade	Encontra serviços (portas tcp e udp) ativas
Saint	Vários	Todas	Abuso de funcionalidade	Descobre vulnerabilidades conhecidas
Satan	Vários	Todas	Abuso de funcionalidade	Descobre vulnerabilidades conhecidas

Fonte: Darpa (DARPA, 2016).

2.2.2. DoS (Negação de serviço)

O ataque *DoS – Denial of Service* ou Negação de Serviço, acontece quando o atacante envia um grande número de mensagens que esgote alguns recursos da vítima, como CPU, memória, banda ou outro e não consiga mais atender as requisições. Na Tabela 2, podemos ver exemplos de ataques *DoS*, assim como os serviços utilizados pelos atacantes, plataformas vulneráveis, mecanismos para efetuar o ataque e o efeito que eles exercem sobre a rede ou *host*.

Tabela 2 - Ataques relacionados a categoria DoS.

Nome	Serviço	Plataformas vulneráveis	Mecanismo	Efeito
Apache2	http	Apache	Abuso de funcionalidade	Interrupção do serviço http
Back	http	Apache	Abuso / Bug no sistema	Queda no tempo de resposta
Land	N/A	SunOS	Bug no sistema	Sistema operacional indisponível
MailBomb	Smtip	Todas	Abuso de funcionalidade	Consumo de recurso
SYN Flood(Neptune)	TCP	Todas	Abuso de funcionalidade	Negação de serviço
Ping of Death	Icmp	Todas	Bug no sistema	Indisponibilidade
Process Table	TCP	Todas	Abuso de funcionalidade.	Impede execução de novos processos.
Smurf	Icmp	Todas	Abuso de funcionalidade	Queda no desempenho da rede
SyslogD	Syslog	Solaris	Bug no sistema	Queda do serviço Syslog.
TearDrop	N/A	Linux	Bug no sistema.	Reinicializa computador vítima
UDPStorm	Echo / chargen	Todas	Abuso de funcionalidade.	Queda no desempenho da rede

Fonte: Darpa (DARPA, 2016).

2.2.3. U2R (*User to Root attack*)

Essa categoria utiliza algum dos ataques da sua lista, para acessar o sistema como um usuário padrão e então passa a explorar as vulnerabilidades para ganhar acesso de root. Na Tabela 3 podemos ver exemplos de ataques U2R, assim como os serviços utilizados pelos atacantes, plataformas vulneráveis, mecanismos para efetuar o ataque e o efeito que eles exercem sobre a rede ou *host*.

Tabela 3 - Ataques relacionados a categoria U2R.

Nome	Serviço	Plataformas vulneráveis	Mecanismo	Efeito
Eject	Qualquer sessão de usuário	Solaris	Buffer overflow	Shell com privilégios de administrador
Ffbconfig	Qualquer sessão de usuário	Solaris	Buffer overflow	Shell com privilégios de administrador
Fdfomat	Qualquer sessão de usuário	Solaris	Buffer overflow	Shell com privilégios de administrador
Loadmodule	Qualquer sessão de usuário	SunOS	Bug	Shell com privilégios de administrador
Perl	Qualquer sessão de usuário	Linux	Bug	Shell com privilégios de administrador
PS	Qualquer sessão de usuário	Solaris	Bug	Shell com privilégios de administrador
Xterm	Qualquer sessão de usuário	Linux	Buffer overflow	Shell com privilégios de administrador

Fonte: Darpa (DARPA, 2016).

2.2.4. R2L (Remote to local attack)

Ocorre quando o atacante envia pacotes para uma máquina através da rede, mas não tem uma conta nessa máquina, e explora alguma vulnerabilidade para ganhar acesso local como usuário da máquina. Na Tabela 4 podemos ver exemplos de ataques R2L, assim como os serviços utilizados pelos atacantes, plataformas vulneráveis, mecanismos para efetuar o ataque e o efeito que eles exercem sobre a rede ou *host*.

Tabela 4 - Ataques relacionados a categoria R2L.

Nome	Serviço	Plataformas vulneráveis	Mecanismo	Efeito
Dictionary	Qualquer serviço que possua autenticação por nome e senha	Todas	Abuso de funcionalidade	Acesso como usuário legítimo
Ftp-write	FTP	Todas	Configuração indevida	Acesso como usuário legítimo
Guest	Telnet, Rlog	Todas	Configuração indevida	Acesso como usuário legítimo
Imap	IMAP4	Linux	Bug	Shell com privilégios de administrador
Named	DNS	Linux	Bug	Shell com privilégios de administrador
Phf	Http	Todas	Bug	Execução de comandos como usuário http
Xlock	X	Todas	Configuração indevida	Obtém senha através de falsificação da identidade
Xsnoop	X	Todas	Configuração indevida	Capacidade de monitorar teclas digitadas remotamente

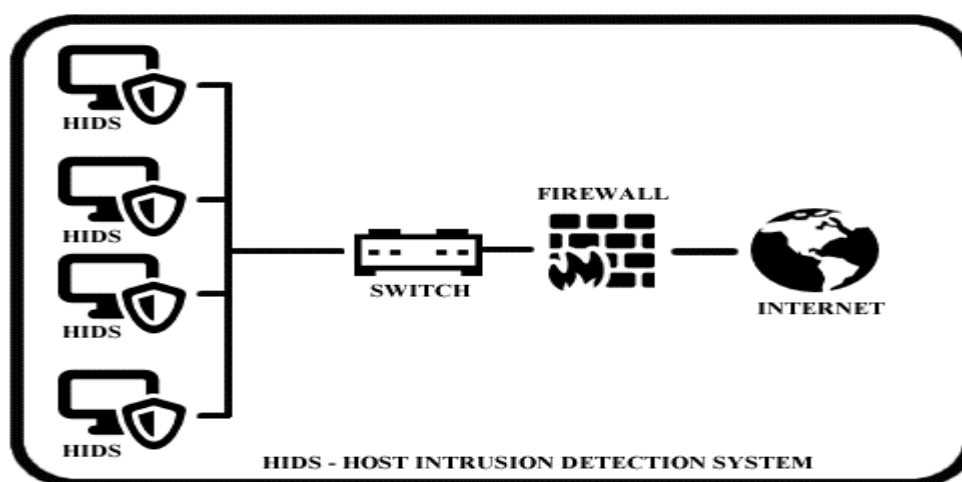
2.3. Sistemas de detecção de intrusão

Um sistema de detecção de intrusão é definido como um sistema de proteção que monitora o tráfego e o comportamento de uso computacional, para detectar qualquer atividade suspeita que possa comprometer a rede. O IDS é capaz de alertar os administradores sobre possíveis ataques ou comportamentos anormais na rede, obtendo informações importantes sobre tentativas de ataques, que não se pode obter normalmente (TRAN e JAN, 2006; NAKAMURA e GEUS, 2007; FAGUNDES *et al.*).

Em 1987, Denning *et al.* (DENNING *et al.*, 1987) criou um IDS de propósito geral, onde detectava ataques analisando comportamento, métricas e estatísticas, cobrindo apenas um dispositivo na rede. Então em 1990 Heberlein *et al.* (HEBERLEIN *et al.*, 1990), abordou um conceito a nível de rede, onde o IDS analisava os pacotes no segmento da rede e detectava o tráfego que estivesse fora do padrão definido, trazendo alguma ameaça para a rede. Em (NAKAMURA e GEUS, 2007) são apresentados os dois tipos principais de IDS:

- **Baseado em Host (*HIDS - Host Intrusion Detection System*):** utilizam arquivos de *logs* ou agentes de auditoria para detecção de intrusão. São capazes de monitorar acessos em arquivos, modificações em permissões, processos do sistema, programas em execução, entre outros. Normalmente os *HIDS* são considerados ferramentas ao invés de sistemas. Pois devido as detecções serem realizadas em informações de *logs* e registros do sistema, por muitas vezes os alertas não são em tempo real. A localização de um *HIDS* em uma rede *TCP/IP* pode ser vista na Figura 3.

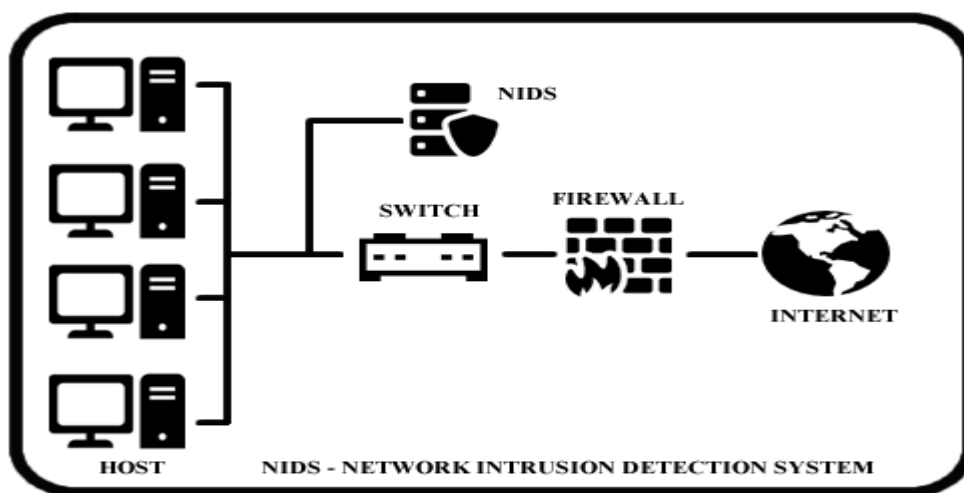
Figura 3 - Localização de um HIDS em uma rede TCP/IP.



Fonte: Autor.

Baseado em rede (NIDS - Network Intrusion Detection System): o sistema de detecção de intrusão baseado em rede monitora um segmento de rede, capturando pacotes e comparando o cabeçalho e o conteúdo com assinaturas e padrões conhecidos. A localização de um NIDS em uma rede TCP/IP pode ser vista na Figura 4.

Figura 4 - Localização de um NIDS em uma rede TCP/IP.



Fonte: Autor.

O NIDS se divide em duas partes: os sensores espalhados na rede e o gerenciador ou console. Os sensores são responsáveis pela captura, formatação dos dados e analisar o tráfego da rede, o gerenciador faz com que os sensores trabalhem de forma integrada definindo os alertas para cada comportamento suspeito detectado. Uma característica importante é a capacidade de detecção em tempo real, devido ao sensor estar no mesmo segmento do servidor atacado, ele pode responder ao ataque em tempo real, mas isso não garante que o ataque não foi efetivado. Os IDS podem utilizar duas abordagens para detectar uma intrusão, por anomalias ou assinaturas. Abaixo temos a definição dada por (SCARFONE e MELL, 2007) para cada um desses métodos.

- **Baseado em assinaturas:** a detecção baseada em assinaturas tem como objetivo, comparar as assinaturas do banco de dados do IDS com caracteres dentro dos pacotes analisados, a fim de identificar um padrão de caracteres conhecidos de ataques e emitir um alerta ao administrador. É a forma mais simples de detecção de intrusão, pois efetivamente utiliza apenas uma operação de comparação de sequência de caracteres, mesmo assim é muito efetivo para intrusões conhecidas.

- **Baseado em Anomalias:** nesse modelo o *IDS* busca um padrão de comportamento diferente do considerado normal. É criado um perfil, onde é monitorado as características dos usuários, *hosts*, conexões de rede ou aplicações durante atividades típicas, então quando o *IDS* percebe uma atividade com um comportamento diferente do perfil definido, emite um alerta ao administrador.

Um *IDS* confiável, deve manter uma baixa ocorrência de falsos positivos e falsos negativos (BISHOP, 2005). Os parâmetros mais importantes na estimação de desempenho de um *IDS*, segundo (CAMPOS *et al.*, 2012) são mostrados na Tabela 5.

Tabela 5 - Parâmetros de desempenho de um *IDS*.

Parâmetro	Definição
Taxa de detecção (DR)	Ataque ocorre e o alarme dispara.
Falso Positivo (FP)	Ataque não ocorre, mas o alarme dispara.
Verdadeiro Negativo (TN)	Ataque não ocorre e o alarme não dispara.
Falso Negativo (FN)	Ataque ocorre, mas alarme não dispara.

Fonte: Campos *et al.* (CAMPOS *et al.*, 2012).

2.4. Datasets

Datasets são conjuntos de dados de tráfego coletados de uma rede, contendo ou não tráfego malicioso (*KDD CUP 99*, 2016). A seguir pode ser visto exemplos de *datasets* e suas respectivas descrições.

- ***KDDCup'99:*** O *KDDCup'99* é um conjunto de dados utilizados no 3º Concurso Internacional de Descoberta de conhecimento e Mineração de dados. A tarefa era construir um detector de intrusão de rede, um modelo preditivo capaz de distinguir “más” conexões ou ataques e conexões normais (*KDD CUP 99*, 2016). Esta base de dados contém um conjunto padrão de dados a serem auditados, que inclui uma grande variedade de intrusões simulados em ambiente militar e abrange as categorias de ataques *DoS*, *R2L*, *U2R* e *Probing* (*KDD CUP 99*, 2016). É o conjunto de dados mais utilizado para avaliação de sistemas de detecção de intrusão (TAVALLAEE *et al.*, 2009).

- **Darpa:** O dataset Darpa foi criado no laboratório Lincoln do MIT (Massachusetts Institute of Technology), sob patrocínio da DARPA, onde foi coletado um conjunto de dados para avaliação de sistemas de detecção de intrusão. Os dados foram coletados de uma rede com tráfego em máquinas reais e simuladas, sendo os ataques efetuados contra máquinas reais. A coleta foi efetuada durante semanas, gerando arquivos com as 4 principais categorias de ataques e tráfego normal (THOMAS *et al.*, 2008).
- **UNB ISCX 2012:** O *dataset* UNB ISCX 2012 foi criado pelo ISCX- *Information Security Centre of Excellence*, pertencente a Universidade de New Brunswick no Canadá. O *dataset* foi baseado no conceito de perfis, contendo descrições detalhadas de intrusões e modelos abstratos de distribuição para aplicações, protocolos ou entidades de rede de nível inferior. Os traços reais foram analisados para criar perfis de agentes que geram tráfego real para HTTP, SMTP, SSH, IMAP, POP3 e FTP. Então foi criado um conjunto de diretrizes para delinear conjuntos de dados válidos, que estabelecem a base para a geração de perfis. Os perfis foram utilizados em vários cenários com ataques, então capturado o tráfego e gerado os arquivos do *dataset* (UNB ISCX, 2016).

2.5. *KDD e Data Mining*

Atualmente nos encontramos na era da informação digital, onde a capacidade de coletar e armazenar dados é muito maior do que nossa capacidade de analisar e compreender os mesmos. Técnicas computacionais e ferramentas são essenciais para o suporte a extração de conhecimento útil em grandes volumes de dados.

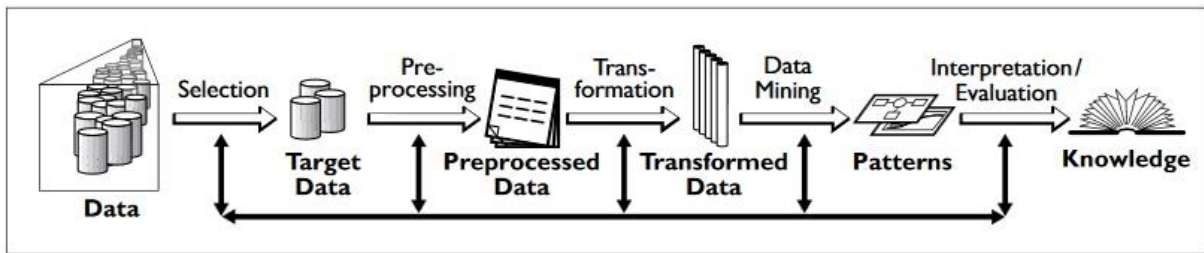
Essas técnicas e ferramentas são do campo da *KDD - Knowledge Discovery in Databases* ou Descoberta de Conhecimento em Base de dados, definido por Fayyad *et al.* como o processo de identificação de padrões válidos, potencialmente úteis e compreensíveis em dados. *Data Mining* ou Mineração de Dados, é a etapa principal desse processo, baseando-se em técnicas estatísticas, inteligência artificial, aprendizado de máquina e outros (FAYYAD *et al.*, 1996).

2.5.1. O processo de *KDD*

As etapas para descoberta de conhecimento podem ser vistas na Figura 5. Essas etapas são descritas por Fayyad *et al.* e citadas abaixo:

- **Compreensão do domínio:** desenvolver uma compreensão do domínio (dados) e os conhecimentos prévios relevantes para identificar o objetivo do processo *KDD*.
- **Selecionar os dados:** selecionar um *dataset*, ou um subconjunto de variáveis ou dados de amostras no qual a descoberta é para ser realizada.
- **Pré-processamento:** também conhecida como etapa de limpeza, se corrige erros e inconsistências caso existam e remoção de dados com ruído.
- **Redução de dados e projeção:** nesta etapa se converte os dados para que possam ser reconhecidos pelo algoritmo, e reduzido o número de variáveis caso necessário, com métodos de redução ou transformação de dimensionalidade.
- **Escolher o algoritmo de Mineração de Dados:** inclui a seleção do método a ser usado para a busca de padrões nos dados, tais como decidir quais modelos e parâmetros mais apropriados.
- **Mineração de Dados:** é a parte mais importante do processo, onde os dados são aplicados a um determinado algoritmo de Mineração de Dados com a finalidade de extrair padrões de conhecimento.
- **Interpretação/Avaliação dos dados:** é etapa de interpretação dos padrões gerados e avaliado o retorno.
- **Utilizar a descoberta de conhecimento:** na última parte do processo, já é conhecido os padrões e existe a descoberta de conhecimento, então é possível a tomada de ações sobre eles, documentar ou reportar aos interessados.

Figura 5 - Uma visão geral dos passos que constituem o processo de KDD.



Fonte: Fayyad *et al.* (FAYYAD; PIATETSKI-SHAPIRO; SMYTH, 1996).

2.5.2. Tarefas de *Data Mining*

As tarefas de *Data Mining* são responsáveis pelas informações que serão extraídas da base de dados, para determinar qual tarefa resolver, é essencial que conheça o domínio da aplicação e o que se deseja obter. As técnicas de mineração de dados estão diretamente ligadas as tarefas que irão resolver (FAYYAD; PIATETSKI-SHAPIRO; SMYTH, 1996). As principais tarefas de *Data Mining* são descritas de forma breve na Tabela 6.

Tabela 6 - Principais tarefas de *Data Mining*.

Classificação	Descrição	Exemplos
Classificação	Consiste em construir um modelo de algum tipo que possa ser aplicado a dados não classificados visando categorizá-los em classes. Um objeto é examinado e classificado de acordo com uma classe definida.	-Classificar solicitações de pedidos de crédito. -Esclarecer fraudes na declaração do imposto de renda.
Regressão	Regressão é aprender uma função que mapeia um item de dado para uma variável de predição real estimada.	-Prever a demanda futura de um novo produto. -Estimar expectativa de vida média dos brasileiros.
Associação	Identificação de grupos de dados que apresentem concorrência entre si.	-Quais produtos são colocados juntos em carrinhos de supermercado.
Segmentação (<i>Clustering</i>)	Processo de partição de uma população heterogênea em vários subgrupos ou grupos mais homogêneos.	-Agrupamento de clientes com comportamento de compras similar. -Comportamento de clientes em compras realizadas na web para uso futuro.
Detecção de desvios (<i>outliers</i>)	Identificação de dados que deveriam seguir um padrão esperado, mas não o fazem.	-Detecção de intrusão em redes de computadores.

Fonte: Campos *et al.* (CAMPOS *et al.*, 2012).

2.5.3. Algoritmos de classificação

Existem diversos algoritmos que utilizam técnicas de classificação e dos mais variados tipos, como Algoritmos de Bayes, Regras de decisão, Árvores de decisão e outros. Nesta seção são abordados de forma breve os 2 algoritmos utilizados neste trabalho: algoritmos árvore de decisão J48 e *Random Tree*.

- **J48:** o algoritmo J48 é uma evolução do algoritmo C4.5 na linguagem *Java*. Esse algoritmo gera uma árvore de decisão baseada em conjunto de dados de treinamento, dessa forma criando padrões para classificar os conjuntos de teste. Utiliza a abordagem dividir-para-conquistar, reduzindo o problema em subproblemas, assim recursivamente (WITTEN e WITTEN, 2005).
- ***Random Tree:*** o algoritmo *Random Tree* constrói uma árvore a partir de um conjunto de possíveis árvores, com K características aleatórias em cada nó. Nesse contexto, significa que no conjunto de árvores, cada uma delas tem uma chance igual de ser amostrada, podendo se dizer que as árvores têm uma distribuição “uniforme”. Árvores aleatórias podem ser geradas de forma eficiente, e o conjunto delas geralmente leva a modelos precisos (ALI *et al.*, 2012).

3. TECNOLOGIAS ENVOLVIDAS

Neste capítulo são abordadas as ferramentas utilizadas no desenvolvimento deste trabalho. Em 3.1 a ferramenta Snort que foi o IDS de monitoramento utilizado e em 3.2 a ferramenta Weka, que foi responsável pela execução dos algoritmos de *Data Mining* e geração dos padrões do ataque.

3.1. Snort

Neste tópico é abordada a ferramenta *IDS Snort*, que foi de fundamental uso neste trabalho. A seguir é feita uma introdução a ferramenta, assim como a sua arquitetura e características.

3.1.1. Introdução ao Snort

O Snort é uma ferramenta *NIDS* desenvolvida por Martin Roesch, “*Open Source*” com um código fonte otimizado e seus módulos desenvolvidos na linguagem de programação C. Sua documentação é aberta e de acesso público.

Esse *IDS* usa uma interface *Libcap*, muito utilizada por ferramentas de análise de pacotes. É um *sniffer* que consegue analisar o *payload* (área de dados) dos pacotes e registrá-los, quando os dados filtrados e comparados com as regras indicam uma intrusão ou ameaça, gera um alerta em um arquivo de *log* (KUROSE, 2010; SNORT, 2016).

O Snort é uma ferramenta muito difundida e utilizada pelos administradores de redes, possuindo uma comunidade oficial de usuários que ultrapassam os 500.000 membros registrados e mais de 4 milhões de *downloads*. Conta com uma equipe de pesquisa denominado Talos, que é um grupo de especialistas em segurança de rede trabalhando proativamente para descobrir, avaliar e responder às últimas tendências em atividades *Hacker*, tentativas de intrusão, *malware* e vulnerabilidades (SNORT, 2016).

Alguns dos profissionais de segurança mais renomados na indústria, incluindo a equipe e os autores de vários livros de referência em padrão de segurança, são membros do Talos. Esta equipe é apoiada pelos vastos recursos do Snort, tornando-se o maior grupo dedicado aos avanços na indústria de segurança de rede. Reflexo disso está nas regras de detecção constantemente atualizadas, assim como o código fonte. Novos padrões são inseridos nas regras

quase na mesma velocidade em que os órgãos responsáveis emitem alertas de ameaças (SNORT, 2016).

3.1.2. Modos de operação

Os modos de operação do Snort (SNORT, 2016) são:

- Modo *Sniffer*: apenas lê os pacotes da rede e apresenta para o usuário de forma contínua na tela.
- Modo *Packet Logger*: registra os pacotes para o disco.
- Modo *Network Intrusion Detection System (NIDS)*: onde o Snort analisa o tráfego na rede e compara com suas regras, executando diversas ações baseadas nelas. Modo mais complexo e configurável.

3.1.3. Arquitetura e funcionamento

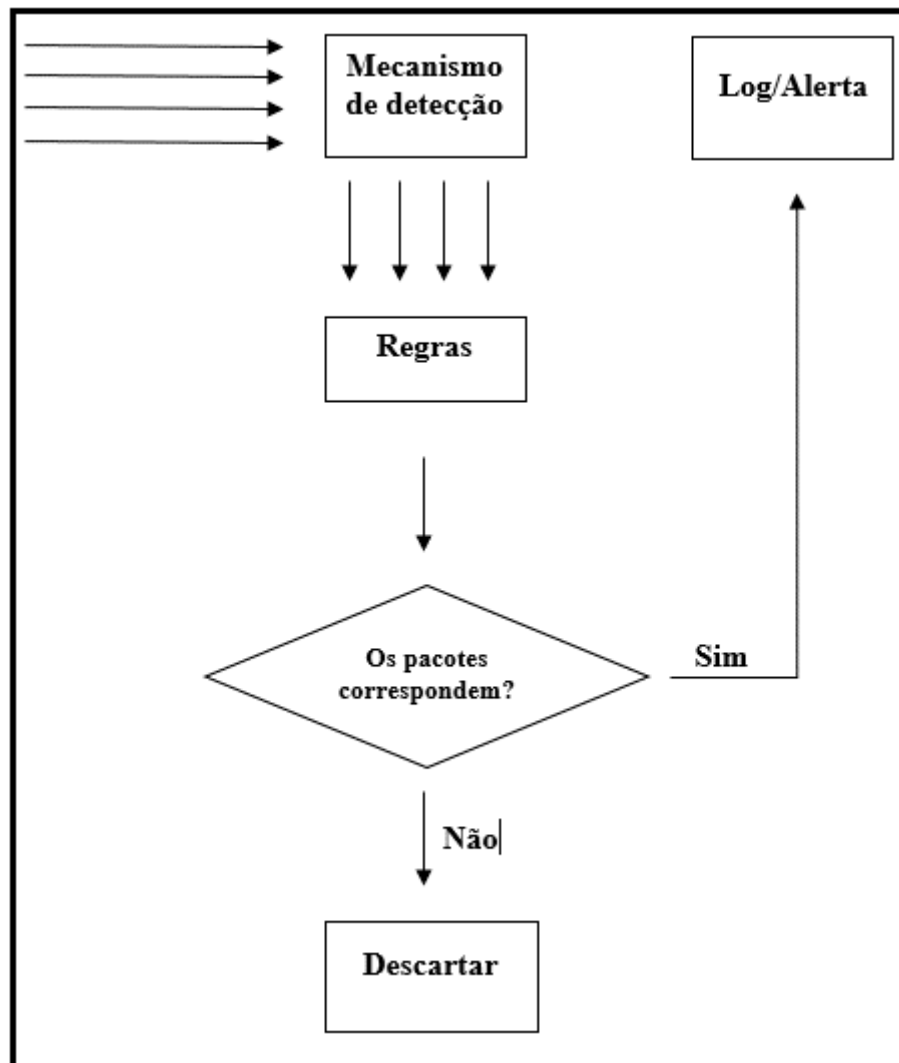
A arquitetura do Snort é focada em desempenho, simplicidade e flexibilidade. É composta por 4 componentes básicos que fazem o processo de decodificação, pré-processamento, detecção, registro e alerta ao administrador (ROESH, 1999).

- **Decodificador de pacotes:** É o *sniffer* da rede, captura os pacotes de diferentes *interfaces* e os prepara para o pré-processamento ou para a etapa de detecção. Após capturados os pacotes, eles são “abertos” e decodificados, então é comparado o protocolo em uso com o comportamento normal para aquele protocolo, o próprio decodificador pode gerar alertas com base na análise dos cabeçalhos e pacotes fora do padrão definido nas regras (REHMAN, 2003; BEALE *et al.*, 2004).
- **Pré-processador:** O pré-processador permite analisar os dados de diferentes maneiras antes da detecção, ele pode ajustar os dados para que chegue da melhor forma para o mecanismo de detecção e possa ser comparado com as regras. Muitos ataques modernos dependem da substituição de dados em sobreposição de fragmentos. Quando o pacote é grande, ele é fragmentado e enviado em partes para o destino, então a assinatura é enviada em partes também. Acontece que o *IDS* necessita da assinatura completa para

comparar com as regras e detectar o pacote malicioso, então o pré-processador pode fazer a desfragmentação e montar novamente o pacote, preparando-o para o mecanismo de detecção (REHMAN, 2003; BEALE *et al.*, 2004).

- **Mecanismo de detecção:** Após os pacotes serem remontados pelo pré-processador, ele passa para o mecanismo de detecção, que é responsável por detectar alguma atividade de intrusão. Para isso, as estruturas de dados internas leem todas as regras e comparam com todos os pacotes. Ao comparar o pacote com a regra, se for detectado uma intrusão, é gerado um *log* ou alerta, caso contrário é descartado. Esta é a fase que mais ocupa recursos do *hardware*, então podemos ter detecções atrasadas, não sendo mais em tempo real. Esse tempo vai depender de vários fatores, como o número de regras, processamento da máquina em que está rodando o Snort, fluxo na rede e outros (REHMAN, 2003). O mecanismo de detecção do SNORT pode ser visto na Figura 6.
- **Registro e Alerta:** O mecanismo de registro no Snort tem a função de arquivar os pacotes que foram reconhecidos na comparação com as regras, enquanto o mecanismo de alerta é responsável por avisar o administrador que uma regra foi acionada. Os registros e alertas dentro da ferramenta são configuráveis e o usuário pode especificar quais serão habilitados. Dessa forma, é possível filtrar os dados que realmente importam dentro do contexto da rede em que se encontra. Existe uma grande variedade de opções para enviar os alertas e como registrar seus pacotes. Exemplos de alertas são através de janelas de pop-up para uma estação de trabalho Windows, um arquivo de log, sockets Unix ou ainda alertas SNMP. Para armazenar os registros é possível utilizar um banco de dados *SQL*, como o *MySQL* ou *PostgreSQL* (BEALE *et al.*, 2004).

Figura 6 - Mecanismo de detecção do Snort.



Fonte: Adaptado de Baker (BAKER, 2007).

3.1.4. Regras

Uma das funções que mais chamam a atenção no Snort é a capacidade dos seus usuários escreverem suas próprias regras, além da grande base de regras que vem por padrão com a ferramenta. Dessa forma, o usuário não depende de fornecedores ou agências externas quando surgem novos ataques, pode escrever suas próprias regras para eliminar a vulnerabilidade e depois compara-las com a comunidade oficial.

Uma regra é definida como um conjunto de instruções projetadas para analisar o tráfego de rede de acordo com um padrão especificado. Após identificado esse padrão efetua uma ação previamente definida (BEALE *et al.*, 2004).

Segundo (REHMAN, 2003), a estrutura das regras do Snort é dividida em duas partes lógicas: cabeçalho da regra e opções da regra. Como pode ser visto na Figura 7.

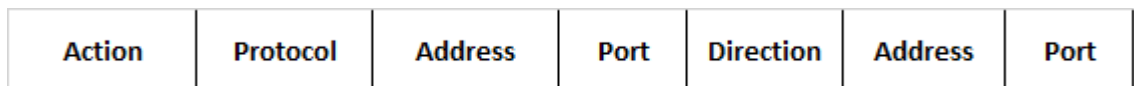
Figura 7 - Estrutura básica das regras do Snort



Fonte: Rehman (REHMAN, 2003).

- **Cabeçalho da regra:** O cabeçalho é geralmente considerado a parte principal das regras, uma vez que ele identifica a ação que deve ser tomada quando a regra é sinalizada. Na Figura 8 podemos visualizar os campos que fazem parte do cabeçalho e uma breve descrição (REHMAN, 2003).

Figura 8 - Estrutura do cabeçalho de regra do Snort.



Fonte: Rehman (REHMAN, 2003).

Action- Ação que deve ser tomada quando a regra é sinalizada, geralmente um alerta.

Protocol- Filtra um determinado protocolo em que a regra deve ser aplicada, por exemplo o protocolo IP, ICMP, UDP.

Address- Possui dois campos, um para o endereço de origem, e outro para o endereço de destino para o pacote em que a regra é aplicada.

Port- Possui dois campos, um para porta de origem e outro para a porta de destino do pacote em que a regra é aplicada.

Direction- Este campo define a direção da regra, no caso de utilização na regra de “->”, indica que origem e destino são da esquerda para a direita.

- **Opções da regra:** Os campos de opções de regra são colocados dentro de parênteses, e são separadas por ponto e vírgula, onde a ação do cabeçalho é ativada, apenas se todas as condições dentro dos parênteses forem atendidas. Na figura 9 temos um exemplo de regra com cabeçalho e opções, que quando ativada emite um alerta ao detectar um pacote de ping *ICMP* com *TTL* igual a 100.

Na Figura 9 podemos visualizar um exemplo de regra dentro do Snort. As regras dentro do *IDS Snort* são conhecidas pela facilidade de criação, assim como a seu uso e personalização (REHMAN, 2003).

Figura 9 - Exemplo de regra no Snort.

```
alert icmp any any -> any any (msg: "Ping with TTL=100"; ttl: 100;)
```

Fonte: Rehman (REHMAN, 2003).

3.2. WEKA

Weka é um conjunto de ferramentas de pré-processamento de dados e algoritmos de aprendizagem de máquina. O software é *open source* e foi desenvolvido na universidade de Waikato, na Nova Zelândia (WITTEN e FRANK, 2005).

O nome *Weka* significa *Waikato Environment for Knowledge Analysis* ou Waikato ambiente para análise do conhecimento. O sistema é escrito em Java e distribuído sob os termos da GNU.

O *Weka* fornece amplo suporte para todo o processo de *Data Mining* experimentais, incluindo preparar os dados de entrada, avaliar os esquemas de aprendizagem estatisticamente, visualizar os dados de entrada e o resultado da aprendizagem. Com o *Weka* é possível pré-processar um *dataset*, executá-lo em um esquema de aprendizagem e analisar o classificador resultante e o seu desempenho. Esse conjunto de ferramentas possui métodos para os principais problemas de *Data Mining*: regressão, classificação, clusterização, mineração de regras de associação e seleção de atributo.

O software *Weka* possui uma interface gráfica do usuário chamada *explorer*, que dá acesso a todas as suas instalações usando a seleção de menu e preenchimento de formulários. Você pode rapidamente ler um *dataset* de um arquivo ARFF (ou planilha) e construir uma árvore de decisão dele. Existem muitos algoritmos para você explorar e a *interface* lhe ajuda na seleção de cada um (WITTEN e FRANK, 2005).

4. TRABALHOS RELACIONADOS

Este capítulo apresenta alguns trabalhos encontrados na literatura que abordam propostas de IDS, utilizando técnicas de *Data Mining*. Como em Panda *et al.* (PANDA e PATRA, 2008) onde os autores apresentam uma proposta de *Data Mining* para que seja integrado a um IDS.

O trabalho citado utiliza a tarefa de classificação e os algoritmos de árvore de decisão ID3, J48 e o algoritmo de Bayes para estimar o desempenho. Para os testes os autores utilizaram o dataset KDDCup'99, com 10% do seu conteúdo dividido nas quatro categorias de ataque: *Probing*, *DoS*, *U2R* e *R2L*.

Os dados filtrados foram inseridos na ferramenta de *Data Mining Weka*, e testado a eficiência dos 3 algoritmos com validação cruzada. Dessa forma é dividido o conjunto de dados em subconjuntos de mesmo tamanho, onde um desses subconjuntos é utilizado para o teste e o restante para gerar os parâmetros, dessa forma é calculado a acurácia do modelo.

Em Senthilnayaki *et al.* (SENTHILNAYAKI *et al.*, 2013) é utilizado um algoritmo de árvore de decisão modificado, baseado no algoritmo J48, melhorando o existente, segundo os autores. Para testes também optaram pelo dataset *KDDCup'99*, selecionando 10% dos dados totais contidos.

No trabalho proposto foi observado a redundância de dados provenientes do *dataset*, então na fase de pré-processamento, são removidas essas redundâncias. Após o algoritmo seleciona 9 características importantes, dentro de 42 presentes e com o resultado os dados são classificados como normal ou como anomalia. Se classificados como anomalia, são categorizados dentro de uma das 4 classes de ataque (*Probing*, *DoS*, *U2R* e *R2L*).

No estudo apresentado por Campos *et al.* (CAMPOS *et al.*, 2012), é proposto um modelo teórico de IDS, utilizando *Data Mining*. Inicialmente foi utilizado dois classificadores, J48 e Redes Neurais MLP.

O *dataset* utilizado foi o *KDDCup'99*, também foi observado pelo autor do trabalho citado a redundância dos dados, que são muito prejudiciais a ataques da categoria *U2R*, impedindo o aprendizado de registros menos frequentes por parte do algoritmo. Então modificou-se o *dataset* visando reduzir o número de atributos, passando de 42 para 27.

Segundo os autores, os testes primeiramente foram feitos com os dois algoritmos anteriormente citados e considerando duas classes de detecção, normal e intrusão, dentro da ferramenta *Weka*. Após, efetuaram os mesmos testes, mas desta vez utilizando os

classificadores J48 e Redes Bayesiana, considerando 5 classes de ataques: Normal, *Probing*, *DoS*, *U2R* e *R2L*.

Após analisar os trabalhos relacionados, é possível fazer algumas comparações com o presente trabalho e posteriormente algumas considerações. Na Tabela 7 podemos visualizar as principais diferenças entre este trabalho e os trabalhos mencionados.

Tabela 7 - Comparativo entre trabalhos estudados e o trabalho proposto.

Trabalho	KDDCup'99	Data Mining	SNORT	Implementação
Panda <i>et al.</i>	✓	✓	✗	✗
Senthilnayaki <i>et al.</i>	✓	✓	✗	✗
Campos <i>et al.</i>	✓	✓	✗	✗
Este Trabalho	✓	✓	✓	✓

Fonte: Autor.

- i) Os trabalhos apresentados generalizam as detecções, não sendo analisado em nenhum momento um ataque específico. Fato que ocorre no trabalho desenvolvido, onde o foco de detecção é para o ataque *SYN Flood*.
- ii) Todos os trabalhos estudados geram padrões para os ataques, e o teste dos padrões gerados acontece dentro da ferramenta Weka. Apenas este trabalho implementa esses padrões em um IDS e os testa em um ambiente real.
- iii) A utilização do IDS SNORT trabalhando de forma integrada a um IDS desenvolvido é proposta apenas neste trabalho.

5. TRABALHO DESENVOLVIDO

Neste capítulo é apresentado o trabalho desenvolvido, dividido em 3 grandes etapas: (i) geração dos padrões do ataque, (ii) log de pacotes suspeitos (iii) detecção do ataque. Cada etapa do trabalho foi descrita de forma detalhada nas subseções deste capítulo.

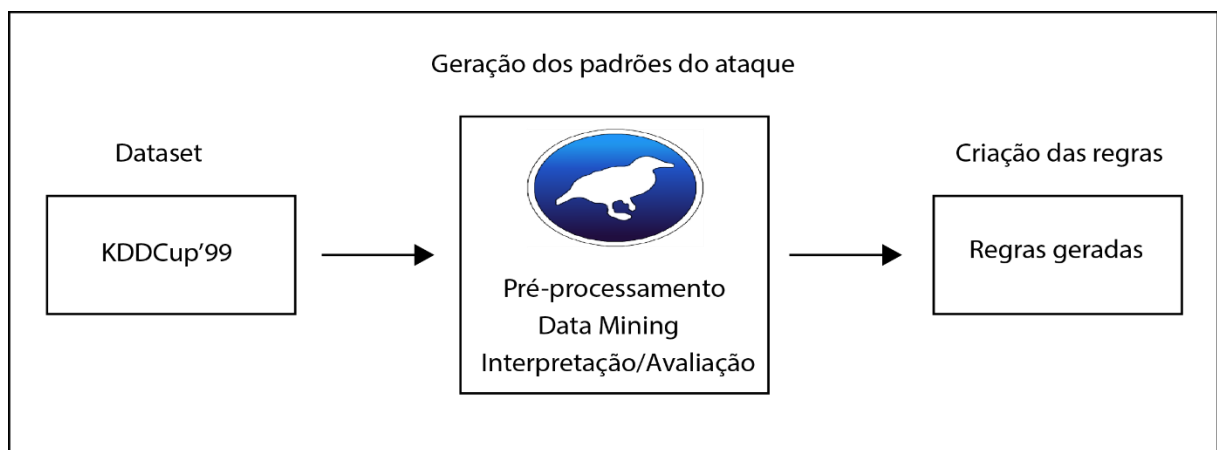
5.1. Geração dos padrões do ataque

Inicialmente a proposta deste trabalho era gerar os padrões para 3 diferentes tipos de ataques DoS, utilizando algoritmos de classificação. Durante a implementação, percebeu-se a complexidade envolvida nas etapas de descoberta do conhecimento, e para que cada etapa fosse efetuada da melhor forma, foi mantido apenas um ataque como alvo do estudo, que está entre aqueles de maior ocorrência na categoria *DoS*.

Ainda durante a implementação, percebeu-se também que os algoritmos *Random Forest* e *Naive Bayes*, não geravam uma árvore de decisão dentro da ferramenta *Weka*, necessária para a criação das regras. Dessa forma optou-se pela escolha de outro algoritmo, que atendia as necessidades do trabalho.

Esta etapa é responsável pela geração dos padrões para o ataque *DoS SYN Flood*. Esses padrões foram gerados a partir da mineração dos dados contidos no *dataset KDDCup'99*, utilizando os algoritmos de *Data Mining* (*J48* e *Random Tree*), dentro da ferramenta *Weka*. Os dados do *dataset KDDCup'99* passaram pelo processo da *KDD* de pré-processamento, mineração e interpretação/avaliação. A arquitetura de geração das regras pode ser vista na Figura 10.

Figura 10 - Arquitetura de geração das regras.



Fonte: Autor.

5.1.1. Pré-processamento

Nesta parte do processo de descoberta do conhecimento, acontece o tratamento dos dados. São filtrados os registros relevantes para o trabalho e verificada a relevância de cada atributo para o ataque, assim como a correlação e a similaridade entre eles.

O *KDDCup'99* é fornecido em formato texto, onde em um arquivo contém os dados capturados separados por vírgula. Com a necessidade do pré-processamento, foi criada uma base chamada “*dataset*” dentro do *Microsoft SQL Server*, com uma tabela nomeada “ataques”, contendo todos os atributos do *dataset*.

Foi verificado que apenas um dos atributos não apresentava o nome, e os dados da respectiva coluna constavam o nome do ataque. Dessa forma, o atributo em questão foi nomeado como “*attack*” e após os dados foram importados para a base.

Inicialmente no *dataset*, continham 42 atributos e 494.021 registros com dados coletados de tráfego. Esses atributos podem ser vistos na Figura 11.

Figura 11 – Tabela attack com os atributos do KDDCup'99.

Column Name	Data Type
duration	numeric(18,0), null
protocol_type	varchar(50), null
service	varchar(50), null
flag	varchar(50), null
src_bytes	numeric(18,0), null
dst_bytes	numeric(18,0), null
land	numeric(18,0), null
wrong_fragment	numeric(18,0), null
urgent	numeric(18,0), null
hot	numeric(18,0), null
num_failed_logins	numeric(18,0), null
logged_in	numeric(18,0), null
num_compromised	numeric(18,0), null
root_shell	numeric(18,0), null
su_attempted	numeric(18,0), null
num_root	numeric(18,0), null
num_file_creations	numeric(18,0), null
num_shells	numeric(18,0), null
num_access_files	numeric(18,0), null
num_outbound_cmds	numeric(18,0), null
is_host_login	numeric(18,0), null
is_guest_login	numeric(18,0), null
count	numeric(18,0), null
srv_count	numeric(18,0), null
error_rate	float, null
srv_error_rate	float, null
error_rate	float, null
srv_error_rate	float, null
same_srv_rate	float, null
diff_srv_rate	float, null
srv_diff_host_rate	float, null
dst_host_count	numeric(18,0), null
dst_host_srv_count	numeric(18,0), null
dst_host_same_srv_rate	float, null
dst_host_diff_srv_rate	float, null
dst_host_same_src_port_rate	float, null
dst_host_srv_diff_host_rate	numeric(18,0), null
dst_host_error_rate	numeric(18,0), null
dst_host_srv_error_rate	numeric(18,0), null
dst_host_error_rate	numeric(18,0), null
dst_host_srv_error_rate	numeric(18,0), null
attack	varchar(50), null

Fonte: Autor.

O objetivo era gerar padrões para diferenciar os dados de tráfego normal e de ataques *SYN Flood*. Como os dados de outros ataques não eram necessários, foi filtrado então o atributo *attack* igual a “neptune” ou “normal”, obtendo 204.479 registros.

Após restringir os pacotes apenas aos de foco do trabalho, foram analisados os 42 atributos do dataset, onde verificou-se a semelhança de valores entre cada atributo. Como o objetivo na geração dos padrões era diferenciar o normal de ataque, atributos com valores iguais foram considerados irrelevantes.

Atributos onde o valor só pode ser definido ao final do ataque, também não foram considerados. A necessidade de detecção em tempo real os torna sem importância, assim como os atributos que não são relacionados com as características de um ataque *DoS*.

Feita a análise descrita, restaram 6 atributos relevantes. Os atributos foram divididos de acordo com a classificação do *KDDCup'99*, que são: (I) características individuais de uma conexão *TCP*: *duration*, *protocol_type*, *src_bytes*, (II) características de tráfego (utilizando 2 segundos de intervalo): *count*, *srv_count* e (III) identificação do ataque: *attack*. A descrição de cada um dos atributos pode ser vista na Tabela 8.

Tabela 8 - Descrição dos atributos do *KDDCup'99*.

Atributo	Descrição
<i>duration</i>	Duração (em segundos) da conexão.
<i>protocol_type</i>	Tipo do protocolo.
<i>src_bytes</i>	Tamanho do pacote da origem para o destino.
<i>count</i>	Número de conexões para o mesmo host no intervalo de 2 segundos.
<i>srv_count</i>	Número de conexões para o mesmo serviço no intervalo de 2 segundos.
<i>attack</i>	Rótulo do ataque.

Fonte: Autor.

Para executar a etapa de *Data Mining* e aplicar os algoritmos aos dados, ainda no pré-processamento, foi necessário criar o *dataset* com os atributos e dados do *KDDCup'99* na extensão *.arff*. Esse que é o formato de leitura dentro da ferramenta *Weka*, possuindo uma sintaxe própria e estrutura dividida em cabeçalho e dados, como pode ser visto no trecho a seguir.

```

@relation ataques

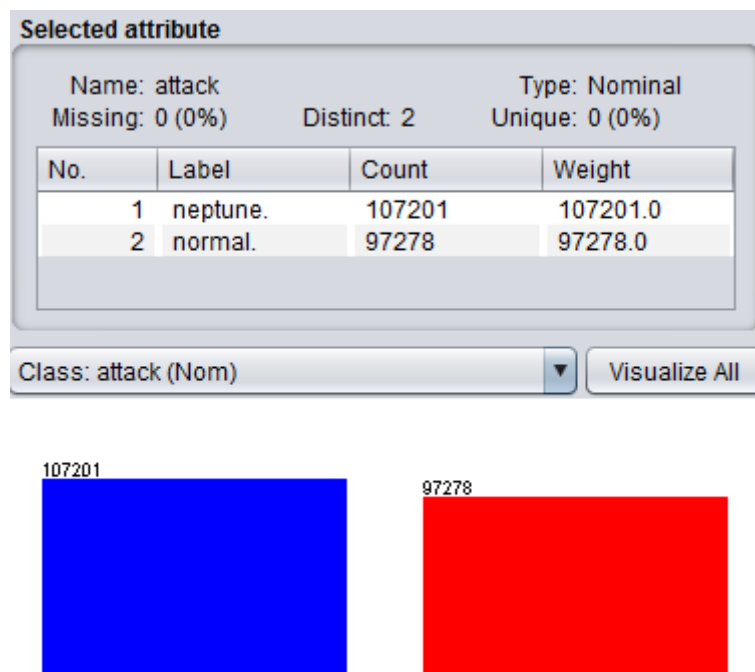
@attribute duration numeric
@attribute protocol_type {tcp,udp,icmp}
@attribute src_bytes numeric
@attribute count numeric
@attribute srv_count numeric
@attribute attack {neptune.,normal.}

@data
0,tcp,194,8,8,normal.
0,tcp,206,18,18,normal.
0,tcp,286,3,9,normal.
0,tcp,216,5,5,normal.
0,tcp,263,10,10,normal.

```

Após a criação do dataset, ele foi executado dentro da ferramenta *Weka*, onde os dados passaram por uma última alteração na etapa de pré-processamento. Os dados totais eram 204.479 registros, sendo 107.201 pacotes *neptune* e 97.278 normais, como pode ser visto na Figura 12.

Figura 12 - Atributo attack após execução na ferramenta Weka.



Fonte: Autor.

Para que as regras criadas não fossem tendenciadas, devido as diferenças nas quantidades das amostras, os dados passaram por um balanceamento. Foi utilizado então o

algoritmo SpreadSubsample (ARORA, 2013) dentro da ferramenta, responsável por deixar a amostra igual para o atributo classificado, de acordo com a quantidade de registros desejada.

Para o trabalho apresentado, utilizou-se como parâmetro 97.000 registros para cada normal e *SYN Flood*, arredondando os dados totais. Após a execução do algoritmo, a quantidade passou a ser igual para ambos, finalizando a etapa de pré-processamento. Os dados prontos para a etapa de *Data Mining* podem ser vistos na Figura 13.

Figura 13 - Dados prontos para o *Data Mining*.

The screenshot shows the Weka Explorer interface with the 'Preprocess' tab selected. The 'Filter' section shows 'SpreadSubsample -M 0.0 -X 97000.0 -S 1' applied. The 'Current relation' shows 6 attributes and 194000 instances. The 'Selected attribute' section shows the 'attack' attribute with a table of counts for 'neptune' and 'normal' classes, both at 97000. A bar chart below shows two bars of height 97000, one blue and one red. The 'Attributes' list shows 'attack' selected. The 'Status' bar shows 'OK'.

Current relation
 Relation: ... Attributes: 6
 Instances: ... Sum of weights: 194000

Selected attribute
 Name: attack Type: No...
 Missing: 0 ... Distinct: . Unique: 0 (0...)

No.	Label	Count	Weight
1	neptune.	97000	97000.0
2	normal.	97000	97000.0

Attributes

No.	Name
1	<input type="checkbox"/> duration
2	<input type="checkbox"/> protocol_type
3	<input type="checkbox"/> src_bytes
4	<input type="checkbox"/> count
5	<input type="checkbox"/> sv_count
6	<input checked="" type="checkbox"/> attack

Status
 OK Log x 0

Fonte: Autor.

5.1.2. *Data Mining*

A etapa da *KDD* de *Data Mining* é responsável pela geração dos padrões a partir dos dados. Com o *dataset* preparado dentro da ferramenta *Weka* para receber a mineração, selecionou-se a opção de testes das regras geradas. A opção escolhida foi “*Use training set*”, onde todos os dados do *dataset* passam pelas regras criadas, e é verificado se eles foram classificados corretamente.

Os algoritmos de classificação *J48* e *Random Tree* foram executados, gerando os padrões para o atributo *attack*. Esses algoritmos foram escolhidos, devido aos resultados gerados por eles, ser uma árvore de decisão, facilitando na criação das regras.

Na execução do algoritmo *J48*, os testes indicaram 193.903 registros classificados corretamente, e apenas 97 incorretamente. Em porcentagem de acerto na classificação, o algoritmo teve 99,95% de acertos, e apenas 0,05% erros.

Para o algoritmo *Random Tree*, houveram resultados semelhantes, com 193.915 registros classificados corretamente e 85 incorretamente. O percentual de acertos e erros se manteve, devido a pequena diferença entre os dados incorretamente classificados para o tamanho da amostra. O que mudou consideravelmente para os dois algoritmos, foi a árvore de decisão gerada. A árvore do *J48* ficou com 29 nodos, enquanto a do *Random Tree* com 101. A árvore de decisão gerada pelo algoritmo *J48* pode ser vista na Figura 14, e a do algoritmo *Random Tree* pode ser vista na Figura 15 e Figura 16.

Figura 14 - Árvore gerada pelo algoritmo J48.

```

count <= 54
| src_bytes <= 0
| | count <= 2
| | | count <= 1: normal. (5342.0/47.0)
| | | count > 1
| | | | srv_count <= 1: neptune. (28.0/1.0)
| | | | srv_count > 1: normal. (160.0/13.0)
| | count > 2
| | | srv_count <= 18
| | | | count <= 18
| | | | | srv_count <= 12: neptune. (662.0/28.0)
| | | | | srv_count > 12
| | | | | | count <= 15
| | | | | | | srv_count <= 14
| | | | | | | | count <= 12: neptune. (2.0)
| | | | | | | | count > 12: normal. (9.0/2.0)
| | | | | | | | srv_count > 14: normal. (4.0)
| | | | | | | | count > 15: neptune. (16.0/5.0)
| | | | | count > 18: neptune. (1141.0)
| | | | srv_count > 18
| | | | | srv_count <= 21
| | | | | | count <= 18: normal. (2.0)
| | | | | | count > 18: neptune. (15.0)
| | | | | | srv_count > 21: normal. (17.0/1.0)
| src_bytes > 0: normal. (90335.0)
count > 54
| src_bytes <= 6: neptune. (95107.0)
| src_bytes > 6: normal. (1160.0)

```

Fonte: Autor.

5.1.3. Interpretação/Avaliação

A etapa de interpretação e avaliação dos dados, é responsável pela análise dos resultados obtidos com o *Data Mining*. A árvore de decisão gerada, nos mostra o comportamento dos dados para o ataque *SYN Flood* e para o tráfego normal, separando cada um dos registros nos padrões que os identificam.

Dessa forma, é necessário analisar os padrões mais relevantes. Para fins de estudo, levou-se em conta, os 3 padrões que obtiveram o maior número de registros selecionados como ataque. Essa análise foi feita separadamente para cada algoritmo.

- **J48**

Regra 1:

```
count <= 54
| src_bytes <= 0
| | count > 2
| | | count <= 18
| | | | srv_count <= 12: neptune. (662.0/28.0) →
```

Classificações corretas/incorretas

Interpretação: se houver um número total de pacotes menor ou igual a 54, e um número maior do que 2 e menor ou igual a 18 pacotes, com tamanho menor ou igual a 0, enviados de um mesmo *host* para o *host* monitorado, desde que esse *host* possua 12 ou menos conexões para um mesmo serviço, em um intervalo de 2 segundos, é um ataque *SYN Flood*.

Regra 2:

```
count <= 54
| src_bytes <= 0
| | srv_count <= 18
| | | | count > 18: neptune. (1141.0) →
```

Classificações corretas/incorretas

Interpretação: se houver um número menor ou igual a 54 pacotes totais, e um número maior que 18 pacotes de tamanho menor ou igual a 0, enviados de um mesmo *host* para o *host* monitorado, desde que esse *host* possua 18 ou menos conexões para um mesmo serviço, em um intervalo de 2 segundos, é um ataque *SYN Flood*.

Regra 3:

```
count > 54
| src_bytes <= 6: neptune. (95107.0) →
```

Classificações corretas/incorretas

Interpretação: se houver um número maior do que 54 pacotes totais, com tamanho menor ou igual a 6, enviados de um mesmo *host* para o *host* monitorado, em um intervalo de 2 segundos, é um ataque *SYN Flood*.

- **Random Tree**

Regra 1:

```
count < 54.5
| src_bytes < 0.5
| | srv_count < 6.5
| | | srv_count >= 3.5
| | | | count >= 6.5 : neptune. (375/0) →
```

Classificações corretas/incorretas

Interpretação: se houver um número menor que 54.5 pacotes totais, e um número maior ou igual a 6.5 pacotes, com tamanho menor ou igual a 0.5, enviados de um mesmo *host* para o *host* monitorado, desde que esse *host* possua 3.5 ou mais e menos do que 6.5 conexões para um mesmo serviço, em um intervalo de 2 segundos, é um ataque *SYN Flood*.

Regra 2:

```
count < 54.5
| src_bytes < 0.5
| | srv_count < 12.5
| | | srv_count >= 6.5
| | | | count >= 11.5 : neptune. (640/0) →
```

Classificações corretas/incorretas

Interpretação: se houver um número menor que 54.5 pacotes totais, e um número maior ou igual a 11.5 pacotes, com tamanho menor ou igual a 0, enviados de um mesmo *host* para o *host* monitorado, desde que esse *host* possua 6.5 ou mais e menos do que 12.5 conexões para um mesmo serviço, em um intervalo de 2 segundos, é um ataque *SYN Flood*.

Regra 3:

```
count >= 54.5
| src_bytes < 6 : neptune. (95107/0) →
```

Classificações corretas/incorretas

Interpretação: Se houver um número maior ou igual a 54.5 pacotes totais, com tamanho menor a 6, enviados de um mesmo *host* para o *host* monitorado, em um intervalo de 2 segundos, é um ataque *SYN Flood*.

A regra 3, a mais relevante entre as regras para o algoritmo J48, classificou 95.107 registros *SYN Flood*, ou seja 98.04% dos dados totais para o ataque. A regra 3, a mais relevante entre as regras do algoritmo *Random Tree*, obteve o mesmo número de classificações e o mesmo percentual, isso devido as regras serem muito semelhantes.

As outras regras selecionadas e apresentadas na análise, obtiveram um percentual menor do que 1% e poderiam ser desconsideradas. Para fins de testes e visualização das diferenças entre um padrão mais relevante e um menos relevante, decidiu-se mantê-las. Com as regras criadas, terminamos a etapa do processo de geração dos padrões do ataque.

5.2. Log de pacotes suspeitos

Esta etapa é responsável pelo monitoramento do *host*, que pode ser alvo de um ataque *SYN Flood*. O *host* é monitorado pela ferramenta Snort em modo *NIDS*, onde cada pacote recebido por ele, é verificado e determinado como suspeito ou não.

Essa verificação em cada pacote recebido, é efetuada através de regras criadas dentro da ferramenta. Essas regras são apresentadas a seguir, explicando-as de forma detalhada.

- **Regras para o algoritmo J48 dentro do SNORT:**

```
alert tcp any any -> $HOME_NET any (msg:"zero";GID:1;sid:1000001;
rev:001;dsiz<1;)
```

Função: um alerta é gerado se o *host* monitorado receber um pacote *TCP* de tamanho 0.

```
alert udp any any -> $HOME_NET any (msg:"zero";GID:1;sid:1000002;
rev:001;dsiz<1;)
```

Função: um alerta é gerado se o *host* monitorado receber um pacote *UDP* de tamanho 0.

```
alert icmp any any -> $HOME_NET any (msg:"zero";GID:1;sid:10000003;
rev:001;dsiz<1;)
```

Função: um alerta é gerado se o *host* monitorado receber um pacote *ICMP* de tamanho 0.

```
alert tcp any any -> $HOME_NET any (msg:"menor6";GID:1;sid:10000004;
rev:001;dsiz<1<>6;)
```

Função: um alerta é gerado se o *host* monitorado receber um pacote *TCP* de tamanho entre 1 e 6.

```
alert udp any any -> $HOME_NET any (msg:"menor6";GID:1;sid:10000005;
rev:001;dsiz<1<>6;)
```

Função: um alerta é gerado se o *host* monitorado receber um pacote *UDP* de tamanho entre 1 e 6.

```
alert icmp any any -> $HOME_NET any (msg:"menor6";GID:1;sid:10000006;
rev:001;dsiz<1<>6;)
```

Função: um alerta é gerado se o *host* monitorado receber um pacote *ICMP* de tamanho entre 1 e 6.

- **Regras para o algoritmo *Random Tree* dentro do SNORT:**

```
alert tcp any any -> $HOME_NET any (msg:"zero";GID:1;sid:10000001;
rev:001;dsiz<1;)
```

Função: um alerta é gerado se o *host* monitorado receber um pacote *TCP* de tamanho 0.

```
alert udp any any -> $HOME_NET any (msg:"zero";GID:1;sid:10000002;
rev:001;dsiz<1;)
```

Função: um alerta é gerado se o *host* monitorado receber um pacote *UDP* de tamanho 0.

```
alert icmp any any -> $HOME_NET any (msg:"zero";GID:1;sid:10000003;
rev:001;dsiz<1;)
```

Função: um alerta é gerado se o *host* monitorado receber um pacote *ICMP* de tamanho 0.


```
alert tcp any any -> $HOME_NET any (msg:"menor6";GID:1;sid:10000004;
rev:001;dsiz:1<>5;)
```

Função: um alerta é gerado se o host monitorado receber um pacote *TCP* de tamanho entre 1 e 5.

```
alert udp any any -> $HOME_NET any (msg:"menor6";GID:1;sid:10000005;
rev:001;dsiz:1<>5;)
```

Função: um alerta é gerado se o host monitorado receber um pacote *UDP* de tamanho entre 1 e 5.

```
alert icmp any any -> $HOME_NET any (msg:"menor6";GID:1;sid:10000006;
rev:001;dsiz:1<>5;)
```

Função: um alerta é gerado se o host monitorado receber um pacote *ICMP* de tamanho entre 1 e 5.

Como pode ser visto, as regras dentro do Snort para cada algoritmo são iguais em alguns casos e muito semelhantes em outros. Os padrões do algoritmo J48, tem como tamanho dos pacotes envolvidos em suas regras entre 0 e 6 bytes, gerando um alerta quando tamanho é 0 e outro alerta quando o tamanho está entre 1 e 6 bytes.

Foram criadas então regras semelhantes para o algoritmo *Random Tree*. Não foi possível utilizá-las para ambos, devido aos pacotes nas regras do *Random Tree* possuírem tamanhos entre 0 e 5 bytes, gerando um alerta quando tamanho é 0 e outro alerta quando o tamanho está entre 1 e 5 bytes.

Essas regras estão em arquivos separados, nomeados como “j48.rules” e “randomtree.rules”. Os arquivos são utilizados de acordo com o algoritmo escolhido para detecção.

Após gerado o alerta por uma das regras, a ferramenta cria um log dentro de um arquivo chamado “alerts.csv”. A saída de alerta nesse formato, foi atribuída dentro do arquivo principal de configuração da ferramenta “Snort.conf”, como pode ser visto na linha a seguir.

```
output alert_csv: /var/log/SNORT/alerts.csv dstport, msg, src, dst, proto
```

Na linha dentro do arquivo, também é atribuído o caminho em que o arquivo deve ser gerado e populado, seguido dos atributos desejados do pacote capturado. Os atributos

selecionados para a criação de log foram: porta de destino, mensagem, IP de origem, IP de destino e o protocolo utilizado. O arquivo de logs suspeitos gerados pelo Snort, pode ser visto na Figura 17.

Figura 17 - Logs de pacotes suspeitos gerados pelo Snort.

The image shows a window titled "alerts.csv" located at "/var/log/snort". The window contains a list of logs in CSV format. The first line is highlighted with a red box and labeled as "Log de pacote suspeito". The logs consist of alternating entries with "menor6" and "zero" as the message, both originating from IP 195.115.218.108 and destined for 172.16.114.50 over TCP port 23. The status bar at the bottom indicates the file is in CSV format with a tab width of 8, currently on line 1, column 1.

Message	Source IP	Destination IP	Port	Protocol
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP
menor6	195.115.218.108	172.16.114.50	23	TCP
zero	195.115.218.108	172.16.114.50	23	TCP

Fonte: Autor.

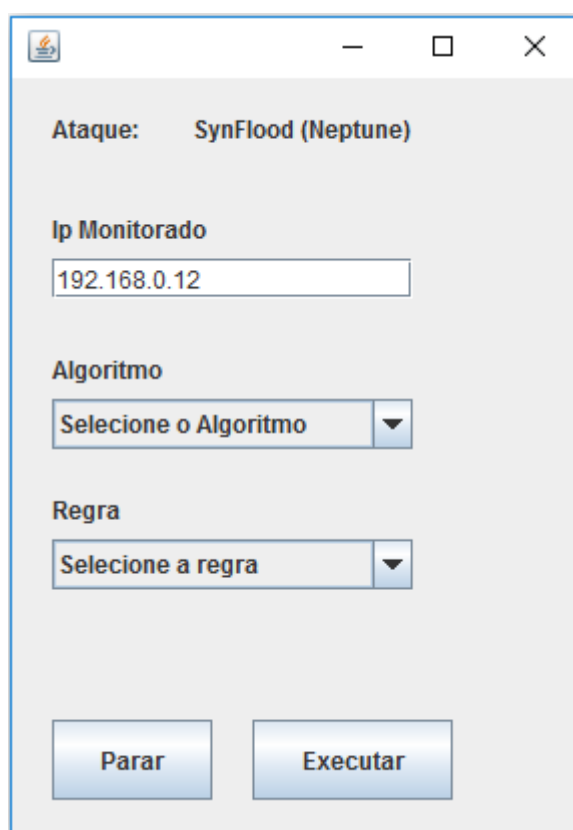
5.3. Detecção do ataque

Esta é a última etapa do desenvolvimento do trabalho, sendo responsável pela detecção do ataque *SYN Flood*. A ideia inicial era que essa detecção fosse efetuada pelo *IDS SNORT*, através de suas regras personalizadas.

Devido a necessidade de verificar todos os *hosts* que estão enviando pacotes para o *host* monitorado, verificar a quantidade de pacotes que esse *host* enviou e o serviço utilizado, tornou-se inviável a utilização do *IDS Snort* para esse fim, sendo ele utilizado para outra etapa, como descrito anteriormente.

Desta forma, optou-se por desenvolver um novo programa para ser utilizado como *IDS*. Esse programa foi desenvolvido na linguagem de programação Java, executada de forma integrada e em paralelo ao *IDS Snort*. Os padrões para detecção do ataque *SYN Flood* gerados na etapa de *Data Mining*, foram criados dentro do programa em forma de métodos. Ao iniciar o *IDS*, é inserido o *IP* do *host* monitorado, selecionado o algoritmo desejado, e então selecionada uma das regras anteriormente citadas, que deve detectar o ataque. Na Figura 18 pode ser vista a *interface* da aplicação.

Figura 18 - Interface da Aplicação Java.



The image shows a Java application window titled "Ataque: SynFlood (Neptune)". The interface includes a text input field for "Ip Monitorado" containing the IP address "192.168.0.12". Below this is a dropdown menu for "Algoritmo" with the text "Selecione o Algoritmo". Another dropdown menu for "Regra" contains the text "Selecione a regra". At the bottom of the window are two buttons: "Parar" and "Executar".

Fonte: Autor.

Para iniciar o monitoramento, é utilizado o botão executar. Antes que o monitoramento inicie efetivamente, o *IDS* desenvolvido seleciona o *IP* do campo "Ip Monitorado", e altera o *IP* contido na variável "ipvar", dentro do arquivo de configuração do *Snort* "snort.conf".

Essa variável é responsável por indicar ao *IDS Snort* o *IP* do *host* monitorado. Em seguida é verificado o algoritmo selecionado, e dentro do mesmo arquivo de configuração, é alterado o arquivo de regras que devem ser utilizadas. Para o algoritmo *J48* é escrito no arquivo "J48.rules", e para o algoritmo *Random Tree* é escrito "randomtree.rules".

Após essa etapa de configuração, é liberado o monitoramento para o *host* dentro do programa desenvolvido. Em seguida o IDS aguarda a execução do Snort, para monitoramento de pacotes suspeitos e geração do arquivo de *logs*.

Os *logs* de pacotes suspeitos são lidos e armazenados pelo programa, e então o arquivo é apagado, em um intervalo de 2 segundos. Esse intervalo de tempo é o informado pelo *KDDCup'99*, da captura dos dados utilizados na geração dos padrões.

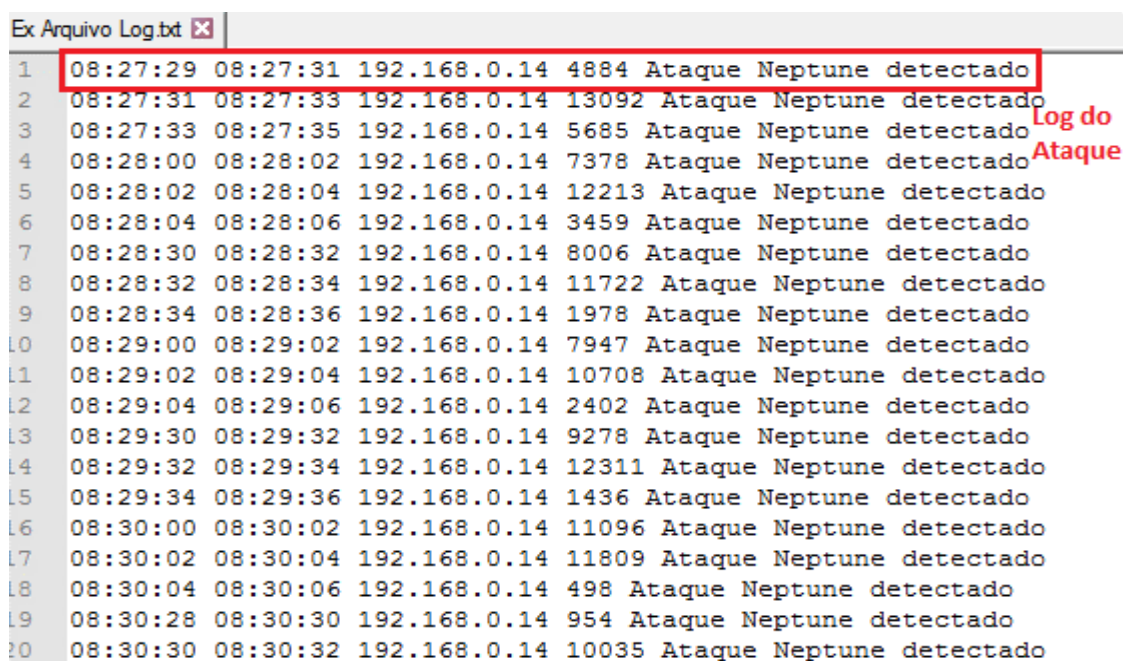
Os dados são armazenados separadamente, de acordo com o *host* que enviou o pacote. Dessa forma é contabilizada a quantidade de pacotes enviados por ele, assim como a quantidade de conexões para cada serviço requerido.

Após, é feita uma chamada para o método que implementa a regra escolhida, onde os dados são verificados. O método retorna verdadeiro caso detecte um ataque *SYN Flood*, emite um alerta e gera um arquivo chamado “logAtaque”, dentro da pasta de logs do IDS Snort.

Esse arquivo contém as seguintes informações: intervalo de ocorrência do ataque, IP do *host* atacante e quantidade de pacotes enviados. Criado o arquivo, a cada novo ataque detectado é incluída uma nova linha dentro dele, com os campos mencionados.

Com a análise do arquivo “logAtaque” é possível verificar o funcionamento do *IDS*, e algum ataque efetuado contra o *host* monitorado. O arquivo de logs gerado pelo IDS desenvolvido, pode ser visto na Figura 19, em seguida na Figura 20 pode ser visualizada a arquitetura de detecção deste trabalho.

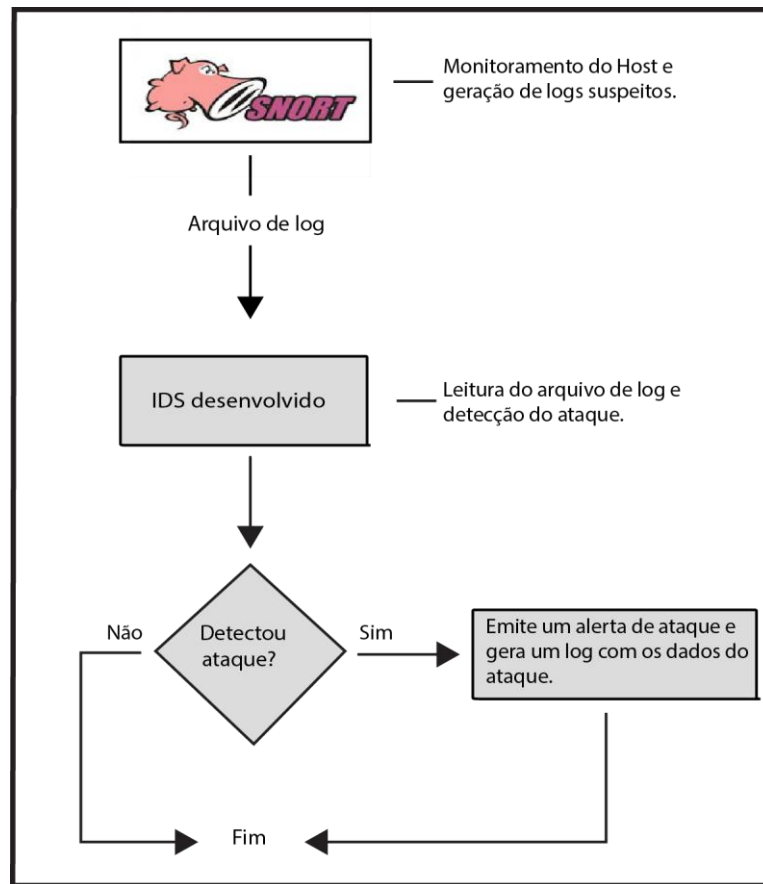
Figura 19 - Arquivo de logs de ataques gerado pelo IDS desenvolvido.



Line	Start Time	End Time	IP	Packets	Attack Name	Status
1	08:27:29	08:27:31	192.168.0.14	4884	Ataque Neptune	detectado
2	08:27:31	08:27:33	192.168.0.14	13092	Ataque Neptune	detectado
3	08:27:33	08:27:35	192.168.0.14	5685	Ataque Neptune	detectado
4	08:28:00	08:28:02	192.168.0.14	7378	Ataque Neptune	detectado
5	08:28:02	08:28:04	192.168.0.14	12213	Ataque Neptune	detectado
6	08:28:04	08:28:06	192.168.0.14	3459	Ataque Neptune	detectado
7	08:28:30	08:28:32	192.168.0.14	8006	Ataque Neptune	detectado
8	08:28:32	08:28:34	192.168.0.14	11722	Ataque Neptune	detectado
9	08:28:34	08:28:36	192.168.0.14	1978	Ataque Neptune	detectado
10	08:29:00	08:29:02	192.168.0.14	7947	Ataque Neptune	detectado
11	08:29:02	08:29:04	192.168.0.14	10708	Ataque Neptune	detectado
12	08:29:04	08:29:06	192.168.0.14	2402	Ataque Neptune	detectado
13	08:29:30	08:29:32	192.168.0.14	9278	Ataque Neptune	detectado
14	08:29:32	08:29:34	192.168.0.14	12311	Ataque Neptune	detectado
15	08:29:34	08:29:36	192.168.0.14	1436	Ataque Neptune	detectado
16	08:30:00	08:30:02	192.168.0.14	11096	Ataque Neptune	detectado
17	08:30:02	08:30:04	192.168.0.14	11809	Ataque Neptune	detectado
18	08:30:04	08:30:06	192.168.0.14	498	Ataque Neptune	detectado
19	08:30:28	08:30:30	192.168.0.14	954	Ataque Neptune	detectado
20	08:30:30	08:30:32	192.168.0.14	10035	Ataque Neptune	detectado

Fonte: Autor.

Figura 20 - Arquitetura de detecção.



Fonte: Autor.

6. RESULTADOS

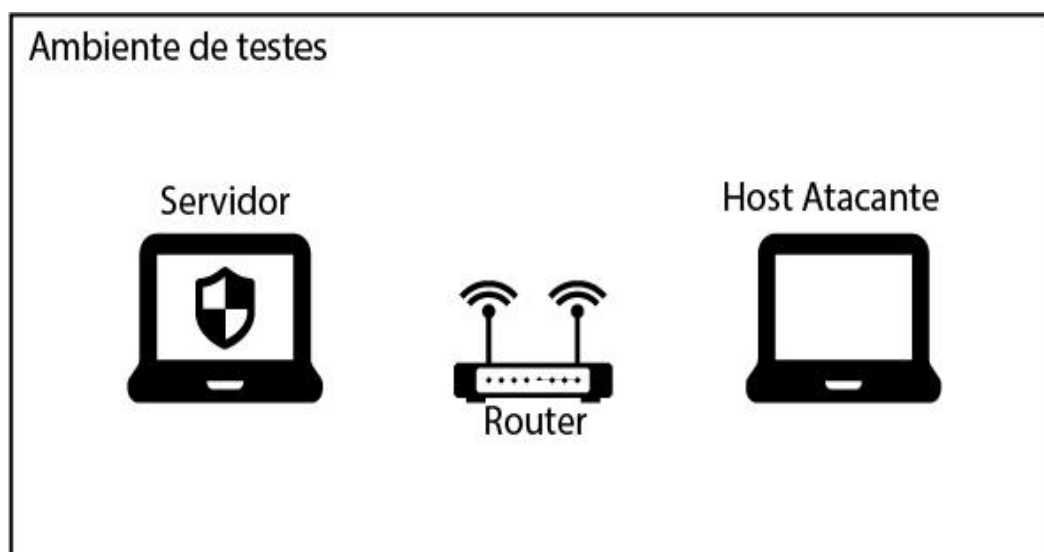
Esta seção apresenta os resultados obtidos com o presente trabalho. Foi montado um ambiente de testes possível de simular ataques *SYN Flood* e sua detecção. Em 6.1 é apresentado o ambiente de testes criado, em 6.2 podem ser vistos os testes realizados para validação do trabalho, 6.3 são apresentados os resultados obtidos e em 6.4 é feita uma análise de desempenho.

6.1. Ambiente de testes

Nesta subseção é apresentado o ambiente criado para a geração dos testes, e então a obtenção dos resultados do trabalho proposto. Para criar o ambiente necessário, foi utilizado um notebook Acer Intel Core I5, 4Gb de memória *RAM*, como *host* monitorado, denominado “Servidor”. Como *host* atacante, foi utilizado um Notebook Samsung Intel Celeron Dual Core, com 4Gb de memória *RAM*.

Os equipamentos foram conectados a um roteador *Wireless* Intelbras WRN 342, ambos conectados pela rede *Wifi*. No servidor foi instalado a distribuição Linux Ubuntu 16.04, o IDS Snort e o IDS desenvolvido. No *host* atacante foi instalado a distribuição Linux Ubuntu 16.04, as ferramentas para ataques DoS Slowloris e Hping3 (MOUSTIS e KOTZANIKOLAOU, 2013; AL-MUSAWI, 2012).

Figura 21 - Ambiente de testes.



Fonte: Autor.

6.2. Testes realizados

Os testes para a validação do trabalho foram realizados de duas formas: i) utilizando os *datasets* Darpa e UNB ISCX 2012 e ii) testes em tempo real utilizando as ferramentas *Slowloris* e *Hping3*.

6.2.1. Testes com datasets

Para a utilização de ambos os *datasets*, Darpa e UNB ISCX 2012, foram necessários alguns ajustes para expressar a realidade. Os arquivos dos *datasets* são no formato .pcap, contendo horas de capturas de pacotes. Esses arquivos são executados em segundos com o *IDS* Snort, dessa forma para a ferramenta, essas horas de tráfego são como segundos de tráfego.

Como mencionado anteriormente, a detecção acontece com a análise dos dados em um intervalo de 2 segundos. Foi necessário então, “abrir” esses *datasets*, utilizando a ferramenta *Wireshark* e filtrar por intervalos de captura. O filtro utilizado pode ser visto na Figura 22.

Figura 22 - Filtro de pacotes no Wireshark de acordo com o intervalo de captura.

No.	Time	Source	Destination	Protocol	Length	Info
1629...	32641.220869	172.16.113.105	194.7.248.153	TELNET	60	Telnet Data ...
1629...	32641.221662	194.7.248.153	172.16.113.105	TELNET	60	Telnet Data ...
1629...	32641.240636	172.16.113.105	194.7.248.153	TCP	60	28825→23 [ACK] Seq=48
1629...	32641.540746	172.16.113.105	194.7.248.153	TELNET	60	Telnet Data ...
1629...	32641.541535	194.7.248.153	172.16.113.105	TELNET	60	Telnet Data ...
1629...	32641.560566	172.16.113.105	194.7.248.153	TCP	60	28825→23 [ACK] Seq=48
1629...	32641.802721	172.16.113.105	194.7.248.153	TELNET	60	Telnet Data ...
1629...	32641.802864	172.16.113.204	195.115.218.108	TELNET	60	Telnet Data ...
1629...	32641.803869	194.7.248.153	172.16.113.105	TELNET	60	Telnet Data ...
1629...	32641.804076	195.115.218.108	172.16.113.204	TELNET	60	Telnet Data ...
1629...	32641.820519	172.16.113.204	195.115.218.108	TCP	60	6953→23 [ACK] Seq=441
1629...	32641.820585	172.16.113.105	194.7.248.153	TCP	60	28825→23 [ACK] Seq=48

▾ Frame 1629564: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
 Encapsulation type: Ethernet (1)
 Arrival Time: Apr 5, 1999 18:04:04.030367000 E. South America Standard Time

Fonte: Autor.

Filtrados os pacotes, eles são selecionados e salvos na extensão .pcap. Após este processo, foram adquiridas 4 amostras de ataque para o *dataset* Darpa, de acordo com o arquivo “*Detection Scoring Truth*”, fornecido dentro do site do *dataset*.

No arquivo são descritos os momentos dos ataques, e utilizando o filtro apresentado na Figura 22, foi possível gerar os arquivos necessários. A pequena quantidade de amostras, se deve ao fato que, alguns momentos informados no arquivo para o ataque *SYN Flood*, não foram encontrados dentro do *dataset*.

Como o *dataset* Darpa fornece o momento dos ataques, poderia se deduzir que um intervalo de tempo filtrado, não constando no arquivo de ataques, seria tráfego normal. Como alguns ataques mencionados não foram encontrados, e para não se basear em deduções, foi decidido utilizar para testes de tráfego normal, o *dataset* UNB ISCX 2012, que possui um arquivo de pacotes normais separado.

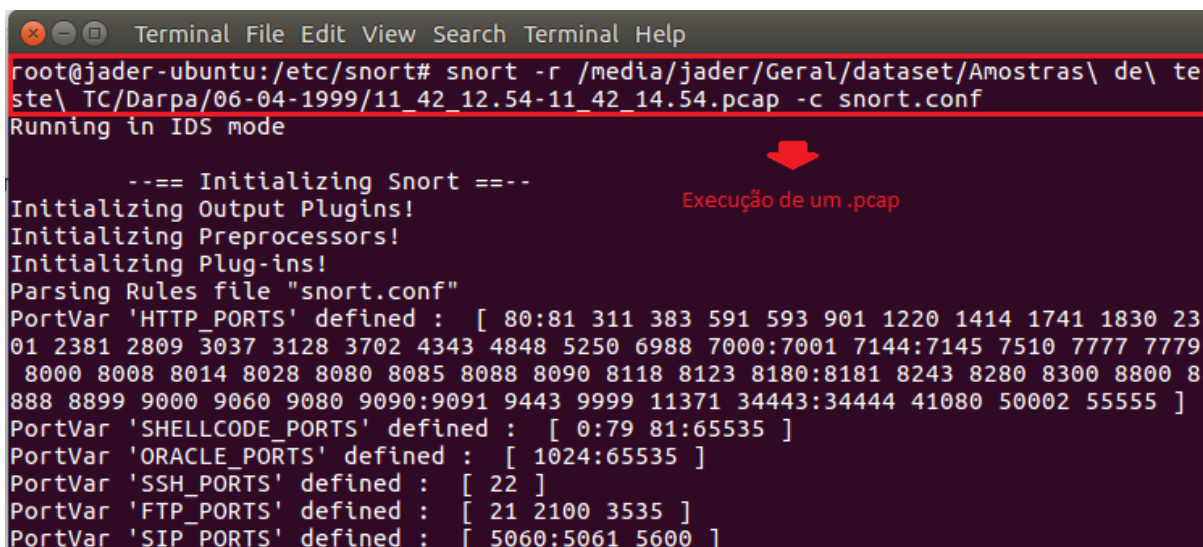
Os dados do *dataset* UNB ISCX 2012 foram adquiridos, após a solicitação por e-mail, instruída dentro do site do *dataset*. Da mesma forma que no Darpa, foi filtrado os intervalos de 2 segundos dentro do arquivo, e geradas 20 amostras de tráfego normal.

Após a geração das amostras na extensão .pcap, foi conferido o *IP* de destino dos pacotes dentro de cada arquivo. Esse procedimento foi necessário para configurar o *IDS* desenvolvido, e o Snort para simular o monitoramento do *host*. Então os arquivos foram executados com o Snort, e o *IDS* trabalhando em paralelo.

Foi verificada então a existência de algum ataque *SYN Flood* no arquivo de log gerado pela aplicação. Deve ser lembrando que a verificação de ocorrência de ataque, acontece em um intervalo de 2 segundos, ou seja, cada arquivo executado é um ataque ou não.

As amostras foram testadas separadamente com todas as regras criadas, para o algoritmo J48 e *Random Tree*. A execução de um arquivo .pcap dentro do Snort, pode ser vista na Figura 23.

Figura 23 - Execução de um arquivo .pcap no SNORT.



```

root@jader-ubuntu:/etc/snort# snort -r /media/jader/Geral/dataset/Amostras\ de\ te
ste\ TC/Darpa/06-04-1999/11_42_12.54-11_42_14.54.pcap -c snort.conf
Running in IDS mode

--== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file "snort.conf"
PortVar 'HTTP_PORTS' defined : [ 80:81 311 383 591 593 901 1220 1414 1741 1830 23
01 2381 2809 3037 3128 3702 4343 4848 5250 6988 7000:7001 7144:7145 7510 7777 7779
8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180:8181 8243 8280 8300 8800 8
888 8899 9000 9060 9080 9090:9091 9443 9999 11371 34443:34444 41080 50002 55555 ]
PortVar 'SHELLCODE_PORTS' defined : [ 0:79 81:65535 ]
PortVar 'ORACLE_PORTS' defined : [ 1024:65535 ]
PortVar 'SSH_PORTS' defined : [ 22 ]
PortVar 'FTP_PORTS' defined : [ 21 2100 3535 ]
PortVar 'SIP_PORTS' defined : [ 5060:5061 5600 ]

```

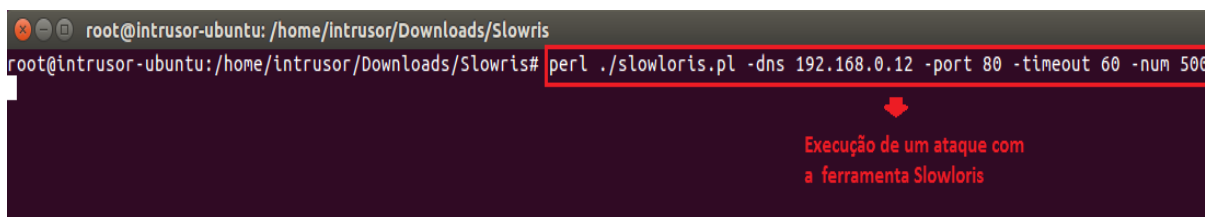
Fonte: Autor.

6.2.2. Testes em tempo real

Para os testes em tempo real, executou-se o *IDS* desenvolvido e o Snort, iniciando o monitoramento no servidor. Com a ferramenta *Slowloris* foi efetuado ataques, em intervalos de 1 minuto, com destino a porta 80. Foi enviado 500 pacotes em cada ataque, durante 30 minutos, para cada regra implementada.

Foi considerado um monitoramento de 30 minutos neste cenário, acreditando ser um tempo adequado para o teste de falsos positivos. O intervalo de 1 minuto para efetuar cada ataque, foi julgado adequado para obtenção de resultados de detecções e possíveis falsos negativos. Conhecendo as regras implementadas, percebeu-se que 500 pacotes eram suficientes para testar a detecção do ataque SYN Flood e obtenção dos resultados. A execução de um ataque na ferramenta *Slowloris* pode ser vista na Figura 24.

Figura 24 - Execução de um ataque com a ferramenta Slowloris.



```

root@intrusor-ubuntu:/home/intrusor/Downloads/Slowloris# perl ./slowloris.pl -dns 192.168.0.12 -port 80 -timeout 60 -num 500

```

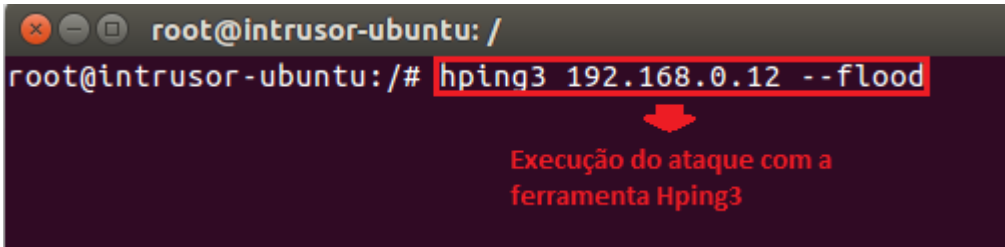
Fonte: Autor.

Com a ferramenta Hping3, foi efetuado ataques em intervalos de 30 segundos, enviando a quantidade máxima de pacotes possíveis, durante 4 segundos. O monitoramento durou 4 minutos, para cada regra implementada.

Na obtenção de resultados com essa ferramenta, por vezes foi iniciado o ataque, e nenhum pacote foi transmitido, visto com a ferramenta *Wireshark*. Dessa forma, os intervalos de ataques ficavam prejudicados, necessitando reiniciar o processo.

Optou-se então por um tempo de monitoramento menor, para que fosse possível reiniciar o processo quando necessário. Foi determinado 4 segundos de duração para cada ataque, pois o número de pacotes enviados não foi definido, podendo existir um atraso no envio pela ferramenta, e não expressar a realidade nos testes. A execução de um ataque com a ferramenta Hping3 pode ser vista na Figura 25.

Figura 25 - Execução de um ataque com a ferramenta Hping3



```
root@intrusor-ubuntu: /
root@intrusor-ubuntu:/# hping3 192.168.0.12 --flood
```

Execução do ataque com a ferramenta Hping3

Fonte: autor.

Após efetuado os testes descritos, foi analisado o arquivo “logAtaque” para cada regra testada. Dessa forma foi possível obter os resultados deste trabalho, apresentados na subseção 6.3.

6.3. Resultados Obtidos

Nesta subsecção são apresentados os resultados obtidos com os testes realizados. Os dados estão separados de acordo com o método utilizado, dataset e tempo real.

a) Resultados obtidos com *Datasets*.

Na Tabela 9 e Tabela 10, são apresentados os resultados obtidos para o algoritmo J48 e *Random Tree*, a partir de testes com os *datasets* Darpa e UNB ISCX 2012. Utilizando para testes de tráfego malicioso o Darpa, e para tráfego normal o ISCX.

Para os testes utilizando *datasets*, a obtenção de resultados ficou separada da seguinte forma: i) para arquivos contendo somente ataques, foram extraídos os dados e contabilizadas, as detecções corretas e os falsos negativos, ii) para arquivos com tráfego normal, foram extraídos os dados e contabilizados, falsos positivos e verdadeiros negativos.

Tabela 9 – Resultados obtidos para o algoritmo J48, utilizando datasets.

Algoritmo J48							
Regras	Tipo de Tráfego	Quantidade de arquivos	Total de pacotes	FN	TN	FP	Detecção Correta
Regra 1	Ataque <i>SYN Flood</i>	4	331	4	-	-	0
	Normal	20	1.645	-	8	12	-
Regra 2	Ataque <i>SYN Flood</i>	4	331	4	-	-	0
	Normal	20	1.645	-	19	1	-
Regra 3	Ataque <i>SYN Flood</i>	4	331	1	-	-	3
	Normal	20	1.645	-	20	0	-

Fonte: autor.

Tabela 10 - Resultados obtidos para o algoritmo *Random Tree*, utilizando datasets.

Algoritmo Random Tree							
Regras	Tipo de Tráfego	Quantidade de arquivos	Total de pacotes	FN	TN	FP	Deteccção Correta
Regra 1	Ataque <i>SYN Flood</i>	4	331	4	-	-	0
	Normal	20	1.645	-	17	3	-
Regra 2	Ataque <i>SYN Flood</i>	4	331	4	-	-	0
	Normal	20	1.645	-	20	0	-
Regra 3	Ataque <i>SYN Flood</i>	4	331	1	-	-	3
	Normal	20	1.645	-	20	0	-

Fonte: autor.

Analisando os resultados obtidos com *datasets*, e comparando a regra 1 em ambos, é possível perceber o padrão fraco gerado para deteccção. Como mencionado anteriormente, essas regras possuíam uma classificação baixa de ataques, mas foram mantidas para a visualização de resultados.

Nos dois casos, não foi detectado nenhum ataque corretamente, isso se deve ao fato das regras restringirem a um valor muito baixo o número de pacotes enviados, para considerar um ataque. Dessa forma, um ataque *SYN Flood*, que é caracterizado por enviar uma grande quantidade de pacotes, em um intervalo curto de tempo, não é detectado. Essa restrição na quantidade, também traz outros problemas, como o aumento de falsos positivos.

Na regra gerada pelo algoritmo J48, houveram 12 falsos positivos em 20, e no *Random Tree* foram 3. Essa diferença se deve as características específicas na regra.

A regra 2 também possui um padrão fraco, para ambos os algoritmos. Analisando os resultados, é possível perceber que não houve nenhuma deteccção de ataque correta, ainda pelos mesmo motivos encontrados na regra 1, a restrição a quantidade de pacotes enviados.

No caso dos falsos positivos, foi detectado apenas 1 para o algoritmo J48 e nenhum para o *Random Tree*. Esses resultados, se devem ao fato de a regra não considerar como ataque, valores tão baixos na quantidade de pacotes enviados.

A regra 3, para o algoritmo J48, e também para o algoritmo *Random Tree*, possuem o maior número de classificações de ataques, ou seja, o padrão mais forte para deteccção. No algoritmo J48, a regra 3 detectou 3 ataques corretamente, em 4 efetuados e não gerou nenhum falso positivo. Para o algoritmo *Random Tree*, também foram detectados 3 ataques em 4 efetuados, e nenhum falso positivo. Analisando o motivo pelo qual não foi detectado um dos

arquivos de ataque, tanto para o algoritmo J48, como para o *Random Tree*, foi percebido que no tráfego do ataque utilizado, os pacotes continham tamanhos maiores aos gerados nas regras. Acredita-se ser algo incomum para o ataque SYN Flood, que geralmente utiliza apenas pacotes com tamanho 0, aumentando a velocidade e quantidade no envio dos pacotes.

b) Resultados obtidos com testes em tempo real.

Nas Tabelas a seguir, são apresentados os resultados obtidos com os testes do IDS em tempo real. Na Tabela 11 e Tabela 12, podem ser vistos os resultados dos testes com a ferramenta *Slowloris* e Hping, para o algoritmo J48.

Tabela 11 - Resultados obtidos para o algoritmo J48 utilizando a ferramenta Slowloris.

Algoritmo J48 – Ferramenta Slowloris					
Regras	Quantidade de Ataques	FN	TN	FP	Detecção Correta
Regra 1	29	29	-	906	0
Regra 2	29	29	-	19	0
Regra 3	29	0	-	2	29

Fonte: autor.

Tabela 12 - Resultados obtidos para o algoritmo J48 utilizando a ferramenta Hping3.

Algoritmo J48 – Ferramenta Hping3					
Regras	Quantidade de Ataques	FN	TN	FP	Detecção Correta
Regra 1	7	7	-	33	0
Regra 2	7	7	-	3	0
Regra 3	7	0	-	0	7

Fonte: autor.

Os resultados obtidos com os testes em tempo real, foram analisados separadamente para cada algoritmo. Analisando a regra 1 para o algoritmo J48, houveram 36 ataques e nenhuma detecção correta, e 939 falsos positivos para um intervalo de 34 minutos de monitoramento.

Da mesma forma que os testes utilizando datasets, a restrição na quantidade de pacotes enviados causou os falsos negativos e falsos positivos. Reforçando o quão fraco é o padrão gerado.

Para a regra 2, utilizando o algoritmo J48, foram 36 ataques e nenhuma detecção correta. Ocorreram 22 falsos positivos no intervalo de 34 minutos, reduzindo consideravelmente o número se comparado com a regra 1.

Na regra 3, utilizando testes em tempo real para o J48, foram obtidas 36 detecções corretas em 36 ataques efetuados. Foram detectados 2 falsos positivos para a regra, durante o intervalo de 30 minutos com ataques da ferramenta *Slowloris*. Nos 4 minutos de monitoramento com ataques da ferramenta *Hping3*, não houveram falsos positivos. Analisando esses resultados, percebeu-se que em intervalos curtos, os falsos positivos podem não acontecer, mas em um tempo de monitoramento maior, eles ocorrem.

Os testes em tempo real, foram efetuados da mesma forma para o algoritmo *Random Tree*. Os resultados obtidos podem ser vistos na Tabela 13 e Tabela 14.

Tabela 13 - Resultados obtidos para o algoritmo *Random Tree* utilizando a ferramenta *Slowloris*.

Algoritmo Random Tree – Ferramenta Slowloris					
Regras	Quantidade de Ataques	FN	TN	FP	Detecção Correta
Regra 1	29	29	-	130	0
Regra 2	29	29	-	75	0
Regra 3	29	0	-	1	29

Fonte: autor.

Tabela 14 - Resultados obtidos para o algoritmo *Random Tree* utilizando a ferramenta *Hping3*.

Algoritmo Random Tree – Ferramenta Hping3					
Regras	Quantidade de Ataques	FN	TN	FP	Detecção Correta
Regra 1	7	7	-	4	0
Regra 2	7	7	-	4	0
Regra 3	7	0	-	0	7

Fonte: autor.

Analisando a regra 1 para o algoritmo *Random Tree*, é possível perceber que ocorrem os mesmos problemas do algoritmo J48. A regra não obteve detecções corretas em 36 ataques efetuados, e gerou 134 falsos positivos no intervalo de 34 minutos, devido à restrição na regra.

A regra 2 do *Random Tree*, não obteve nenhuma detecção correta, e gerou 79 falsos positivos. Esse valor de falsos positivos é bem superior para a regra, comparado aos testes com *dataset*. Essas diferenças ocorrem pelo fato do monitoramento ocorrer em tempos diferentes, podendo existir um tráfego maior no *host* em um momento, gerando mais falsos positivos.

Os resultados obtidos pela regra 3, para o algoritmo *Random Tree*, nos mostra a relevância da regra. Foram obtidas 36 detecções corretas, em 36 ataques efetuados, e 1 falso positivo, no intervalo de 34 minutos.

6.4. Análise de desempenho

Nesta subseção é apresentada a análise de desempenho do IDS, para os testes com *datasets* e tempo real. Os cálculos de desempenho foram efetuados da seguinte forma:

- **Cálculo de Falso Negativo:** o percentual de falsos negativos foi calculado, a partir dos dados caracterizados como falsos negativos, em relação aos ataques efetuados, para ambas as ferramentas.
- **Cálculo da Taxa de Detecção:** o percentual de detecção foi calculado, a partir da soma das detecções corretas, em relação a quantidade de ataques efetuados, para ambas as ferramentas.
- **Cálculo de Falso Positivo:** foi utilizado dois cálculos para gerar o percentual de falsos positivos. Quando utilizado *dataset*, é calculado o total de falsos positivos em relação ao total de pacotes normais. O cálculo para os resultados em tempo real de falsos positivos, foi obtido calculando os falsos positivos, em relação ao total de detecções, que são falsos positivos e detecções corretas.
- **Cálculo de Verdadeiro Negativo:** o percentual de verdadeiros negativos foi estimado, calculando a quantidade de verdadeiros positivos, em relação ao total de pacotes normais. Para os testes em tempo real, não foi estimado o percentual, pois não temos a

quantidade de pacotes que trafegaram na rede, durante o intervalo de 34 minutos de monitoramento.

Na Tabela 15 e Tabela 16, podem ser vistos os desempenhos do IDS para cada algoritmo, utilizando *datasets*. Após, é feita uma análise dos percentuais obtidos.

Tabela 15 - Desempenho do IDS com o Datasets para o algoritmo J48.

Desempenho do IDS com <i>Datasets</i> - Algoritmo J48				
Regras	FN - Falso Negativo (%)	TN - Verdadeiro Negativo (%)	FP - Falso Positivo (%)	DR - Taxa de Detecção (%)
Regra 1	100 %	40%	60 %	0 %
Regra 2	100 %	95%	5 %	0 %
Regra 3	25 %	100%	0 %	75 %

Fonte: do autor.

Tabela 16 - Desempenho do IDS com o Datasets para o algoritmo *Random Tree*.

Desempenho do IDS com <i>Datasets</i> - Algoritmo Random Tree				
Regras	FN - Falso Negativo (%)	TN - Verdadeiro Negativo (%)	FP - Falso Positivo (%)	DR - Taxa de Detecção (%)
Regra 1	100 %	85 %	15 %	0 %
Regra 2	100 %	100 %	0 %	0 %
Regra 3	25 %	100 %	0 %	75 %

Fonte: autor.

Analisando o desempenho do IDS, é possível perceber que para ambos os algoritmos, as regras geradas com os padrões fracos, são totalmente inviáveis de utilização. A regra 1 e regra 2, possuem uma taxa de detecção de 0% para o Algoritmo J48.

A regra 1 e 2 do algoritmo *Random Tree*, possui o mesmo percentual para detecções, que é de 0 %. Independente do percentual de falsos positivos, um *IDS* que não detecta ataques, não tem propósito.

A regra 3 para o algoritmo J48, obteve 75% de detecção, e não foram gerados falsos positivos. Para a regra 3 do algoritmo *Random Tree*, foi obtido o mesmo resultado. Esse percentual de detecção torna a regra totalmente viável de utilização, assim como o *IDS*.

Na Tabela 17 e na Tabela 18, podem ser vistos os percentuais de desempenho do *IDS*, em tempo real. São apresentados os desempenhos separadamente para o algoritmo J48 e *Random Tree*.

Tabela 17 - Desempenho do IDS em tempo real para o algoritmo J48.

Desempenho do IDS com testes em tempo real - Algoritmo J48				
Regras	FN - Falso Negativo (%)	TN - Verdadeiro Negativo (%)	FP - Falso Positivo (%)	DR - Taxa de Detecção (%)
Regra 1	100 %	-	100 %	0 %
Regra 2	100 %	-	100 %	0 %
Regra 3	0 %	-	5,2%	100 %

Fonte: autor.

Tabela 18 - Desempenho do IDS em tempo real para o algoritmo *Random Tree*.

Desempenho do IDS com testes em tempo real - Algoritmo <i>Random Tree</i>				
Regras	FN - Falso Negativo (%)	TN - Verdadeiro Negativo (%)	FP - Falso Positivo (%)	DR - Taxa de Detecção (%)
Regra 1	100 %	-	100 %	0 %
Regra 2	100 %	-	100 %	0 %
Regra 3	0 %	-	2,7 %	100 %

Fonte: autor.

Assim como nos testes com *datasets*, a regra 1 e a regra 2 para o algoritmo J48 são inviáveis de serem utilizadas, o percentual de detecção para ambas é de 0%. A regra 3, possui um percentual de 100% de acertos para os ataques, e 5,2% de falsos positivos. Dessa forma, a regra 3 para o algoritmo J48, se mostra muito eficaz para seu propósito, que é detecção de ataques.

Para o algoritmo *Random Tree*, a regra 1 e a regra 2 também possuem percentual de detecção de 0%. A regra 3 para esse algoritmo, detectou 100% dos ataques efetuados, e possui um percentual de 2,7 % de falsos positivos. Essa regra também se mostrou muito eficaz, com percentuais ainda melhores que a melhor regra para o algoritmo J48.

Ao final da análise dos resultados obtidos, é possível perceber claramente o motivo de se eliminar padrões fracos, com poucas classificações. Mostra que esses padrões são casos excepcionais.

7. CONCLUSÃO E CONSIDERAÇÕES FINAIS

Os resultados obtidos com este trabalho mostram que foi possível utilizar técnicas de Data Mining, e encontrar padrões para ataques *SYN Flood*. Foi possível concluir que o desenvolvimento de um *IDS* integrado ao Snort, utilizando os padrões gerados, foi eficaz para a detecção desses ataques. Tais resultados permitem afirmar, que o objetivo deste trabalho foi alcançado, e acredita-se que o trabalho criado pode tornar as redes mais seguras para esse tipo de ataque.

É importante salientar que os resultados foram obtidos em um ambiente controlado, mas as ferramentas utilizadas nos testes para os ataques, são bastante difundidas na Internet e utilizadas pelos atacantes. O *IDS* desenvolvido apresentou um percentual baixo de falsos positivos, em um intervalo de 34 minutos. Não foi verificado, se em um intervalo maior ou em outro ambiente de testes, o resultado seria diferente.

O *IDS* desenvolvido é integrado ao Snort, mas não automatiza sua execução. Como melhorias possíveis ao *IDS* desenvolvido, é sugerida a automatização de execução do Snort pelo programa, não somente as alterações dos arquivos de configuração da ferramenta, como faz este trabalho.

Uma outra sugestão de implementação, é a integração da API do *Weka* com o *IDS* desenvolvido, para geração dos padrões. Dessa forma, o *IDS* utilizaria a API e instanciaría seus métodos para a criação das regras, sem a necessidade da utilização da ferramenta *Weka* instalada.

O processo de descoberta de conhecimento se mostrou complexo, mas com o estudo desenvolvido, foi visto que os padrões podem ser gerados para qualquer outro tipo de ataque. Uma otimização para o *IDS* desenvolvido, agregando a detecção de novos ataques, indica um caminho para trabalhos futuros. Dessa forma, ampliando a capacidade do programa e tornando as redes mais seguras.

REFERÊNCIAS

AL-MUSAWI, Bahaa Qasim M. Mitigating DoS/DDoS attacks using iptables. *International Journal of Engineering & Technology*, v. 12, n. 3, 2012.

ALBIN, Eugene. A comparative analysis of the Snort and suricata intrusion-detection systems. Tese de Doutorado. Monterey, California. Naval Postgraduate School, 2011.

ALI, J.; KHAN, R.; AHMAD, N.; MAQSOOD, I. Random forests and decision trees. *International Journal of Computer Science Issues (IJCSI)*, v. 9, n. 5, 2012.

ARORA, Pooja. A Comparative Study of Instance Reduction Techniques. *International Journal of Advances in Engineering Sciences*, v. 3, n. 3, p. 7-13, 2013.

BAKER, Andrew. Snort IDS e IPS Toolkit. Jay Beal's Open Source Security. Rockland: Synpress, 2007.

BEALE, J.; BAKER, A. R.; CASWELL, B.; POOR, M. Snort 2.1 Intrusion Detection. 2. Ed. 2004.

BREIMAN, Leo. Random forests. *Machine learning*, v. 45, n. 1, p. 5-32, 2001.

CAMPOS, L. M. L. D.; LIMA, A. S. Sistema para Detecção de Intrusão em Redes de Computadores com Uso de Técnica de Mineração de Dados. In: V Congresso Tecnológico, Fortaleza, 2012.

CAMPOS, L. M. L. D.; OLIVEIRA, R. C. L. D.; ROISENBERG, M. Network intrusion detection system using *Data Mining*. In: *Engineering Applications of Neural Networks*. Springer Berlin Heidelberg, 2012. p. 104-113.

CHANDRASEKHAR, A. M.; RAGHUVeer, K. Confederation of FCM clustering, ANN and SVM techniques to implement hybrid NIDS using corrected KDD cup 99 dataset, *Communications and Signal Processing (ICCSP)*, 2014 International Conference on, Melmaruvathur. 2014. p. 672-676.

DARPA, Defense Advanced Research Projects Agency "Intrusion Detection Evaluation", MIT - Massachusetts Institute of Technology – Lincoln Laboratory, 1998, 1999, 2000. [Online]. Available <https://www.ll.mit.edu/ideval/docs/attackDB.html>. Acesso em: maio de 2016.

DENNING, Dorothy. An Intrusion-Detection Model. *Software Engineering, IEEE Transactions em*, Fevereiro de 1987. v.SE-13, n.2, p.222–232.

ENACHE, A. C.; SGÂRCIU, V. Anomaly intrusions detection based on support vector machines with bat algorithm. System Theory, Control and Computing (ICSTCC), 2014 18th International Conference, Sinaia. 2014, p. 856-861.

FAGUNDES, B.; NEU C. V.; OROZCO, A. M. S.; MICHELIN, R. A.; ZORZO, A. F.. SNORTIK: uma integração do IDS SNORT e o sistema de firewall do MIKROTIK ROUTEROS. In: Anais da 14ª ESCOLA REGIONAL DE REDES DE COMPUTADORES, em Setembro, 2016, Porto Alegre - RS, editora: Sociedade Brasileira de Computação, p. 27–30.

FARID, D. M.; HARBI, N.; RAHMAN, M. Z. Combining naive bayes and decision tree for adaptive intrusion detection. In: International Journal of Network Security & Its Applications (IJNSA), Vol. 2, N.2, 2010.

FAYYAD, U.; PIATETSKI-SHAPIRO, G.; SMYTH, P. The KDD Process for Extracting Useful Knowledge from Volumes of Data. In: Communications of the ACM, pp.27-34, Nov.1996.

FAYYAD, U.; PIATETSKI-SHAPIRO, G.; SMYTH, P. From *Data Mining* to knowledge discovery in databases. AI magazine, v. 17, n. 3, p. 37, 1996.

GUMUS, F.; SAKAR, C. O.; ERDEM, Z. O. Kursun. Online Naive Bayes classification for network intrusion detection. Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on, Beijing, 2014, p. 670-674.

HEBERLEIN, L.; DIAS, G.; LEVITT, K.; MUKHERJEE, B.; WOOD, J.; WOLBER, D. A network security monitor. In: RESEARCH IN SECURITY AND PRIVACY, 1990. PROCEEDINGS., 1990 IEEE COMPUTER SOCIETY SYMPOSIUM ON, 1990. p.296–304.

JUNQI, W.; ZHENG BING, H. Study of Intrusion Detection Systems (IDSs) in Network Security. In: WIRELESS COMMUNICATIONS, NETWORKING AND MOBILE COMPUTING, 2008. WICOM '08. 4TH INTERNATIONAL CONFERENCE ON, 2008. p.1–4.

KDD Cup 1999 Dataset, Information and Computer Science University of California, Irvine, repository, 1999. [Online]. Available <http://kdd.ics.uci.edu/databases/KDDCup'99/KDDCup'99>.

KUROSE, James F. Redes de computadores e a Internet: uma abordagem top-down. 5. Ed. São Paulo: Addison Wesley, 2010.

LAHOTI, L.; CHANDANKHEDE, C.; MUKHOPADHYAY, D. Modified Apriori Approach for Evade Network Intrusion Detection System, International Conference on Information Technology(ICIT), 2014.

MOUSTIS, D.; KOTZANIKOLAOU, P.. Evaluating security controls against HTTP-based DDoS attacks. In: Information, Intelligence, Systems and Applications (IISA), 2013 Fourth International Conference on. IEEE, 2013. p. 1-6.

NAFIR, A.; MAZOUZI, S.; CHIKHI, S. Collective intrusion detection in wide area networks. Innovations in Intelligent Systems and Applications (INISTA) Proceedings, 2014 IEEE International Symposium on, Alberobello, 2014, p. 46-51.

NAKAMURA, E. T.; GEUS, P. L. Segurança de Redes em Ambientes Cooperativos. Editora: Novatec, 2007.

NJOGU, H. W.; JIAWEI, L.; KIERE, J. N.; HANYURWIMFURA, D. A Comprehensive Vulnerability Based Alert Management Approach for Large Networks. Future Gener. Comput. Syst., Amsterdam, The Netherlands, The Netherlands, Janeiro de 2013. v.29, n.1, p.27-45.

PANDA, M.; PATRA, M. R. A Comparative Study of *Data Mining* Algorithms for Network Intrusion Detection, 2008 First International Conference on Emerging Trends in Engineering and Technology, Nagpur, Maharashtra, 2008, pp. 504-507.

PAXSON, Vern. Bro: a system for detecting network intruders in real-time. In: Conference On Usenix Security Symposium – Vol. 7, 7, 1998, Berkeley, CA, USA. USENIX Association, 1998. p.3-3.

REHMAN, Rafeeq. Intrusion Detection with Snort: Advanced IDS Technique Using SNORT, Apache, MySQL, PHP and ACID. 1. Ed. New Jersey: Prentice Hall, 2003.

RICHWINE, Lisa. Cyber attack could cost Sony studio as much as \$100 million. Local: Los Angeles, dezembro de 2014 Disponível em: <<http://www.reuters.com>> Acesso em: fevereiro de 2016.

ROESH, Martin. Proceedings of LISA '99: 13 Systems Administration Conference Seattle, Washington, EUA, 7-12 novembro de 1999.

SCARFONE, Karen; MELL, Peter. Guide to intrusion detection and prevention systems (IDS), Relatório técnico, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2007.

SENTHILNAYAKI, B.; VENKATALAKSHMI, K.; KANNAN, A. An intelligent intrusion detection system using genetic based feature selection and Modified J48 decision tree classifier, 2013 Fifth International Conference on Advanced Computing (ICoAC), Chennai, 2013, pp. 1-7.

SHIRAVI, Ali et al. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, v. 31, n. 3, p. 357-374, 2012.

SILVA, Renato. Redes Neurais Artificiais Aplicadas a Detecção de Intrusos em Redes TCP/IP. In *Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia Elétrica do Departamento de Engenharia Elétrica, PUC-Rio, Rio de Janeiro, Rio de Janeiro, Brasil, 2005.*

SNORT. The Snort Project. Disponível em: <<http://www.Snort.org/>>. Acesso em: fevereiro de 2016.

SOUZA, E. P.; MONTEIRO, J. S. Estudo sobre sistemas de detecção de intrusão por anomalias: uma abordagem utilizando redes neurais, 14º Workshop de Gerência e Operação de Redes e Serviços, Universidade Salvador/Salvador, 2008.

SYMANTEC. Welcome to the Security 1:1 - Part 3. Disponível em: <<http://www.symantec.com/connect/articles/security-11-part-3-various-types-network-attacks/>>. Acesso em: maio de 2016.

TANENBAUM, A. S; WETHERALL, D. J. *Computer Networks* 5 ed., 2012.

TAVALLAEE, M. *et al.* A detailed analysis of the KDD CUP 99 data set. In: *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009.* 2009.

THOMAS, C.; SHARMA, V.; BALAKRISHNAN, N. Usefulness of DARPA dataset for intrusion detection system evaluation. In: *SPIE Defense and Security Symposium. International Society for Optics and Photonics, 2008.* p. 69730G-69730G-8.

TRAN, T. P.; JAN, T. "Boosted Modified Probabilistic Neural Network (BMPNN) for Network Intrusion Detection," *The 2006 IEEE International Joint Conference on Neural Network Proceedings, Vancouver, BC, 2006,* pp. 2354-2361.

UNB ISCX. Intrusion Detection Evaluation Dataset. Disponível em: <<http://www.unb.ca/research/iscx/dataset/iscx-IDS-dataset.html/>>. Acesso em: novembro de 2016.

WITTEN, I. H.; WITTEN, E. *Data Mining: practical machine learning tools and techniques.* 2 ed. San Francisco: Morgan Kaufmann Publishers, 2005.

ZAMAN, S.; KARRAY, F. TCP/IP Model and Intrusion Detection Systems. In: Advanced Information Networking and Applications Workshops, 2009. Waina '09. International Conference On, 2009. p.90–96.

Santa Cruz do Sul, 2016.

Jáder Friderichs Vieira

Charles Varlei Neu