INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - IFSUL, *CAMPUS* PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

JOÃO RICARDO CLARO

SISTEMAS IDS E IPS – ESTUDO E APLICAÇÃO DE FERRAMENTA *OPEN SOURCE* EM AMBIENTE LINUX

Orientador: Prof. Me. Lisandro Lemos Machado

PASSO FUNDO, 2015

JOÃO RICARDO CLARO

SISTEMAS IDS E IPS – ESTUDO E APLICAÇÃO DE FERRAMENTA *OPEN SOURCE* EM AMBIENTE LINUX

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, *Campus* Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Prof. Me. Lisandro Lemos Machado

PASSO FUNDO, 2015

JOÃO RICARDO CLARO

SISTEMAS IDS E IPS – ESTUDO E APLICAÇÃO DE FERRAMENTA *OPEN SOURCE* EM AMBIENTE LINUX

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Prof. Me. Lisandro Lemos Machado

Orientador

_____

Prof. Me. Roberto Wiest

_____

Prof. Esp. José Antônio Oliveira de Figueiredo

_____

Prof. Me. Adilso Nunes de Souza

Coordenador do Curso

PASSO FUNDO, 2015

DEDICATÓRIA

A minha esposa Denise, pela compreensão, apoio e carinho.
E as minhas filhas Larissa e Vitória por entenderem
que o pai não podia brincar em vários momentos.
Amo vocês.

AGRADECIMENTOS

A este Instituto, direção e administração por terem proporcionado um ambiente sempre focado e direcionado, da melhor forma possível, ao aprimoramento dos estudos.

Aos professores com quem tive o privilégio de conviver e aprender, seus ensinamentos serão sempre lembrados. Obrigado!

Ao meu orientador Prof. Me. Lisandro Lemos Machado, pelo suporte, pelas correções e incentivos.

Aos funcionários da Coordenadoria de Tecnologia da Informação pelo convívio, pelo apoio, pela compreensão e pela amizade.

Aos colegas de curso, que fizeram parte da minha trajetória, dividindo momentos de descontração, estudos e conquistas.

Aos meus familiares, pelo amor, incentivo e apoio incondicional.

E a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“O preço da liberdade é a vigilância eterna.”

Thomas Jefferson

RESUMO

O presente trabalho apresenta um estudo sobre sistemas de detecção de intrusão e sistemas de prevenção de intrusão (IDS/IPS) em redes, baseadas em ambiente Linux, utilizando ferramenta *open source*. Pesquisou-se sobre os principais ataques a redes e sobre as principais ferramentas IDS existentes, sendo determinada a implementação, de forma atualizada, do IDS Snort, juntamente com o interpretador de *logs Basic Analysis and Security Engine* (BASE), no sistema operacional Debian 8 (Jessie), utilizando as últimas versões das ferramentas necessárias ao seu funcionamento. Como resultado deste procedimento, foi produzido um roteiro de instalação e configuração funcional. Para a realização de testes com a ferramenta implantada, foi criado um ambiente virtual, contendo um servidor com o Snort e uma máquina atacante, no qual foram realizadas três simulações de ataques. As tentativas de intrusão usadas nas simulações foram um teste de conectividade, um Scan de vulnerabilidades e um ataque de negação de serviço (DoS). Durante cada simulação, foi realizado o monitoramento do sistema, referente ao uso da unidade central de processamento (CPU), memória e fluxo de rede. O uso do Snort no ambiente virtual criado mostrou-se efetivo, pois diante das simulações a que ele foi submetido detectou as intrusões e gerou os alertas respectivos. Também foi verificado que a influência do Snort, na taxa de uso total da CPU, após o término de sua inicialização, e desta forma estando pronto para as detecções, pode ser considerada pequena, baseando-se nos resultados obtidos com as simulações aplicadas.

Palavras-chave: Segurança de redes. IDS/IPS. Snort. Simulações de ataques.

ABSTRACT

This paper presents a study on intrusion detection systems and intrusion prevention systems (IDS/IPS) in networks based on Linux environment, using open source tool. It was researched about the main network attacks and on the major existing IDS, being determined the implementation, in updated form, of the IDS Snort, along with the interpreter of logs Basic Analysis and Security Engine (BASE), in the operating system Debian 8 (Jessie), using the latest versions of the necessary tools for its operation. As a result of this procedure, a functional installation and configuration script was produced. In order to carry out tests with the implanted tool, a virtual environment was created, containing a server running Snort and an attacking machine, in which were accomplished three attack simulations. Intrusion attempts used in the simulations were a connectivity test, a vulnerability scan and a denial of service attack (DoS). During each simulation, system monitoring was performed, related to central processing unit (CPU) usage, memory and network flow. The use of Snort in the created virtual environment was effective, because before the simulations that it was subjected, it detected intrusions and generated their alerts. It was also verified that the influence of Snort, on the total CPU usage rate, after the end of its boot, and thus being ready for the detections, can be considered small, based on the results obtained with the applied simulations.

Keywords: Network Security. IDS/IPS. Snort. Attack simulations.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface
BASE – Basic Analysis and Security Engine
CERT – Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança
CPU – Central Processing Unit
CUDA – Compute Unified Device Architecture
DDoS – Distributed Denial of Service
DoS – Denial of Service
FTP – File Transfer Protocol
GPL – General Public License
GPU – Graphics Processing Unit
HIDS – Host Intrusion Detection System
HIPS – Host Intrusion Prevention System
HLBR – Hogwash Light BR
HTTP – Hypertext Transfer Protocol
ICMP – Internet Control Message Protocol
IDES – Intrusion Detection Expert System
IDS – Intrusion Detection System
IIS – Internet Information Services
IP – Internet Protocol
IPS – Intrusion Prevention System
IRQ – Interrupt Request
LAN – Local Area Network
NADIR – Network Anomaly Detection and Intrusion Reporter
NIDS – Network Intrusion Detection System
NIPS – Network Intrusion Prevention System
NSM – Network System Monitor
OISF – Open Information Security Foundation
OSI – Open Systems Interconnection
OSSEC – Open Source Host Based Intrusion Detection System
PC – Personal Computer
PHP – Hypertext Preprocessor

PIN – Personal Identification Number

POSIX – Portable Operating System Interface for Unix

SGBD – Sistema de Gerenciamento de Banco de Dados

SIEM – Security Information and Event Management

SIM – Security Information Management

SMB – Server Message Block

SNMP – Simple Network Management Protocol

SUID – Set User ID

TCP – Transmission Control Protocol

TLS – Transport Layer Security

UDP – User Datagram Protocol

UNIX – Uniplexed Information and Computing System

SUMÁRIO

1 INTRODUÇÃO

No cenário atual, com relação ao avanço na área da tecnologia da informação, a possibilidade de armazenar e também compartilhar informações facilmente tornou a segurança em redes de computadores imprescindível para a proteção dos dados que nelas trafegam. Para tanto, são utilizados inúmeros métodos para garantir a segurança das redes, por exemplo, uso de *firewalls*, controle de acesso, redes privadas virtuais, assinatura digital, criptografia e dentre eles, a detecção e prevenção de intrusão.

Este trabalho visa executar um estudo de sistemas de detecção de intrusão e sistemas de prevenção de intrusão (IDS e IPS), utilizando ferramenta *open source*, em ambiente Linux, a fim de analisar essa alternativa de segurança como forma de evitar problemas recorrentes em redes, tais como roubo de informações, paralização de serviços, perda de arquivos, entre outros. Serão abordadas neste trabalho noções acerca de estrutura e segurança de redes, tipos de ataques, descrição de IDS e IPS, bem como as principais ferramentas *open source* existentes, implementação de uma ferramenta IDS com configurações atualizadas no sistema operacional Debian 8 (Jessie), simulações de ataques e análise dos resultados.

1.1 MOTIVAÇÃO

Cada vez mais, arquivos físicos estão sendo substituídos por armazenamento digital, em ambientes profissionais ou domésticos. Com isso, as trocas de informações também estão sendo feitas digitalmente, ou seja, através de redes que se interconectam por meio da Internet. Esta evolução mundial na troca de informações provocou, na mesma proporção, o crescimento de crimes virtuais, dos mais variados tipos.

Segundo Tanenbaum (2003), grande parte dos problemas relacionados à segurança é provocada intencionalmente por pessoas que tentam obter algum benefício, chamar atenção ou prejudicar alguém. Isso prova que uma rede de computadores, seja ela doméstica ou corporativa, para estar segura, não basta que esteja livre de erros de programação.

O crescimento, juntamente com o aperfeiçoamento, dos mais diferentes tipos de ataques a redes de computadores, como, por exemplo, negação de serviço, ataques de força bruta, ataques para obtenção de permissões, etc. estão causando enormes prejuízos em nível mundial, tanto na área pública, como privada.

Conforme dados apresentados pelo Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.br), o número de incidentes relativos a intrusões a redes de computadores, reportados no ano de 2014, foi de 1.047.031 incidentes. Esta quantia é somente aquela referente aos ataques reportados. Estima-se que o número seja ainda maior.

Em virtude destas constatações, e procurando formas de proteção contra possíveis intrusões, optou-se por realizar o trabalho na área de segurança de redes, abordando ferramentas de IDS e IPS. Verificou-se, em um estudo preliminar, que grande parte dos materiais disponíveis acerca do tema estão desatualizados, o que gera grandes dificuldades para a instalação e configuração de um IDS atualizado, pois para que ele funcione corretamente, é necessário uma série de outros recursos a serem integrados, que também devem estar atualizados, o que provoca inevitavelmente problemas entre versões. Com isso, surge a necessidade da criação de um material atual e funcional.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Implementar e analisar o desempenho de um sistema de detecção e prevenção de intrusão em redes baseadas em ambiente Linux, utilizando ferramenta *open source*.

1.2.2 Objetivos específicos

a) Conceituar os sistemas de detecção e prevenção de intrusão IDS e IPS, expondo como funcionam, características e aplicações;

b) Pesquisar sobre os principais ataques a redes, como acontecem e os fins a que se destinam;

c) Estudar ferramentas *open source* que se destinam à detecção e prevenção de intrusões, demonstrando seus requisitos, suas características e funcionalidades;

d) Implementar, de forma atualizada, o sistema de detecção de intrusão Snort no sistema operacional Debian 8 (Jessie), utilizando as últimas versões das ferramentas necessárias ao seu funcionamento, gerando com isso um roteiro funcional de instalação e configuração;

e) Analisar a eficiência na detecção de intrusão do IDS Snort, bem como a verificação do uso da unidade central de processamento antes, durante e após sua inicialização, e durante as simulações de ataques.

2 SEGURANÇA EM REDES

A necessidade de comunicação de dados, em todos os níveis e em escala mundial está cada vez mais presente. Vive-se uma revolução na área de informação e comunicação sem precedentes em toda a história. Mas todo esse avanço trouxe também um grande problema, como proteger todas essas informações?

Em outra época, a segurança das informações se limitava a restringir o acesso físico a uma sala que continha os arquivos que guardavam as informações impressas em papéis. Já a segurança de informações em tempos atuais é bem mais complexa, pois depende de inúmeros fatores que às vezes passam despercebidos.

Uma rede para estar segura deve atender a alguns princípios básicos de segurança, que segundo a recomendação X.800 da *International Telecommunication Union* (1991) são:

- Autenticação - assegurar que as partes envolvidas na comunicação são de fato autênticas, ou seja, o emissor precisa provar para o receptor que ele é de fato quem diz ser, e vice-versa;
- Controle de acesso - o acesso ao sistema deve ser limitado e controlado para assegurar que somente mediante identificação e autorização, o serviço seja disponibilizado;
- Confidencialidade - refere-se à proteção dos dados transmitidos contra ataques passivos;
- Integridade - os dados recebidos devem ter a garantia de que não houve qualquer modificação, inserção, exclusão ou repetição;
- Não repudiação - deve ser oferecida a proteção contra a negação do dado transmitido, isto é, o receptor pode comprovar que o emissor é de fato quem enviou a mensagem, e o mesmo é válido para o emissor.

Com a observância desses princípios, foram desenvolvidos diferentes tipos de mecanismos de segurança que, quando bem empregados, elevam o grau de proteção contra os riscos de intrusão à rede.

2.1 PRINCIPAIS TIPOS DE AMEAÇAS A REDES

Com o objetivo de proteger a rede contra intrusões, é necessário se ter conhecimento sobre as principais ameaças existentes e sobre as pessoas que possuem a habilidade de concretizar tais ameaças.

Carvalho (2005) define o termo "atacante" como a pessoa que efetua um ataque a um sistema computacional, isto é, uma tentativa de comprometimento ou uma invasão, podendo obter êxito ou não. O termo mais conhecido e genérico para tais atacantes é *hacker*, que, segundo o autor, possui algumas ramificações:

- *Script Kiddies* – considerados o atacante mais comum, possuem baixo conhecimento técnico e utilizam ferramentas disponíveis na internet, não atacam um alvo em específico;
- *Crackers* – possuem alto conhecimento em informática, entre seus objetivos estão roubar informações importantes e destruir os sistemas invadidos. Praticam a maioria dos crimes virtuais envolvendo grandes perdas financeiras;
- *Carders* – responsáveis por efetuar compras com cartões de crédito roubados ou com seus números gerados através de software específico;
- *Cyberpunks* – possuem grande conhecimento na área e preocupam-se com a privacidade, comumente encontram e publicam vulnerabilidades encontradas em serviços, sistemas e protocolos;
- *Insiders* – utilizam engenharia social e suborno para a execução de espionagem industrial, geralmente são funcionários ou ex-funcionários da própria empresa atacada;
- *Coders* – são *hackers* que não atuam mais ilegalmente e compartilham seus conhecimentos sobre segurança;
- *White hats* – usam seus conhecimentos para encontrar vulnerabilidades em aplicações e sites e posteriormente divulgam e corrigem os erros encontrados para todos ou para quem os contrata;
- *Black hats* – tem por objetivo prejudicar ou causar grandes prejuízos a uma determinada vítima. Invadem sistemas de organizações com a finalidade de roubar informações para obter retorno financeiro. Utilizam a prática de chantagem denominada *black* mail;

- *Phreackers* – conhecidos por *hackers* da telefonia, alteram contas, atacam centrais telefônicas e realizam ligações gratuitas.

Nesse contexto, independente da ramificação, o atacante usa seus conhecimentos para encontrar vulnerabilidades ou erros que permitam atingir seus objetivos. Conforme Nakamura (2007, p. 80), "a proteção da informação depende da segurança em todos os níveis, que incluem: sistema operacional, serviços e protocolos, rede e telecomunicações, aplicação, usuários e organização, físico".

De acordo com a recomendação X.800 da *International Telecommunication Union* (1991), os ataques à segurança classificam-se em passivos e ativos. Os ataques passivos têm o objetivo de obter informações que estão sendo transmitidas na rede através do monitoramento das transmissões, porém sem afetar seus recursos. Já os ataques ativos modificam de alguma forma o fluxo de dados ou criam fluxos falsos, com intenção de alterar ou afetar sua operação.

Para a efetivação de um ataque, o primeiro passo que os *hackers* executam é a obtenção de informações a respeito do sistema que será invadido, através das diversas técnicas existentes. Segundo Nakamura (2007), o *hacker* após ter obtido informações acerca do alvo, age por meio de uma das seguintes maneiras:

- Monitoramento da rede;
- Invasão do sistema;
- Inserção de códigos danosos ou dados falsos no sistema;
- Sobrecarga do sistema com pacotes inúteis, comprometendo sua disponibilidade.

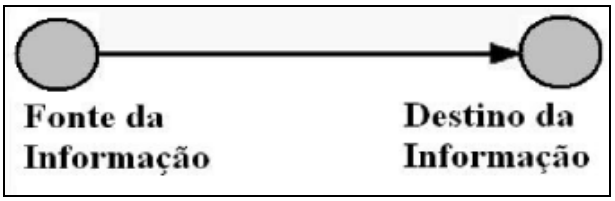
Ainda de acordo com Nakamura (2007), as consequências de um ataque são sempre negativas, podendo variar entre:

- Monitoramento sem autorização;
- Roubo de informações confidenciais;
- Modificações na base de dados e nos servidores da empresa;
- Lentidão ou indisponibilidade de serviço;
- Prejuízos financeiros;
- Danos à imagem da empresa;
- Retrabalho para recuperar o que foi danificado;
- Rompimento de contratos com clientes e perda de oportunidades.

Existem muitas formas de ataque e invasão a redes de computadores, que se proliferam intensamente e se renovam constantemente. Para Stallings (1999, apud Guimarães

et al., 2006), os ataques podem ser melhor visualizados através do fluxo das informações entre os integrantes da comunicação, dessa forma eles se dividem em quatro categorias, que são: por interrupção, por interceptação, por modificação ou por falsificação. A Figura 1 representa o fluxo normal da informação, onde ela percorre o caminho entre a origem e seu destino sem interrupção.

Figura 1 – Fluxo normal da informação.



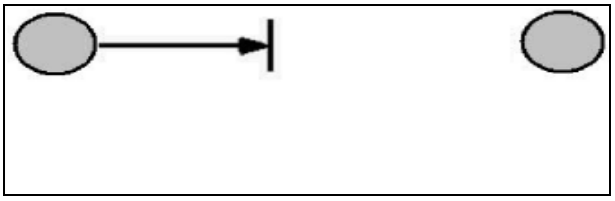
Fonte: Stallings (1999, apud Guimarães et al., 2006)

Quando ocorrem ataques, o fluxo normal da informação é alterado, os próximos subitens descrevem algumas destas alterações de fluxo.

2.1.1 Ataques de Interrupção

Este tipo de ataque tem como objetivo interromper ou destruir o serviço, desta forma afetando a disponibilidade da informação, como mostra a Figura 2.

Figura 2 – Ataque de interrupção.



Fonte: Stallings (1999, apud Guimarães et al., 2006)

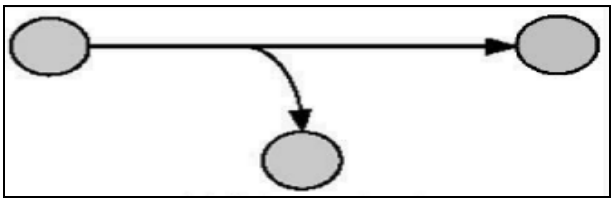
Como representante deste tipo de ataque tem-se o *Denial of Service* (DoS) e *Distributed Denial of Service* (DDoS), que tem como principal objetivo sobrecarregar o servidor com várias solicitações de serviço, a fim de torná-lo lento ou mesmo interrompê-lo. Conforme o CERT.br (2014), o ataque DoS é aquele no qual é utilizado um computador para efetuar a tentativa de bloqueio do serviço, já o DDoS ocorre quando são utilizados um conjunto de computadores de forma coordenada para tornar o ataque mais eficaz. Os ataques de negação de serviço possuem algumas subdivisões entre elas estão: SYN *Attack, Ping Attack, Flood Attack, Teardrop Attack, Smurf Attack*.

O ataque de força bruta (*Brute Force*) pode resultar em um ataque DoS devido ao excesso de tentativas de obter as informações em um período de tempo determinado. Este ataque consiste na técnica de tentar adivinhar nomes de usuários e senhas para utilizar os privilégios destes no acesso de sites e serviços diversos (CERT.br, 2014).

2.1.2 Ataques de Interceptação

A forma de ataque por interceptação, como mostra a Figura 3, visa capturar informações que estão sendo transmitidas sem a percepção do alvo, com isso a privacidade é comprometida. Seu objetivo principal é gerar cópias de informações, arquivos ou programas, de forma não autorizada. O agente para tal ataque pode ser um programa, uma pessoa ou um computador.

Figura 3 – Ataque de interceptação.



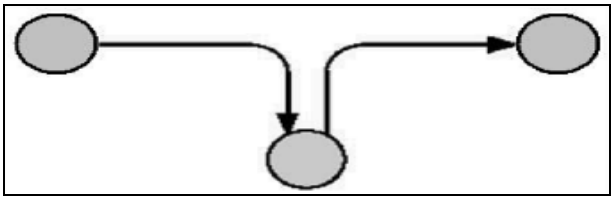
Fonte: Stallings (1999, apud Guimarães et al., 2006)

Um exemplo deste tipo de ataque é o *Man-in-the-Middle*, onde o intruso assume a identidade de um usuário válido. Outro exemplo é o uso de ferramentas denominadas *Sniffers*, esta técnica efetua buscas através do tráfego da rede com o objetivo de capturar os pacotes que estão nela circulando, seu maior objetivo é capturar informações que estejam trafegando de forma insegura (CERT.br, 2014).

2.1.3 Ataques de Modificação

Esta forma de ataque ocorre quando as informações transmitidas são alteradas, após terem sido capturadas, afetando sua integridade, de acordo com a Figura 4.

Figura 4 – Ataque de modificação.



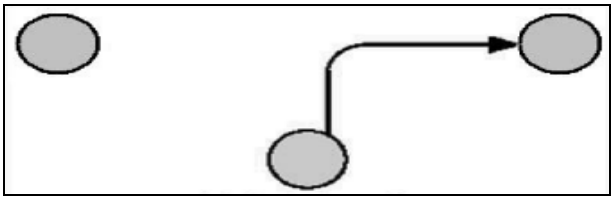
Fonte: Stallings (1999, apud Guimarães et al., 2006)

Um exemplo deste ataque é o *Replay Attack*, no qual um agente malicioso intercepta uma informação de um serviço, reproduzindo esta mesma mensagem algum tempo depois com alterações.

2.1.4 Ataques de Falsificação

No ataque de falsificação, a finalidade é se passar por um usuário do sistema para obter informações e transmiti-las na rede, comprometendo a autenticidade da informação, conforme a Figura 5.

Figura 5 – Ataque de falsificação.



Fonte: Stallings (1999, apud Guimarães et al., 2006)

Um ataque deste tipo é o IP *Spoofing*, que substitui o IP do computador invasor fazendo ele se passar por um computador confiável da rede, ganhando os privilégios na comunicação.

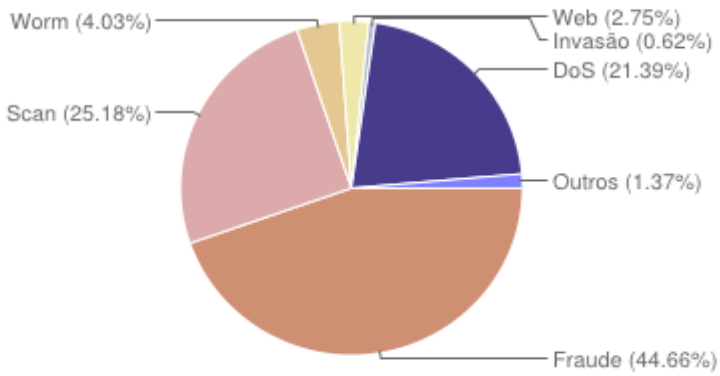
O *scanning* de vulnerabilidades é uma forma de ataque de falsificação, pois simula requisições verdadeiras para obter informações da máquina atacada. Este ataque geralmente é precedido por outro ataque denominado *port scanner*. Segundo Carvalho (2005), o *port scanner* identifica as portas e serviços disponíveis no sistema computacional, mas isto é insuficiente para colocar a segurança em risco. Para que haja um efetivo ataque, usa-se o *scanning* de vulnerabilidades nas portas e serviços anteriormente detectados.

2.1.5 Códigos Maliciosos (*Malwares*)

São programas desenvolvidos com o objetivo específico de causar danos aos computadores, prejudicando seu desempenho e a privacidade do usuário. Segundo a Cartilha de Segurança para Internet (CERT.br, 2014), os principais *malwares* existentes são: vírus, *worms*, *bots* e *botnets*, *spywares*, *backdoors*, *rootkits* e cavalos de tróia (*trojans*).

Dados estatísticos do CERT.br no ano de 2014 apontam que a forma de ataque mais comum são as chamadas Fraudes, que englobam os ataques de *phishing*, como por exemplo páginas falsas de bancos e lojas, que somaram 44,66% dos incidentes, a Figura 6 mostra qual a incidência de cada tipo de ataque reportado.

Figura 6 - Incidentes Reportados ao CERT.br – 2014 (Tipos).



Fonte: CERT.br

Diante dos inúmeros tipos de atacantes e das diferentes formas de ataque, torna-se muito difícil garantir a segurança da rede. É fundamental que o administrador esteja a par de todos os riscos envolvidos para que possa, diante de cada situação, escolher a melhor forma de proteção a ser utilizada.

2.2 MECANISMOS DE SEGURANÇA

Para fazer com que os princípios de segurança sejam alcançados a fim de evitar que os invasores consigam alcançar seus objetivos, existem uma série de práticas de segurança da informação, bem como mecanismos disponíveis.

Tão importante quanto implantar mecanismos de segurança, é adotar uma política de segurança para a rede. Entre os inúmeros cuidados que compõem uma política de segurança em uma organização, estão as regras para formação de acesso ao sistema.

Existe atualmente uma série de mecanismos de segurança à disposição, todos eles com o mesmo objetivo, que é garantir a segurança da rede. Podem-se destacar os seguintes mecanismos, dentre outros:

- Criptografia – é o processo no qual as informações são transformadas em códigos, de forma que somente pessoas autorizadas tenham acesso a elas. Segundo Carvalho (2005), "A criptografia é a ciência de transformar dados que aparentemente podem ser entendidos e interpretados pelas pessoas, em dados que não possuem significado algum, e que quando necessário podem ser recuperados à sua forma original".
- *Firewall* - conhecido como a primeira barreira contra ataques a uma rede. Segundo Cheswick et al. (2005, p. 177),

> ...um firewall é qualquer dispositivo, software, arranjo ou equipamento que limita o acesso à rede. Ele pode ser uma caixa que você compra ou constrói, ou uma camada de software em alguma outra coisa. Atualmente, os firewalls vem "gratuitamente" dentro de muitos dispositivos: roteadores, modens, estações de base sem fio e switches de IP, para citar alguns, Os firewalls de software estão disponíveis para (ou são incluídas com) todos os sistemas operacionais populares. Eles podem ser um calço de cliente (uma camada de software) dentro de um PC executando Windows, ou um conjunto de regras de filtragem implementado em um kernel Unix.

- Autenticação – o sistema de autenticação visa reconhecer a identificação de um determinado usuário a fim de lhe permitir ou não o acesso ao sistema. Conforme Carvalho (2005) existe maneiras de validar a identificação do usuário, essas maneiras são baseadas no que o usuário sabe, por exemplo: senha, chave ou *Personal Identification Number* (PIN); no que o usuário tem, por exemplo: *smart card*, *Token*; nas características do usuário, por exemplo, biometria; e através da combinação dos métodos.
- Sistemas de detecção de intrusão – este mecanismo detecta atividades suspeitas de serem invasivas à rede, gerando alertas, avisando ao administrador da rede sua ocorrência.
- Sistemas de prevenção de intrusão – esta forma de proteção, não só reconhece e alerta uma intrusão na rede, mas também bloqueia imediatamente a ameaça, impedindo um dano maior à rede.

Dentre os mecanismos apontados, serão destacados na presente pesquisa, o Sistema de Detecção de Intrusão (IDS) e o Sistema de Prevenção de Intrusão (IPS).

2.3 SISTEMA DE DETECÇÃO DE INTRUSÃO (IDS)

O Sistema de Detecção de Intrusão (*Intrusion Detection System* - IDS) é um monitorador de processos que acontece em uma rede ou em determinado host, com o objetivo de analisar o fluxo de dados a fim de detectar possíveis intrusões. Esse sistema tem a capacidade de prever e reconhecer potenciais ataques, identificá-los e emitir um alerta aos administradores da rede. Conforme Silva,

> Sistemas de detecção de intrusão (IDS – Intrusion Detection System) são ferramentas de software ou appli*an*ce utilizadas em conjunto com outros mecanismos de segurança tais como *firewalls*, antivírus e mecanismos de criptografia para reforçar a segurança de um ambiente de rede, relatando eventos suspeitos ou impedindo que ações maliciosas tenham êxito e se propaguem pela rede. (2008, p.57)

Nakamura (2007) ressalta a importância do IDS como mecanismo de defesa de uma rede, pois pode detectar os ataques que são efetuados através de portas legítimas, não protegidas pelo *firewall*.

Conforme Neto et al. (2011), nas décadas de 1950 e 1960 foi introduzida a prática de usar auditorias para inspecionar dados e procurar fraudes e/ou erros. Na década de 1970, criaram-se muitas iniciativas de segurança computacional continuando a ter a auditoria como importante ferramenta de inspeção.

Nos anos 80, os dados coletados eram revisados manualmente para detectar violações e segurança, mas com o aumento do volume destes dados, o procedimento tornou-se muito demorado. Para resolver isso, foram propostos métodos de redução da quantia de dados através da comparação de estatísticas de comportamento com observações resumidas. Nessa década, surgiu um importante projeto de pesquisa na área, denominado *Intrusion Detection Expert System* (IDES), que serviu como base para os sistemas de detecção de intrusão conhecidos. Ao final dos anos 80, muitos sistemas foram desenvolvidos, baseados em uma combinação de estatísticas e sistemas especializados, como por exemplo, *Haystack*, NADIR e *Wisdom and Sense*, incorporados a sistemas comercias de gerenciamento de banco de dados, tais como *Oracle* e *Sybase* (Neto et al., 2011).

O *Network System Monitor* (NSM), desenvolvido nos anos 90 pela Universidade da Califórnia, foi o primeiro sistema a monitorar diretamente o tráfego de rede e utilizava esse tráfego como principal fonte de dados para realizar suas análises, técnica utilizada nos sistemas IDS comerciais utilizados até a presente data.

2.3.1 Estrutura interna do IDS

Segundo Miguel (s.d.), o IDS é formado internamente por seis módulos, como detalhado na Figura 7. O gerenciador de análise é o cérebro do sistema, ele consulta através de regras preestabelecidas o gerenciador de pacotes para efetuar uma eventual análise. Se houver um pacote a ser analisado, o gerenciador de análise compara o pacote com as assinaturas requeridas ao gerenciador de assinaturas. Caso haja confirmação das características do pacote serem descritas por alguma assinatura, o gerenciador de análise aciona o gerenciador de medidas de defesa para que seja combatida a ameaça detectada.

Figura 7 - Estrutura interna do Sistema de Detecção de Intrusão.

Fonte: MIGUEL (s.d.)

2.3.2 Tipos de IDS

Nakamura (2007) classifica dois tipos primários de IDS, que são *Host-Based Intrusion Detection System* – HIDS e *Network-Based Intrusion Detection System* – NIDS. Além destes dois tipos, foi desenvolvido o *Hybrid* IDS, que abrange características dos dois sistemas listados anteriormente.

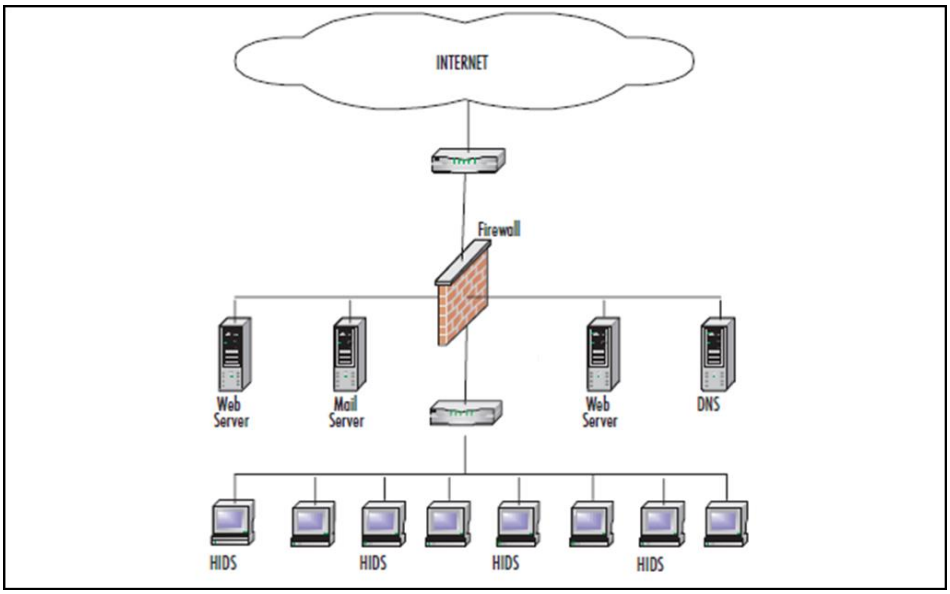
2.3.2.1 Sistemas de Detecção de Intrusão Baseado em *Host* (HIDS)

Neste caso, o IDS encontra-se instalado em cada máquina monitorada, como mostra a Figura 8, a fim de analisar os eventos gravados nos arquivos de *log* ou pelos agentes de

auditoria. Funciona como última linha de defesa, no caso do ataque ter sido bem sucedido e ter conseguido atravessar o *firewall* e o NIDS. Segundo Assunção (2009), o HIDS consegue detectar as seguintes situações:

- Uso não correto e exagerado da memória;
- Processos cujo comportamento é suspeito;
- Conexões suspeitas na rede;
- Utilização da CPU;
- Utilização de *System Calls*;
- Uso detalhado do disco.

Figura 8 - Sistemas de Detecção de Intrusão Baseado em *Host* (HIDS).



Fonte: Baker et al. (2007)

O HIDS é essencial na detecção de certos tipos de ataque como, por exemplo, cavalo de troia, cuja comunicação é criptografada, inviabilizando a detecção de assinaturas do NIDS (Assunção, 2009).

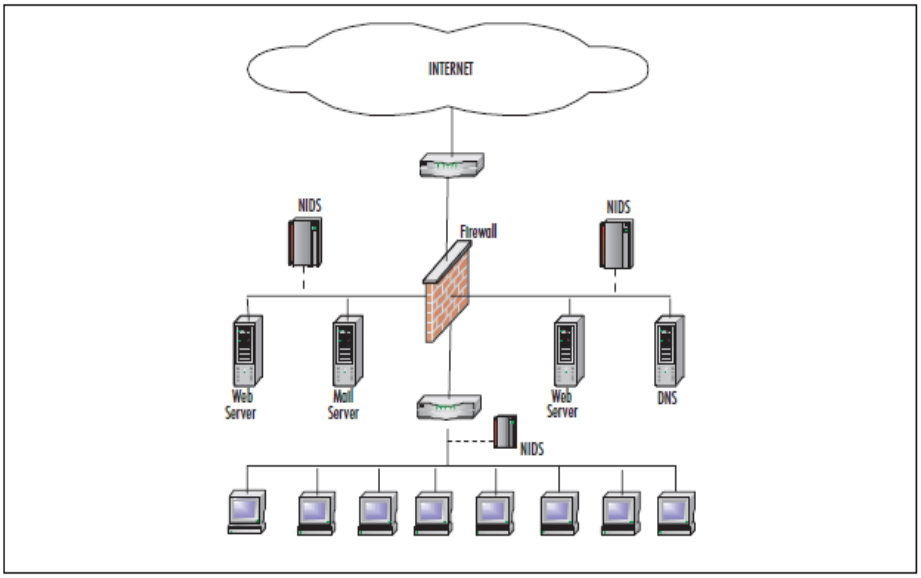
Conforme Carvalho (2005), o HIDS possui como vantagem não necessitar de hardware adicional, pois se encontra diretamente no *host* monitorado; não possuem dependência quanto à topologia da rede; geram poucos falsos positivos e ainda, os ataques físicos contra o sistema podem ser detectados. Já as desvantagens são a dependência do sistema operacional; não conseguir detectar ataques de rede; possível perda de desempenho do *host* monitorado e caso o HIDS seja invadido, informações podem ser perdidas.

2.3.2.2 Sistemas de Detecção de Intrusão Baseado em Rede (NIDS)

Ao invés de monitorar um único computador, o NIDS monitora a rede como um todo. Conforme Nakamura (2007), ele monitora o tráfego do segmento de rede no qual está inserido, com sua interface de rede atuando em modo promíscuo. A detecção é feita através da captura e análise dos cabeçalhos e conteúdos dos pacotes, os quais são comparados com padrões ou assinaturas estabelecidas, sendo um mecanismo eficaz contra ataques como *port scanning*, IP *spoofing*, SYN *flooding*.

Na Figura 9 está representada uma configuração de rede com três NIDS, que foram posicionados em segmentos estratégicos para monitorar o tráfego de toda a rede. Conforme Baker et al. (2007), o uso de múltiplos NIDS em uma rede garante uma forma de defesa abrangente.

Figura 9 - Sistemas de Detecção de Intrusão Baseado em Rede (NIDS).



Fonte: Baker et al. (2007)

O sistema de detecção baseado em rede é constituído por dois componentes principais, os sensores e a estação de gerenciamento. Os sensores são os dispositivos colocados em determinados pontos da rede, que realizam o monitoramento propriamente dito, já a estação de gerenciamento é responsável pelo gerenciamento remoto de todos os sensores (MIGUEL, s.d.).

O NIDS operando com essas particularidades consegue ter uma abrangência maior na detecção de intrusos, comparado ao HIDS. Um grande ponto positivo é que o NIDS apresenta a propriedade de não ser visível ao atacante, fazendo com que o ataque seja efetuado sem

cuidados. Outras vantagens do NIDS são, segundo Carvalho (2005), que o desempenho da rede não é afetado; a detecção e identificação dos ataques em tempo real, facilitando tomadas de decisões imediatas; eficácia na detecção de *port scanning*; não se restringe a somente detectar ataques, mas também às tentativas de ataque não concretizadas. As desvantagens deste sistema são a dificuldade em monitorar dados cifrados e em redes saturadas pode haver perdas de pacotes.

2.3.2.3 Sistemas de Detecção de Intrusão Híbrido (Hybrid IDS)

O uso conjunto dos dois tipos de IDS descritos anteriormente é denominado sistema híbrido. Este sistema de detecção de intrusão trabalha como se fosse um NIDS, porém esse procedimento é feito na forma de um HIDS, pois ele analisa somente pacotes endereçados ao próprio sistema. Dessa forma, a desvantagem com relação ao desempenho, que ocorre no NIDS, é descartada. No entanto, surge outro problema, a escalabilidade, o sistema híbrido é instalado em cada *host* (NAKAMURA, 2007).

2.3.3 Tipos de detecção

Os sistemas de detecção de intrusão utilizam duas metodologias para realizar a análise dos dados e detectar os ataques, que são: detecção baseada em assinatura e detecção baseada em anomalias. Além destes dois tipos de análises, existe uma técnica de análise híbrida, que combina as duas anteriores.

Conforme Carvalho (2005), após estas análises terem sido executadas, existe quatro tipos de atividades possíveis:

- Intrusiva e anômala (verdadeiro positivo) - a intrusão é detectada e esta é verdadeira;
- Intrusiva e não anômala (falso negativo) - a atividade é intrusiva, mas não é reconhecida como tal;
- Não intrusiva e anômala (falso positivo) - a atividade é acusada como sendo intrusiva, quando na verdade não é;
- Não intrusiva e não anômala (verdadeiro negativo) - não há intrusão, e esta não é detectada.

2.3.3.1 Sistema de detecção baseado em assinatura

Este mecanismo de detecção, baseado em assinatura, compara os pacotes recebidos com um conjunto de assinaturas previamente definidas. Neste caso, o IDS compara um padrão apresentado pelo possível ataque a uma base de dados de padrões de ataques (assinaturas). Se o padrão suspeito for confirmado na base de dados, o ataque é detectado.

Carvalho (2005) afirma que esta metodologia é a mais utilizada, pois é considerada mais rápida e gera menos falsos positivos que o outro sistema de detecção. Sua desvantagem é a incapacidade de detectar ataques novos ou desconhecidos, que ainda não foram atualizados no banco de dados.

Outro fator importante é possibilidade de o administrador criar sua própria assinatura contendo um conjunto de regras personalizadas, o que fará diminuir os falsos positivos e adequar melhor o IDS ao seu ambiente (NAKAMURA, 2007).

2.3.3.2 Sistema de detecção baseado em anomalia

O sistema de detecção baseado em anomalias analisa o comportamento do tráfego de rede, comparando-o a um modelo de comportamento considerado normal para o ambiente. Quando o tráfego de rede se desvia do comportamento considerado normal, ocorre uma anomalia, e o IDS o considera como um possível ataque, gerando o alerta (MIGUEL, s.d.).

Conforme Nakamura (2007), este sistema considera que tudo que não é conhecido pode ser perigoso e deve ser evitado, dessa forma este IDS tem a vantagem de poder detectar ataques que não tiveram assinaturas previamente definidas e até mesmo ataques novos. No entanto, apresenta desvantagens como a geração de falsos negativos, quando um ataque não altera significativamente o comportamento do tráfego, e um grande número de falsos positivos.

2.3.3.3 Sistema de detecção híbrido

A combinação dos métodos de detecção anteriores resulta no sistema híbrido que permite detectar ataques conhecidos assim como as ameaças desconhecidas que eventualmente possam causar algum tipo de anomalia no sistema. Primeiramente é feita a análise no conjunto de assinaturas e caso não encontre nenhuma correspondente, passa a analisar a possível ameaça pela técnica de anomalia.

2.4 SISTEMA DE PREVENÇÃO DE INTRUSÃO (IPS)

O Sistema de Prevenção de Intrusão (*Intrusion Prevention System* - IPS) é um complemento do IDS, ele acrescenta à detecção de ataques, a possibilidade de prevenção.

Ambos IDS e IPS necessitam de uma base de dados de assinaturas conhecidas para realizar a comparação com possíveis ataques. No entanto, o IDS se restringe a detectar tentativas de intrusão, registrá-las e enviá-las ao administrador da rede, o IPS opera "*inline*" na rede, adotando medidas adicionais para bloquear as intrusões em tempo real. Embora sejam conceitos similares, aparentemente tenham as mesmas funções e até possam substituir firewalls, cada uma delas na verdade oferece uma camada a mais de proteção à rede (DOHERTY et al., 2008).

2.4.1 Sistema de prevenção de intrusão baseado em *host* (HIPS)

Funciona de modo semelhante ao HIDS, as verificações são em cima da máquina na qual se encontra instalado, porém além de detectar o ataque, ele toma decisões a respeito das análises efetuadas. Tem acesso direto ao sistema operacional da máquina e ao próprio *kernel*, podendo dessa forma controlar os acessos ao sistema de arquivos, configuração e registros do sistema. Outro diferencial do HIPS é que ele identifica comportamentos suspeitos no sistema operacional, ao invés de comparar assinaturas (NAKAMURA, 2007).

Além disso, o HIPS traz a possibilidade de que o tráfego de rede criptografado possa ser identificado após o processo de descriptografia do pacote, possibilitando a detecção do ataque antes cifrado, fato que não ocorre no uso do NIPS e NIDS (VACCA, 2010).

2.4.2 Sistema de prevenção de intrusão baseado em rede (NIPS)

Esse sistema se baseia em um dispositivo *inline*, este dispositivo pode ser um roteador ou um *switch*, pois eles repassam os pacotes entre as redes. Sempre que um ataque é identificado são tomadas decisões baseadas nas regras pré-definidas e são essas regras que irão bloquear o ataque suspeito.

De acordo com Nakamura (2007), o NIPS apresenta a propriedade de efetuar *drop* na conexão, dessa forma fazendo com que os pacotes não cheguem a seu destino, tal como os *firewalls*.

2.5 FERRAMENTAS *OPEN SOURCE* DE IDS/IPS

Atualmente existem várias ferramentas *open source* disponíveis para a detecção de intrusão, cada uma com suas características próprias. A escolha da ferramenta adequada a uma determinada rede dependerá dos recursos que a ferramenta disponibiliza e da necessidade da rede. Algumas dessas ferramentas também possuem serviço de prevenção de intrusão. Na busca de ferramentas que atendam a proposta deste trabalho pode-se destacar: OSSEC, Suricata, Samhain, HLBR e Snort.

2.5.1 OSSEC

Desenvolvido por Daniel B. Cid em 2004, o OSSEC (*Open Source Host-based Intrusion Detection System*) foi adquirido pela empresa Trend Micro em 2009. Sua licença segue os termos GNU *General Public License* v2 e atualmente encontra-se na versão 2.8.1.

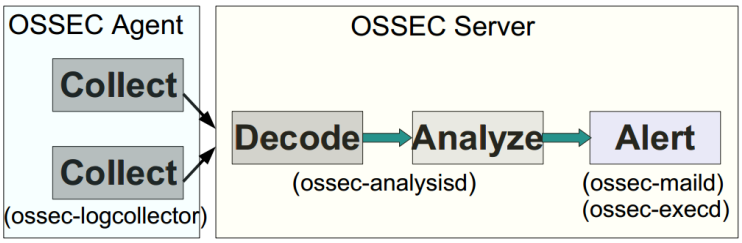
O OSSEC é uma ferramenta multiplataforma de detecção de intrusão baseada em *host* (HIDS), que utiliza *log* de monitoramento e SIM (*Security Information Management*) / SIEM (*Security Information and Event Management*), que juntos permitem uma rápida identificação e resposta a incidentes. Ele permite a verificação de integridade de arquivos em sistemas Unix e Windows, checagem da integridade de registro do Windows e detecção de *rootkits* em sistemas Unix. Possui um sistema de alerta configurável que possibilita o envio de e-mails e mensagens para dispositivos móveis. Através do *Active Response* é possível relacionar um evento a uma regra de bloqueio do processo suspeito. Seu gerenciamento é centralizado e seu monitoramento pode ser feito com ou sem um agente (OSSEC, s.d.).

A composição do OSSEC consiste em:

- Gerente – armazena todas as regras, analisa as informações recebidas e gerencia os agentes;
- Agente – é um software instalado no sistema a ser monitorado para coletar as informações, em tempo real, e enviar à central para análise. O agente pode também ser usado em um ambiente de virtualização para monitoramento de sistemas operacionais convidados e hospedeiros (VMware);
- *Agentless* – utilizado para verificação de integridade de arquivos onde não é permitido instalar um agente, como em firewalls e roteadores, possibilitando receber os eventos *syslog*.

Os principais processos internos do OSSEC, referentes ao fluxo da análise de *log*, estão demonstrados na Figura 10.

Figura 10 – Fluxo de *log* (*agent*/*server*).



Fonte: Cid, 2007

Os processos internos, segundo Cid (2007), envolvem as seguintes tarefas:

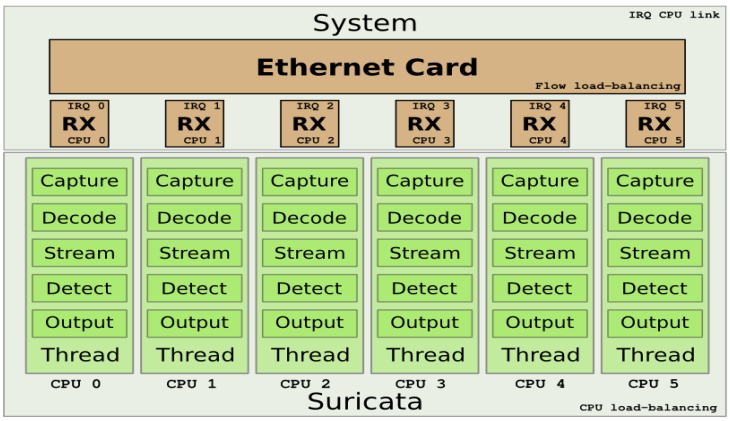
- *Logcollector* – lê os arquivos de *log* (*syslog*, *Flat files*, Windows *event log*, IIS, etc);
- *Remoted* – recebe os *logs* remotos dos agentes;
- *Agentd* – encaminha os *logs* para o gerente;
- *Analysisd* – é o processo principal, decodifica e executa todas as análises;
- *Maild* – envia os alertas de e-mail;
- *Execd* – executa o *Active Response*;
- *Monitord* – monitora o status do agente, comprime e identifica os arquivos de *log*, etc.

2.5.2 Suricata

Criado pela OISF (*Open Information Security Foundation*) em 2010, o Suricata é um sistema NIDS/NIPS, multiplataforma, com licença GPLv2 que hoje se encontra na versão 2.0.4, lançada em setembro de 2014.

O Suricata é altamente escalável, possui um mecanismo *multi thread* que permite inspecionar uma grande quantidade de tráfego na rede, com maior velocidade e eficiência na análise dos pacotes como demonstra a Figura 11. Somada a essa característica, o Suricata disponibiliza um sistema de aproveitamento da GPU (*Graphics Processing Unit*) através de um suporte a plataforma de computação paralela CUDA (*Compute Unified Device Architecture*), que consequentemente acelera o processamento.

Figura 11 – Modelo *multi thread*



Fonte: Leblond, 2012

Tem capacidade de detectar automaticamente os protocolos mais comuns, tais como IP, TCP, UDP e ICMP, e também HTTP, TLS, FTP e SMB. As regras de detecção, desta forma, são baseadas no protocolo, independentemente da porta em que ocorra. Inclui decodificação de IPv4, IPv6 e registro de certificado.

Possui completo suporte a API pcap (*P acket CAP ture*) permitindo análises fáceis. O Suricata pode identificar mais de 4000 arquivos ao analisar o tráfego da rede em tempo real. Apresenta análises avançadas e funcionalidades disponíveis para detectar ameaças que não estejam presentes no conjunto de regras, além disso, fornece uma saída de eventos e alertas de fácil integração com outras ferramentas (SURICATA, s.d.).

2.5.3 Samhain

Elaborado por Rainer Wichmann, o Samhain atualmente encontra-se na versão 3.1.2 e é distribuído sob a licença GNU GPL v2.

O Samhain é uma ferramenta de detecção de intrusão baseada em *host* (HIDS), para plataformas POSIX, podendo ser usado no sistema operacional Windows com emulação POSIX, somente no agente. Permite a verificação da integridade de arquivos, o monitoramento e análise de arquivos de *log*, detecção de *rootkits*, monitoramento de portas, detecção de executáveis SUID maliciosos, verificação de processos ocultos, monitoramento e relatório de eventos *login/logoff*, entre outros. Para garantir a integridade de um sistema Linux, o Samhain gera um *checksum* dos arquivos que monitora, conferindo-o em intervalos regulares e se detectar algum tipo de intrusão, alerta o administrador do sistema.

Ele pode ser usado em único host de modo autônomo ou no monitoramento de um conjunto de *hosts*, com diferentes sistemas operacionais, permitindo o registro e gerenciamento centralizado. Um sistema completo cliente/servidor Samhain é constituído dos seguintes componentes (SAMHAIN, s.d.):

- Verificador de integridade de arquivos/*host* Samhain – é o cliente/agente no host monitorado, projetado para ser executado como um daemon;
- Servidor de *log* Yule – coleta e registra relatórios de clientes Samhain em *hosts* remotos ou locais, funciona como um ponto central de coleta na LAN, mantendo sua organização (SCHÜRMANN, 2010);
- Base de dados relacional – armazena os relatórios de clientes, é compatível com Oracle, MySQL, PostgreSQL;
- Console Beltane, baseado em web – é uma aplicação PHP disponível como um pacote separado, permite a extração de relatórios da base de dados e sua apresentação para análise;
- Sistema de implantação – componente opcional para facilitar a implantação de clientes Samhain.

2.5.4 HLBR

Hogwash Light BR (HLBR) é uma ferramenta brasileira, criada no ano de 2005, mantido pelos líderes do projeto André Bertelli Araújo e João Eriberto Mota. É baseada no aplicativo Hogwash, desenvolvido por Jason Larsen, em 1996.

O HLBR é um IPS que filtra pacotes diretamente na camada 2 do modelo OSI, não necessitando do endereço IP na máquina em que está instalada. A detecção de intrusão é baseada em assinatura, e o usuário poderá adicionar novas regras.

É uma ferramenta versátil, podendo até ser utilizada como *bridge* para *honeypots* e *honeynets*. O HLBR pelo fato de não utilizar a pilha TCP/IP do sistema operacional, torna-se transparente na rede. Atualmente o projeto do HLBR se encontra interrompido, desde agosto de 2013, devido a *bugs* na ferramenta (HLBR, s.d.).

2.5.5 Snort

É uma ferramenta *open source* para detecção de intrusão, atuando como NIDS, desenvolvida por Martin Roesch no ano de 1998. Atualmente a empresa Sourcefire é

responsável pelo desenvolvimento e também distribui versões pagas da ferramenta. Atualmente o Snort encontra-se na versão estável 2.9.7.6.

A ferramenta analisa o conteúdo dos pacotes, fazendo a comparação com um amplo conjunto de regras, combinado ao método de inspeção baseado em anomalias.

O Snort pode operar em multiplataformas, tem uma capacidade de fornecer alertas em tempo real, incorporando mecanismos de alerta para *syslog*, arquivos específicos do usuário, soquete UNIX ou mensagens *popup* do Windows.

O Snort opera em três modos principais:

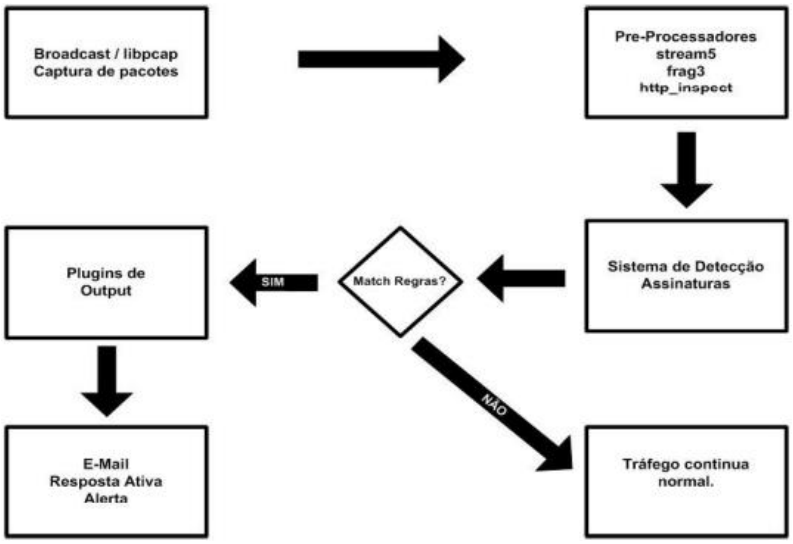
- Modo *sniffer* – faz a leitura dos pacotes da rede e os mostra no console;
- Modo *packet logger* – registra os pacotes capturados no disco;
- Modo de detecção de intrusão – é o modo mais completo e pode ser configurado, para analisar tráfego de rede comparando-o a um conjunto de regas predefinido pelo usuário.

A arquitetura do Snort é composta basicamente por quatro componentes:

- *Sniffer* – é o sensor do sistema, faz a captura dos pacotes;
- Pré-processador – seleciona e faz o envio dos pacotes;
- Motor de detecção – recebe os pacotes e faz a verificação em relação ao conjunto de regras estabelecidas.
- Saída – apresenta o resultado da análise em um arquivo de *log* (SNORT, s.d.).

O processo de captura dos pacotes e análise de tráfego do Snort pode ser visto na Figura 12.

Figura 12 – Processos do Snort.



Fonte: Montoro, 2012

Existe a possibilidade de tornar o Snort um IPS, isto acontece através do uso do mecanismo Snort-*inline*, permitindo que a ferramenta, além do registro e alerta da conexão suspeita, efetue um *drop* na mesma.

2.5.6 Comparativo entre ferramentas

Na análise destas ferramentas consideraram-se alguns aspectos relevantes de um sistema IDS/IPS, demonstrados na Tabela 1.

Tabela 1 – Comparativo entre ferramentas

Aspecto	OSSEC	Suricata	Samhain	HLBR	Snort
Tipo de sistema	HIDS	NIDS	HIDS	NIPS	NIDS
Plataformas suportadas	Multi plataforma	Multi plataforma	Posix	Unix	Multi plataforma
Licença	GNU/GPL	GNU/GPL	GNU/GPL	GNU/GPL	GNU/GPL
IPS nativo	Não	Sim	Não	Sim	Não
Suporte IPS	Sim	Sim	Não	Sim	Sim
Gera logs	Sim	Sim	Sim	Sim	Sim
Dispara alertas	Sim	Sim	Sim	Sim	Sim
Emite relatórios	Sim	Sim	Sim	Não	Sim
Análise em tempo real	Sim	Sim	Sim	Sim	Sim
Boa documentação	Sim	Sim	Sim	Não	Sim
Interface gráfica nativa	Não	Não	Não	Não	Não
Suporte à interface gráfica	Sim	Sim	Sim	Não	Sim
Multi *thread*	Não	Sim	Não	Não	Não
Aceleração de hardware	Não	Sim	Não	Não	Não

Fonte: do Autor.

Como pode ser visto na Tabela 1, as cinco ferramentas estudadas estão disponíveis sob a licença GNU/GPL, e apresentam algumas características em comum, tais como geração de *logs*, disparo de alertas, análise em tempo real e ausência de interface gráfica nativa. Os sistemas Samhain e OSSEC são baseados em *host*, não possuem IPS nativo, sendo que no OSSEC é possível habilitar suporte para o mesmo. Já o Snort e o Suricata são sistemas de detecção de intrusão baseados em rede, sendo que o Suricata tem IPS nativo, e o Snort só desempenha esta função se for habilitado o suporte. Uma diferença entre os dois NIDS é que

o Suricata possui um sistema de processamento multi *thread*, e também aceleração de hardware através de aproveitamento de GPU. A ferramenta HLBR é a única exclusivamente IPS, que no momento está com seu projeto descontinuado, aguardando soluções para seus problemas operacionais.

Diante das variadas opções de ferramentas IDS existentes, na execução do presente estudo, foi utilizado o IDS Snort V. 2.9.7.6, última versão estável, para verificação das simulações de ataques. O Snort pode ser considerado uma das ferramentas *open source* mais populares existentes. Segundo TRUHAN (2011), o Snort praticamente transformou-se em um padrão para detecção de intrusão *open source* e também está inserido como componente principal em algumas das ofertas comerciais mais avançadas de detecção. De acordo com SNORT (s.d.), atualmente esta ferramenta tem mais de 4 milhões de downloads e mais de 500.000 usuários registrados, tornando-se o IDS mais amplamente difundido na atualidade.

2.6 TRABALHOS CORRELATOS

O assunto referente a IDS e IPS é objeto de diversas pesquisas. Nesse sentido, serão destacados três trabalhos relacionados ao tema abordado no presente estudo.

SILVA e FONSECA (2007) apresentaram em sua monografia, para obtenção do título de especialista em Segurança de Redes de Computadores, da Faculdade Salesiana de Vitória, um estudo sobre tecnologias e serviços de redes locais, demonstrando formas de endereçamentos e tipos de protocolos. Também está descrito no trabalho algumas ameaças à segurança de redes locais, bem como algumas soluções para estes problemas.

Como resultado, o estudo feito por eles, mostra a criação de uma ferramenta protótipo com o objetivo de simplificar a utilização e implantação de sistema de detecção de intrusão (IDS) em uma rede local (LAN). Ela permite ao administrador, monitorar a autenticidade das máquinas ativas na rede e bloquear as que não estejam autorizadas na mesma. Este protótipo para redes locais é focado em autenticação e tem como finalidade comprovar se o *host* é realmente quem alega ser. A ferramenta em questão foi desenvolvida em módulos:

- Inicialização – responsável pelo início da execução do *script* e carregamento das listas de ACKs e variáveis globais;
- Varredura – fica escaneando a rede a procura de *hosts* conectados;
- Primeira validação – compara o endereço MAC das máquinas ativas na rede com um arquivo previamente estabelecido contendo os *hosts* autorizados;

- Segunda validação – tentativa de conexão entre o servidor e a máquina cliente através de SSH;
- Resposta – promove a ação baseada nos resultados dos módulos anteriores, sendo ele positivo, ou seja, o *host* não está autorizado, este módulo pode bloquear o IP da máquina ou simplesmente enviar um aviso;
- Auditoria – encarregado de armazenar os eventos coletados pelo protótipo, os dados armazenados são data e hora de ocorrência, o módulo que a originou, o tipo de evento, qual foi a resposta dada, dentre outros.

SILVA e FONSECA (2007) alegam ser este protótipo uma solução eficaz e de baixo custo no auxílio à detecção de intrusão em redes locais.

MEDEIROS (2008), em seu trabalho de conclusão apresentado como requisito parcial para obtenção do grau de especialista em Tecnologias, Gerência e Segurança de Redes de Computadores, da Universidade Federal do Rio Grande do Sul, realizou um estudo sobre segurança da informação e sistemas de detecção de intrusão (IDS). O estudo foi voltado à avaliação de algumas técnicas de geração de tráfego para a avaliação da eficiência de sistemas de detecção de intrusão e teve como objetivo a apresentação de um modelo virtual, sem implementação, para geração de tráfego real simulando *traces*, que englobasse tanto o cenário de ações normais, como também o de potenciais ataques.

Durante a realização de seu trabalho, MEDEIROS (2008) apresentou três formas de geração de dados para determinar a eficiência de um IDS:

- Análise sobre a geração de dados de forma real – que, segundo ele, se torna difícil uma análise mais aprofundada, pois sofre restrições políticas e pode causar algum prejuízo, pois se trata de dados reais;
- Análise sobre geração de dados de forma artificial – conforme o autor, esta prática elimina os problemas de política e privacidade do modelo anterior, já que ela somente faz uma simulação na rede real, porém, são apenas técnicas que se aproximam da realidade;
- Análise sobre a geração de dados sobre uma rede de testes criada – esta técnica, no trabalho apresentado, é a mais comum em testes de IDS, porém é a de maior dificuldade, já que se deve criar desde a rede a ser testada, o mais próximo possível da real, até as simulações a serem empregadas.

Como resultado de seu trabalho, MEDEIROS (2008) mostrou, sem implantar, um modelo de ambiente simulado separando os tráfegos em normais e anômalos, ficando a critério do avaliador o que deseja simular.

SILVA e JULIO (s.d.) em seu artigo publicado na Revista Infra Magazine 1, apresentaram o conceito de sistema de detecção de intrusão (IDS), suas finalidades e formas de detecção. Também analisaram os tipos de IDS e considerações acerca dos mesmos. Além disto, apresentaram como resultado, a implementação da ferramenta de IDS Prelude, identificando seus componentes, formas de funcionamento e um guia de instalação.

3 IMPLEMENTAÇÃO DE UM SISTEMA IDS

Neste capítulo é descrita a metodologia e o processo de implementação utilizado para alcançar os objetivos propostos.

3.1 Metodologia

Com o intuito de atingir os objetivos de implementar e configurar, de forma atualizada, um sistema de detecção de intrusão, gerando um roteiro de instalação, analisar sua eficácia nas detecções e verificar suas influências no desempenho do sistema, foi criado um ambiente com duas máquinas virtuais, uma denominada SnortTCC e outra de Atacante.

No servidor SnortTCC foi instalado e configurado o Snort, juntamente com as ferramentas BASE, Monitor do sistema e Top, necessárias para coletar as informações utilizadas na posterior análise. Já o Atacante está configurado com os softwares T50 e Nikto, que serão utilizados para executar as simulações de ataque.

Com o ambiente criado e configurado, foram realizados os testes. A partir do Atacante, tendo como alvo o SnortTCC, foi simulado um teste de conectividade, um scan de vulnerabilidades e um ataque DoS.

Para analisar os resultados, primeiramente foram coletados dados referentes ao sistema sem o IDS estar em funcionamento, durante e após sua inicialização, e no decorrer das simulações dos ataques. Foi analisada a eficiência do IDS nas detecções das simulações, e a sua influência no desempenho do servidor SnortTCC, referente ao uso de CPU, memória e fluxo de rede. Para detalhar estas análises foram criados gráficos comparativos.

3.2 Ambiente de teste

Para a realização dos testes com a ferramenta Snort, trabalhando como IDS, e posterior análise dos resultados, inicialmente foi criado um ambiente virtual com o software de virtualização VMware Workstation 10.0.3. Neste ambiente foi implementado, para a execução dos testes, um servidor contendo o Snort versão 2.9.7.6, com seu sensor escutando a interface de rede eth0 do mesmo, desta forma operando como NIDS. Este servidor, denominado SnortTCC, tem como sistema operacional a última versão estável do Debian

GNU/Linux 8 (Jessie), e tem como características de hardware um processador com 4 núcleos, 4 GB de memória RAM e HD com 30GB.

Para a execução das simulações de ataques ao servidor, onde está instalado o sistema de detecção de intrusão, foi criada uma máquina virtual, para dar origem às intrusões, com o sistema operacional Kali Linux 2.0, chamada de Atacante.

3.3 Instalação do Snort como IDS

Para a execução dos testes, utilizando o Snort, foi feita a instalação de uma série de ferramentas que irão trabalhar em conjunto, além do próprio Snort, foram implementados o MySQL, PHP, Apache, Barnyard, BASE e Pulledpork. Este roteiro de instalação é baseado na documentação disponibilizada no site oficial do Snort (SNORT, s.d.). Foram feitas modificações para que o processo se adequasse às versões atualizadas de todas as ferramentas utilizadas. Este procedimento acarretou em uma série de dificuldades, pois não foi encontrado material que contemplasse a instalação e configuração do conjunto das ferramentas, com suas versões atualizadas, dando origem a diversas incompatibilidades entre as versões das mesmas, que foram sendo resolvidas ao longo do processo, gerando o roteiro apresentado no Apêndice A.

3.3.1 Ferramentas integradas ao Snort

Após a configuração do Snort, conforme arquivo apresentado no Apêndice B, foi instalado o Barnyard2, para compor o IDS, que é um interpretador, de código aberto, para os arquivos *logs* de saída em formato (*unified2*) do Snort (BARNYARD2, s.d.). Na verdade, ele aumenta a eficiência do sistema, funcionando como um *buffer* entre o Snort e o MySQL, processando os *logs* de saída, inclusive aqueles em banco de dados. Para tanto, foi necessário criar um *script* de inicialização, conforme Apêndice C.

Além disso, foi instalado o BASE (*Basic Analysis and Security Engine*), que é uma ferramenta, escrita em PHP, que apresenta uma interface web para consultar e analisar os alertas provenientes do Snort. Ele permite obter informações sobre estatísticas de alertas, de números de acessos divididos por protocolo, por porta de origem e destino, e diversas outras informações, todas extraídas dos *logs* gerados.

Para que o Snort funcione em conjunto com o BASE, é necessário um sistema de gerenciamento de banco de dados (SGBD) para armazenar os dados capturados para posterior

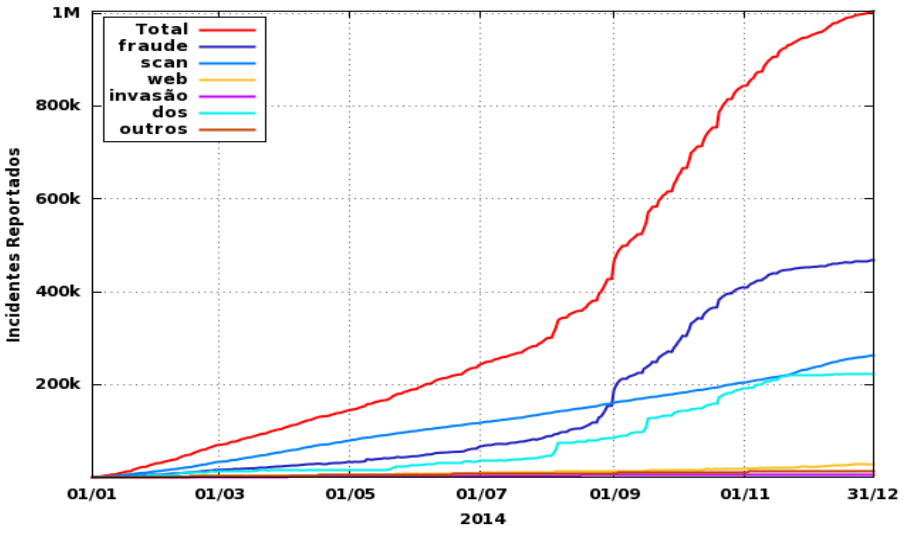
monitoramento dos *logs*. Nesse caso, foi utilizado o MySQL como banco de dados para armazenar os alertas do Snort.

Outra ferramenta instalada é o Pulledpork, escrita em *Perl*, ele é utilizado para gerenciamento do conjunto de regras do Snort, e irá permitir o *download* e atualização automática das regras[1]. Para usar as regras certificadas do **Sourcefire VRT** é necessário acessar o site **snort.org**, e registrar uma conta para obter uma **oinkcode,** que irá permitir o *download* do conjunto de regras de usuário registrado. Para o trabalho foram usadas as regras da comunidade, não necessitando o registro.

3.4 Execução dos Testes

Neste item está descrita a metodologia utilizada para executar as três simulações de intrusão. A escolha dos tipos de ataque a serem utilizados, para a posterior análise dos resultados, foram determinados, com base em informações de incidentes reportados ao CERT no ano de 2014, onde o Scan de vulnerabilidades e o DoS foram os que tiveram maior crescimento ao longo do ano, conforme mostra a Figura 13. Precedendo esses ataques, está o teste de conectividade que também foi empregado nesse trabalho.

Figura 13 – Incidentes reportados ao CERT.br – 2014 (Acumulado).



Fonte: CERT.br

[1] As regras profissionais estão disponíveis em http://www.emergingthreatspro.com e http://www.snort.org.

3.4.1 Teste de conectividade

Simulando o primeiro passo de um ataque a um servidor, foi utilizado o comando **"ping"** a partir da máquina Atacante para o servidor SnortTCC. Este comando é usado para verificar se um determinado servidor está conectado a rede e qual o tempo de resposta do mesmo, para que isso aconteça ele envia pacotes ICMP *ECHO_REQUEST* para o servidor, e espera a resposta *ECHO_REPLY (*Silva, 2010).

Para executar a simulação foi executado, na máquina Atacante, o comando da seguinte forma:

```
#ping –c 10 192.168.159.132
```

Onde o parâmetro **"-c"** determina o número de pacotes a ser enviado, que nesta simulação foi definido como 10.

3.4.2 Scan de Vulnerabilidades

Foi usado, nessa simulação, o Nikto *Web Scanner* 2.1.5, ferramenta *Open Source* (GLP), desenvolvida em *perl*, destinada a scannear servidores, abrangendo uma ampla variedade de itens, incluindo programas e arquivos maliciosos, versões desatualizadas ou problemas de versão em servidores, arquivos de índices múltiplos, opções de servidores HTTP, entre outros (CIRT, 2015). Essa ferramenta encontra-se disponível nas aplicações do Kali, sistema operacional da máquina Atacante.

A simulação foi feita através do seguinte comando:

```
#nikto –h 192.168.159.132
```

Onde o parâmetro **"-h"** é obrigatório para que todas as opções de comando estejam habilitadas.

3.4.3 DoS

Para efetivar a simulação de negação de serviço, foi usada a ferramenta T50, que é um injetor de pacotes utilizado na execução de testes de stress em uma rede, permitindo simular ataques Dos e DDos. Esta ferramenta foi desenvolvida pelo brasileiro Nelson Brito, é liberada sob a licença (GPLv2), permitindo disparar simultaneamente pacotes de diferentes protocolos (MORENO, 2015). Na execução da simulação, foi usado o seguinte comando:

```
#t50 192.168.159.132 - -flood -S - -turbo - -dport 80
```

Neste comando os seguintes parâmetros são definidos: **“- -flood”** substitui o *threshold*, **“-S”** inicia a conexão através do TCP SYN Flag, **“- -turbo”** aumenta a potencialidade do ataque e **“- -dport”** define a porta a ser atacada.

Para a coleta dos dados, na execução desta simulação, foi definido um tempo de 3 minutos, já que esse tipo de teste é continuo.

3.4.4 Ferramentas de monitoramento

Durante cada simulação, foram realizados monitoramentos no sistema, referentes ao uso da CPU, memória e fluxo de rede. Para isso, foi usado o monitor do sistema do Debian, e também o gerenciador de processos através do comando **“top”**.

O monitor do sistema foi acessado pressionando as teclas **“Alt+F2”** e no “Diálogo de Execução” digitado: **“gnome-system-monitor”**. Ele ficou ativo, desde antes da inicialização do Snort até ter sido feita a última simulação, registrando em tempo real, o uso da CPU, memória e fluxo de rede.

Para que se pudesse obter os dados referentes ao uso da CPU, de forma mais precisa possível, foi executado o comando **“top”**, da seguinte forma:

```
#top -d 60
```

Este comando dá acesso a um gerenciador de processos do Debian, que permite, através do parâmetro **“-d”**, estabelecer um intervalo de tempo onde o uso da CPU é registrado para cada processo. Dessa forma, foi definido o tempo de um minuto para cada leitura, ou seja, o gerenciador fica registrando quanto de uso da CPU é feito durante esse intervalo de tempo. A cada início da simulação de ataque, o gerenciador é ativado e fica monitorando pelo próximo minuto, exibindo ao final a quantidade aproximada de uso da CPU que o processo utilizou.

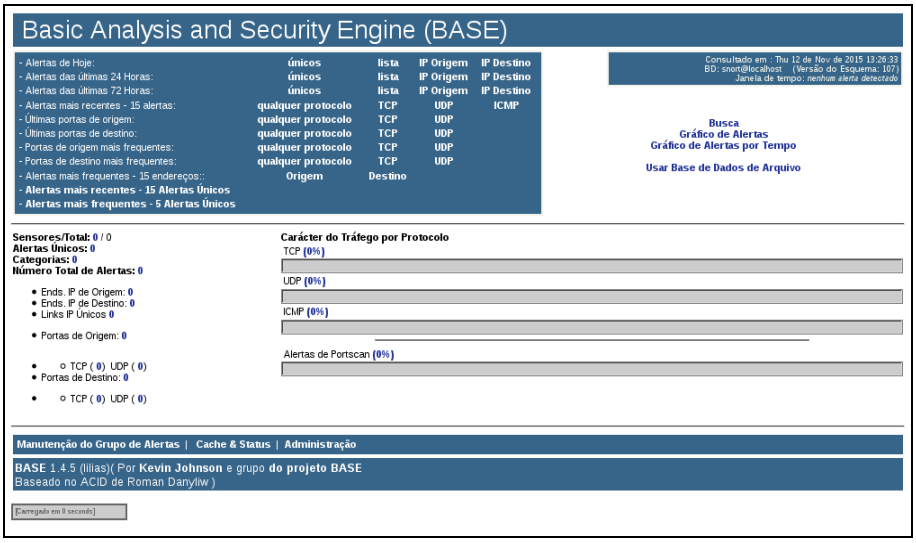
4 RESULTADOS

Este capítulo apresenta os resultados das simulações de ataques executadas de acordo com a metodologia proposta no capítulo anterior. Após a instalação e configuração do IDS Snort, juntamente com as ferramentas integradas para seu funcionamento, para obter os resultados dos testes, foram executadas três simulações de ataques. Na primeira simulação foi testada a conectividade entre a máquina Atacante e o servidor SnortTCC. A segunda simulação foi realizada utilizando um Scan de vulnerabilidades. A última simulação foi baseada em um ataque do tipo DoS.

4.1 Características Pré-simulação

A tela inicial do BASE, sem qualquer tipo de alerta detectado pelo Snort, está representada na Figura 14.

Figura 14 – Tela inicial do BASE.



Fonte: do Autor

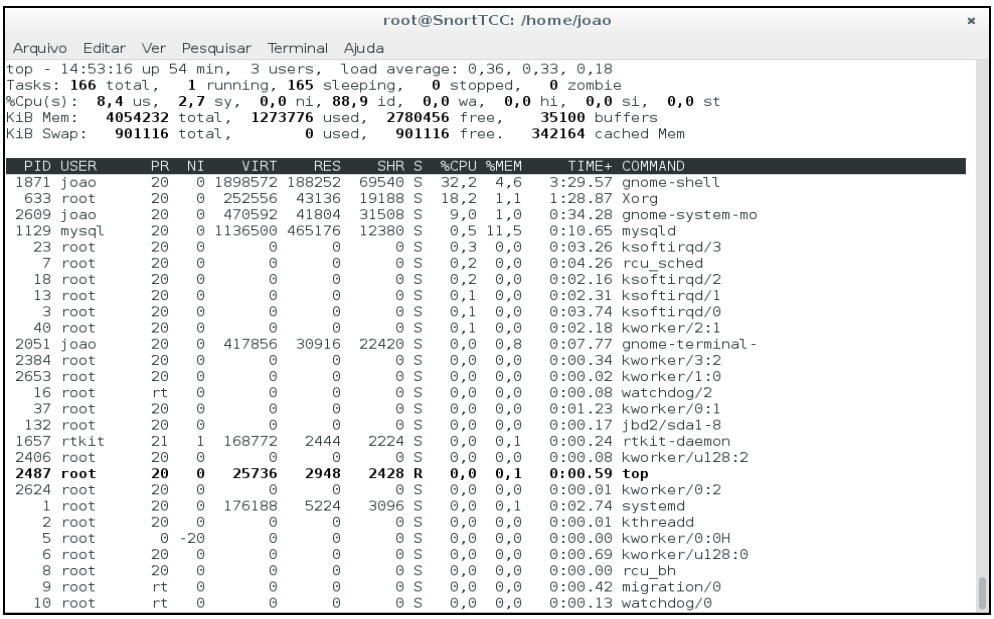
A Figura 15 exibe a tela do monitor do sistema, com as informações relativas somente aos processos do sistema operacional, sem a ativação do conjunto de ferramentas necessárias para o funcionamento do IDS.

Figura 15 – Tela do Monitor do sistema – sem o IDS.

Fonte: do Autor

Os processos, relativos ao sistema operacional, estão demonstrados na Figura 16, através do Top, antes da inicialização do IDS.

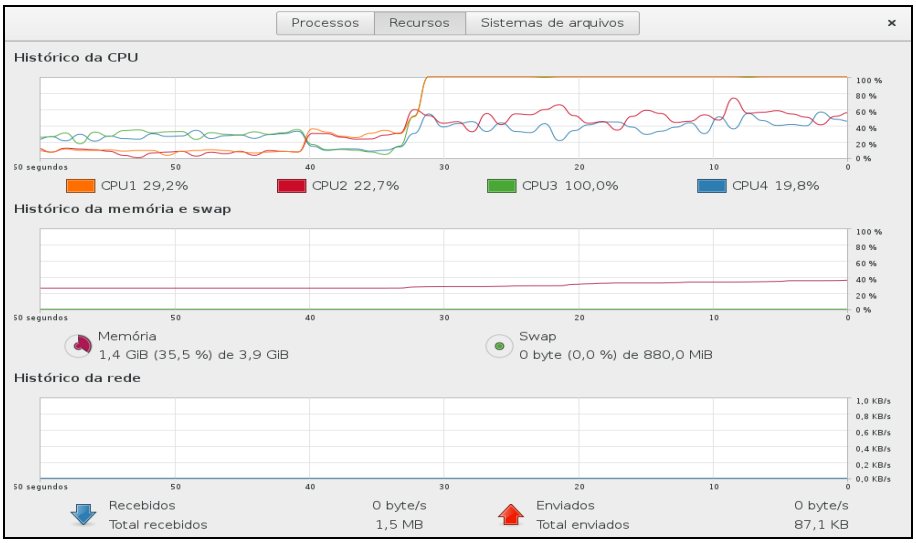
Figura 16 – Tela do Top – sem o IDS.



Fonte: do Autor

O resultado apresentado na Figura 17 é referente ao momento da inicialização do conjunto de ferramentas necessárias ao funcionamento do IDS Snort, onde é possível visualizar um significativo aumento na taxa de uso da CPU, assim como na demanda por memória.

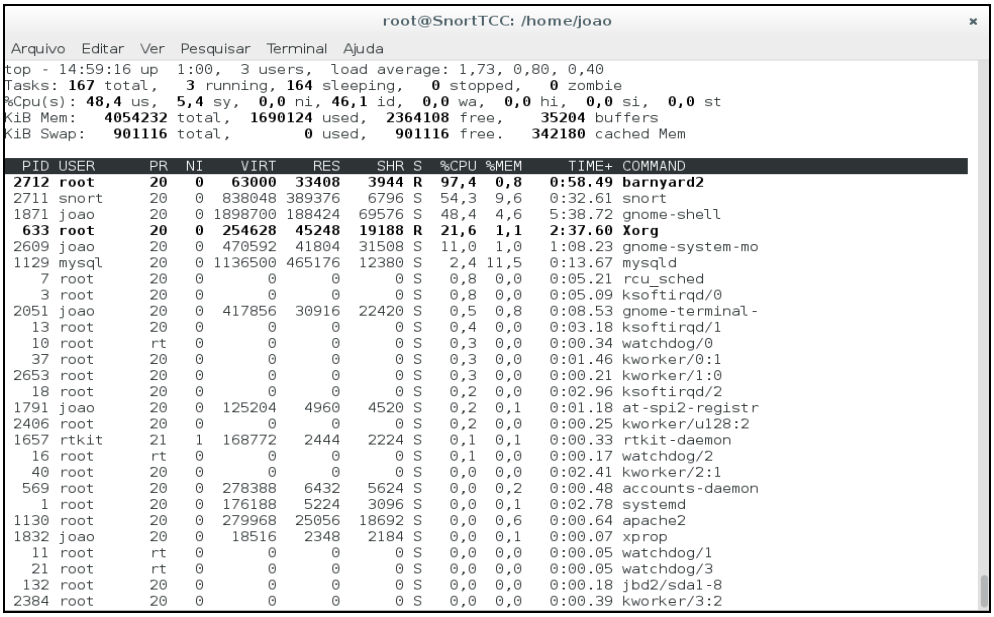
Figura 17 – Tela do Monitor do sistema – inicialização do IDS.



Fonte: do Autor

Durante a inicialização do IDS, foi registrado pelo Top, uma taxa elevada de uso da CPU no servidor SnortTCC, como demonstra a Figura 18. Esse processo demorou em torno de 3 minutos.

Figura 18 – Tela do Top – inicialização do IDS.

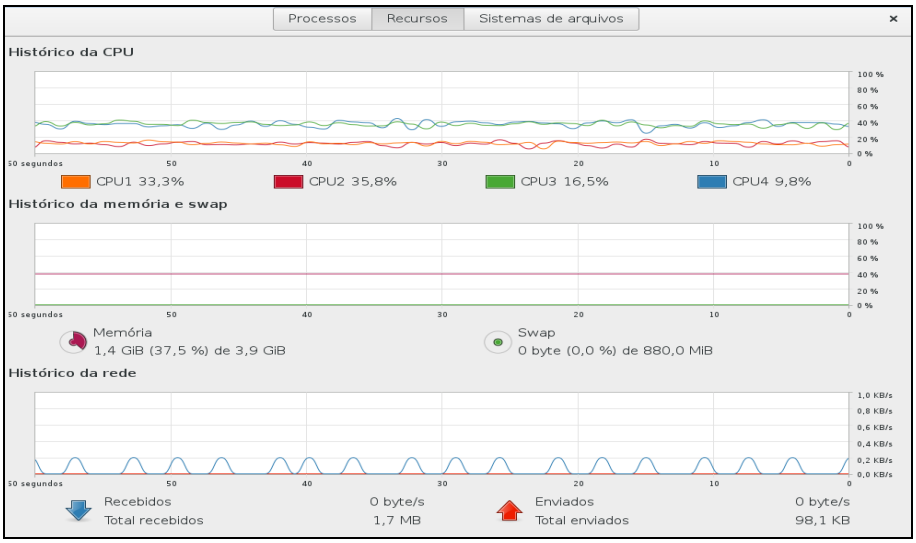


Fonte: do Autor

Após o processo de inicialização, o IDS estabiliza, e com o auxilio do Monitor do sistema, exibido na Figura 19, é possível avaliar que não há aumento significativo na taxa de uso da CPU. Já com relação ao uso de memória, o percentual dobrou, quando comparado ao

estado antes da ativação do IDS. Também ficou alterada a quantidade de fluxo de dados na rede, devido ao sensor do IDS estar ativo, recebendo informações.

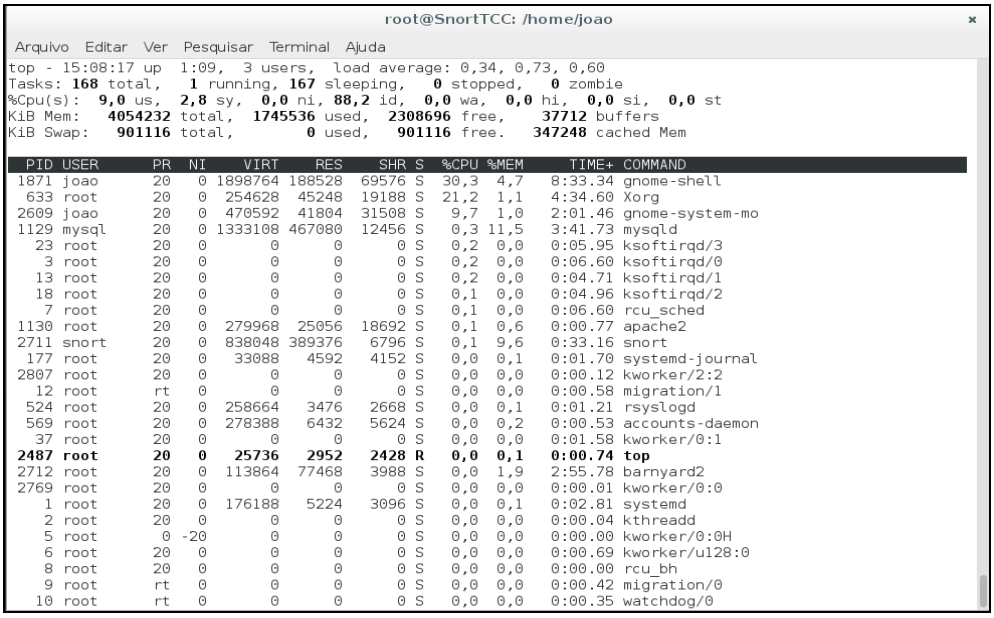
Figura 19 – Tela do Monitor do sistema – IDS ativo.



Fonte: do Autor

A Figura 20, referente à tela do Top, comprova que não há um aumento significativo de uso da CPU, após a inicialização e estabilização do conjunto de ferramentas necessárias ao funcionamento do IDS.

Figura 20 – Tela do Top – IDS ativo.



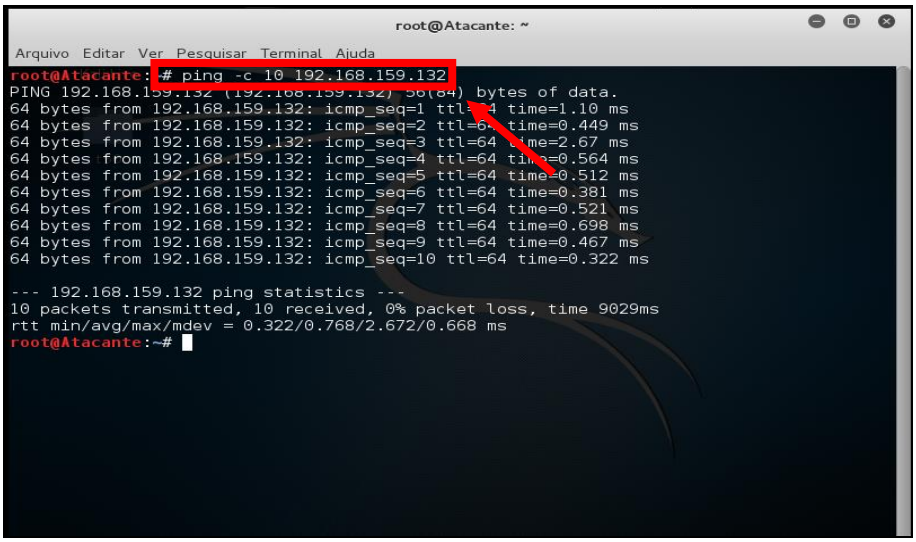
Fonte: do Autor

4.2 Simulação do teste de conectividade

Esta seção apresenta os resultados da simulação para testar a conectividade entre a máquina Atacante e o servidor SnortTCC, a detecção feita pelo IDS Snort e os gráficos da taxa de uso da CPU durante a simulação.

A Figura 21 mostra o comando **"ping"** sendo executado no terminal da máquina Atacante, para testar a conectividade com o servidor SnortTCC, e o resultado desse processo, juntamente com o tempo do *Echo Reply* do ICMP e sua latência.

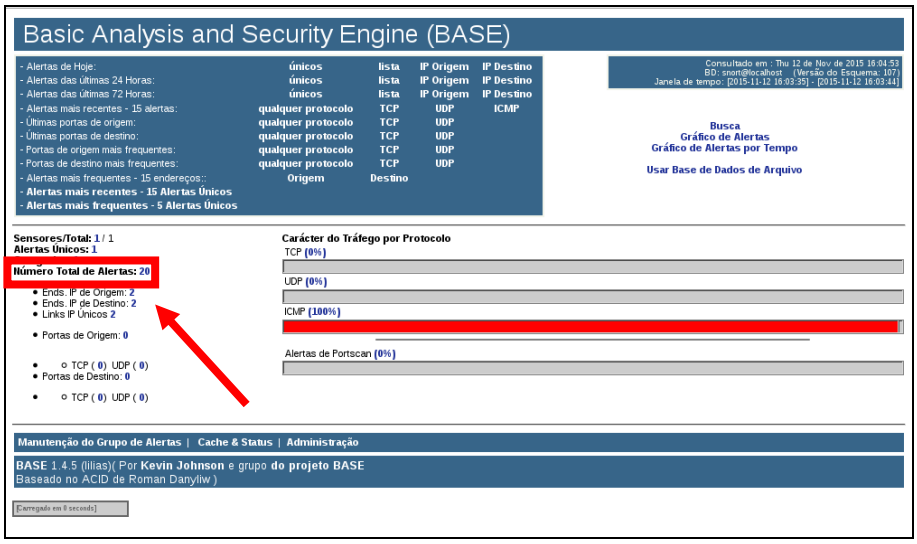
Figura 21 – Execução do comando "ping".



Fonte: do Autor

Na Figura 22, pode ser observada a detecção do teste de conectividade feita pelo Snort mostrada na tela do BASE. Foi gerado um total de 20 alertas, que correspondem a soma das 10 mensagens *Echo Request* enviadas pela máquina Atacante e as 10 mensagens *Echo Reply* produzidas pelo servidor SnortTCC.

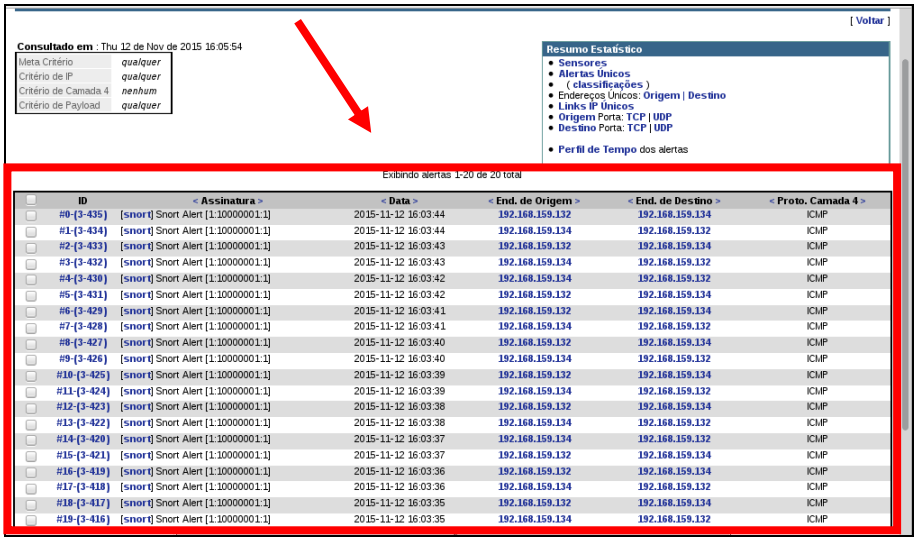
Figura 22 – Tela inicial do BASE (teste de conectividade).



Fonte: Do autor

O detalhamento dos 20 alertas gerados pelo Snort pode ser visto na Figura 23. São listadas informações como: assinatura do que causou o alerta, data e hora da detecção, endereços de origem e destino e o tipo de protocolo usado, que no caso foi o ICMP.

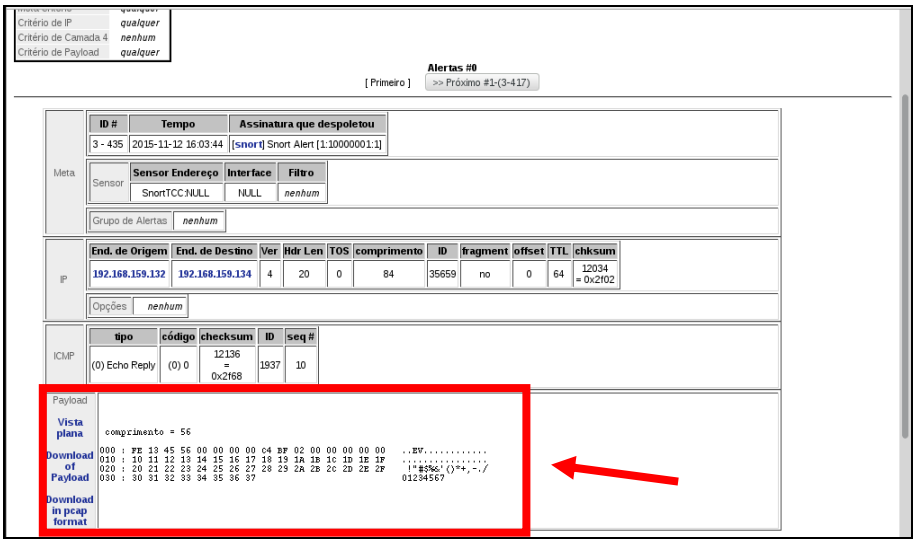
Figura 23 – Detalhamento dos alertas gerados (teste de conectividade).



Fonte: do Autor

Para se obter maiores informações, a respeito de cada alerta, inclusive seu *payload*, foi criada a tela de especificações individuais para o alerta de ID #0, como está descrito na Figura 24.

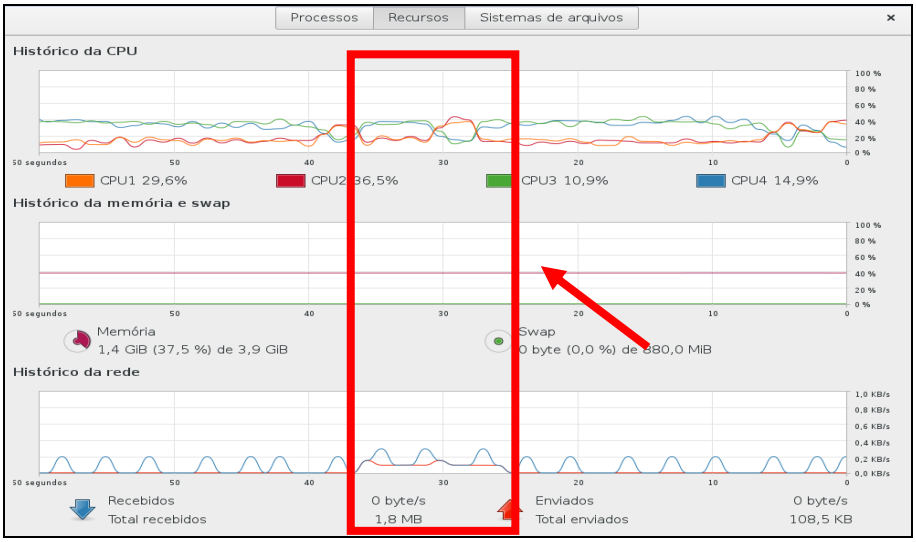
Figura 24 – Especificações individualizadas dos alertas (teste de conectividade).



Fonte: do Autor

A Figura 25 exibe as informações de desempenho do sistema no servidor SnortTCC, relativas à percentagem do uso da CPU, memória e fluxo de rede no instante em que o teste está ocorrendo. Observa-se uma variação no fluxo de rede referente ao recebimento e envio dos pacotes. Nos gráficos de uso da CPU e uso de memória não foi possível visualizar alterações.

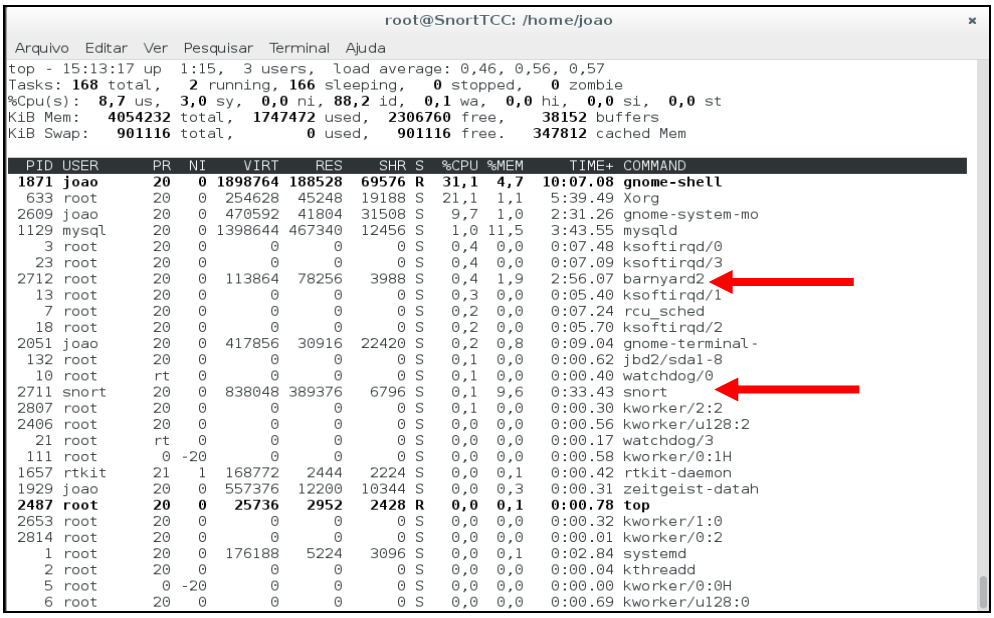
Figura 25 – Informações do Monitor do sistema (teste de conectividade).



Fonte: do Autor

Com o Top foi possível visualizar a taxa de uso da CPU durante a detecção do teste de conectividade. A Figura 26 apresenta também a variação de uso da CPU, referente ao conjunto de processos do Snort.

Figura 26 – Informações do sistema com o Top (teste de conectividade).



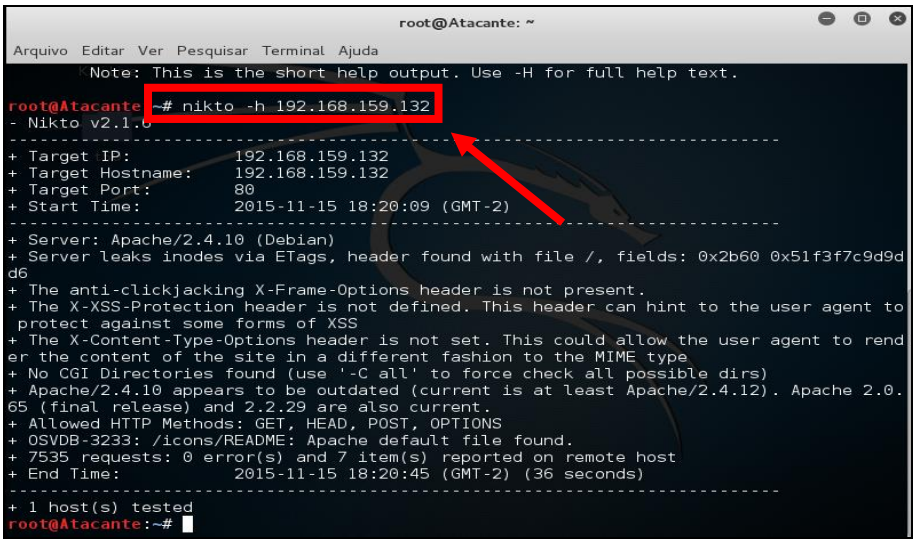
Fonte: do Autor

4.3 Simulação do Scan de vulnerabilidades

A seguir estão descritas as telas, com os resultados, da simulação do Scan de vulnerabilidades feito no servidor SnortTCC pela máquina Atacante.

Na Figura 27 é apresentada a tela do terminal da máquina Atacante na execução do aplicativo Nikto, usado para realizar esta simulação, nela estão exibidas as vulnerabilidades encontradas no servidor alvo.

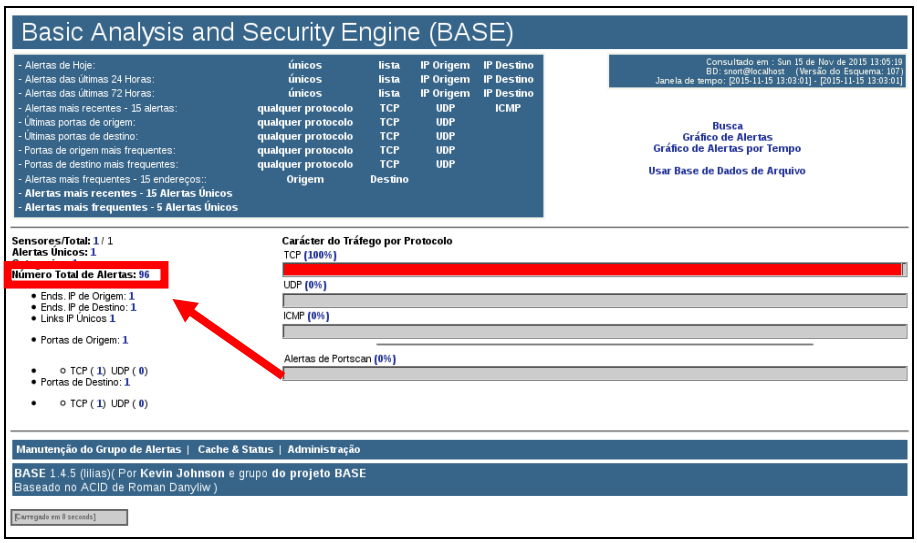
Figura 27 – Tela com o Nikto sendo executado.



Fonte: do Autor

A tela inicial do BASE, mostra um total de 96 tentativas de intrusão detectadas pelo Snort, nessa simulação conforme mostra a Figura 28.

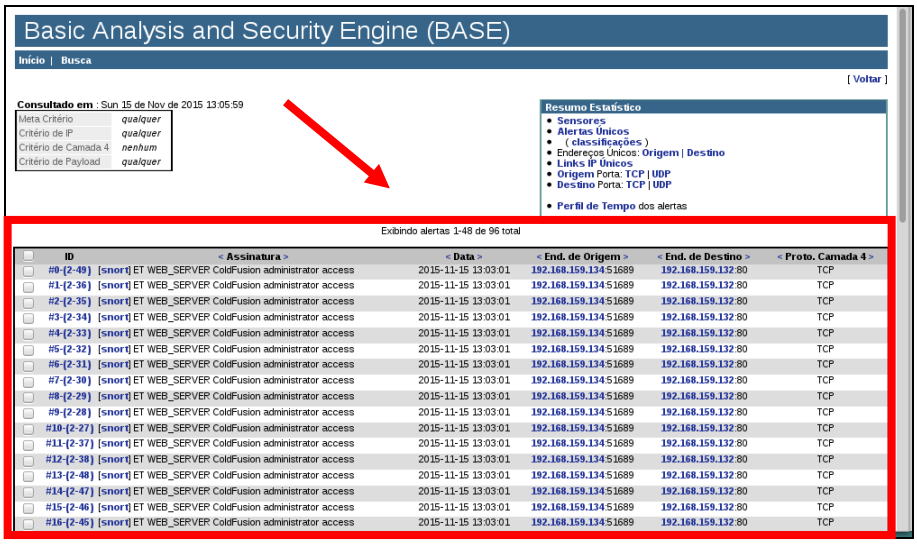
Figura 28 – Tela inicial do BASE (scan de vulnerabilidades).



Fonte: do Autor

A Figura 29 apresenta os alertas gerados pelo Snort com suas respectivas informações, relativas à assinatura causadora do alerta, data e hora do mesmo, endereços de origem e destino e o protocolo, que neste caso foi o TCP, empregado na intrusão.

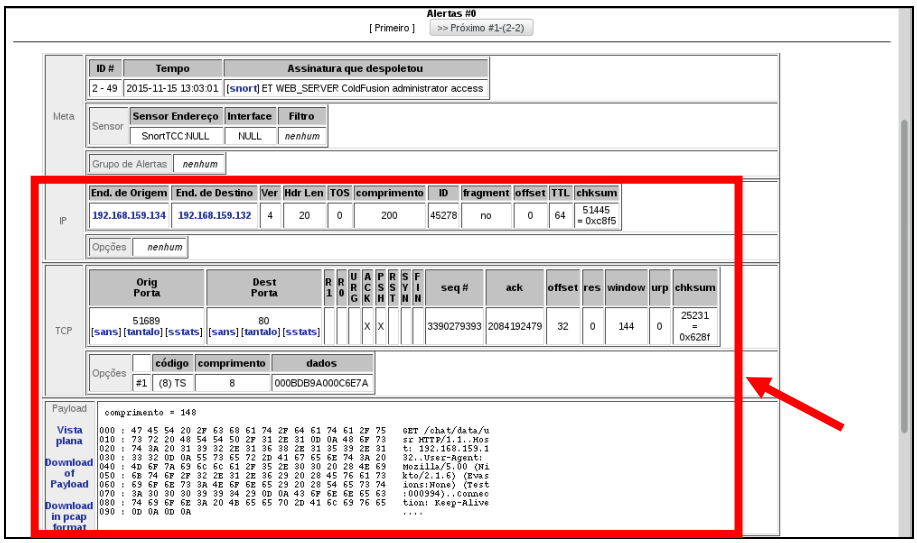
Figura 29 – Detalhamento dos alertas gerados (scan de vulnerabilidades).



Fonte: do Autor

As especificações individuais do alerta ID #0 estão exibidas na Figura 30, que mostra detalhes sobre o IP, o protocolo TCP com suas *flags* detectadas ACK e PSH, e o *payload*.

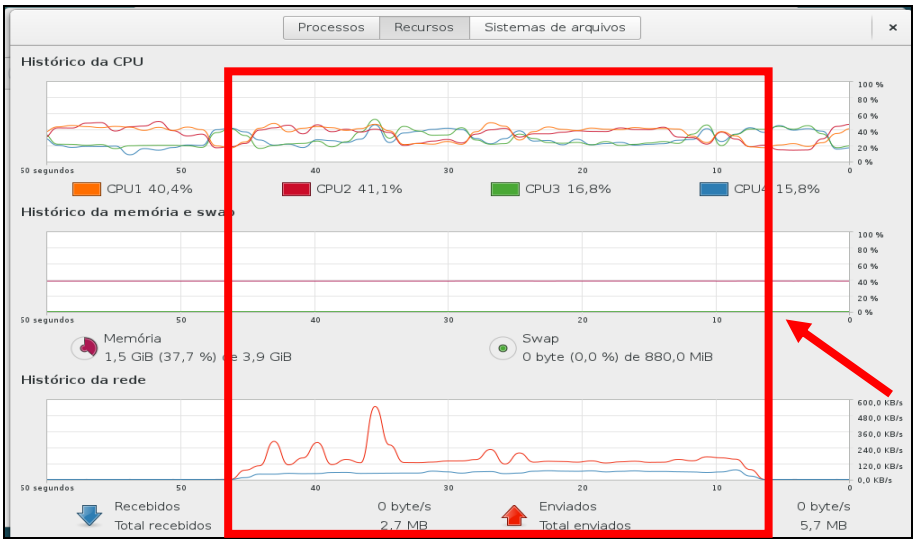
Figura 30 – Especificações individualizadas dos alertas (scan de vulnerabilidades).



Fonte: do Autor

É possível perceber, na tela do Monitor do sistema, que durante a simulação, a quantidade de informações enviadas pelo servidor aumentou bastante devido às respostas das requisições feitas pela máquina Atacante, já no uso de CPU e memória não houve alterações perceptíveis com o uso do monitor do sistema, como mostra a Figura 31.

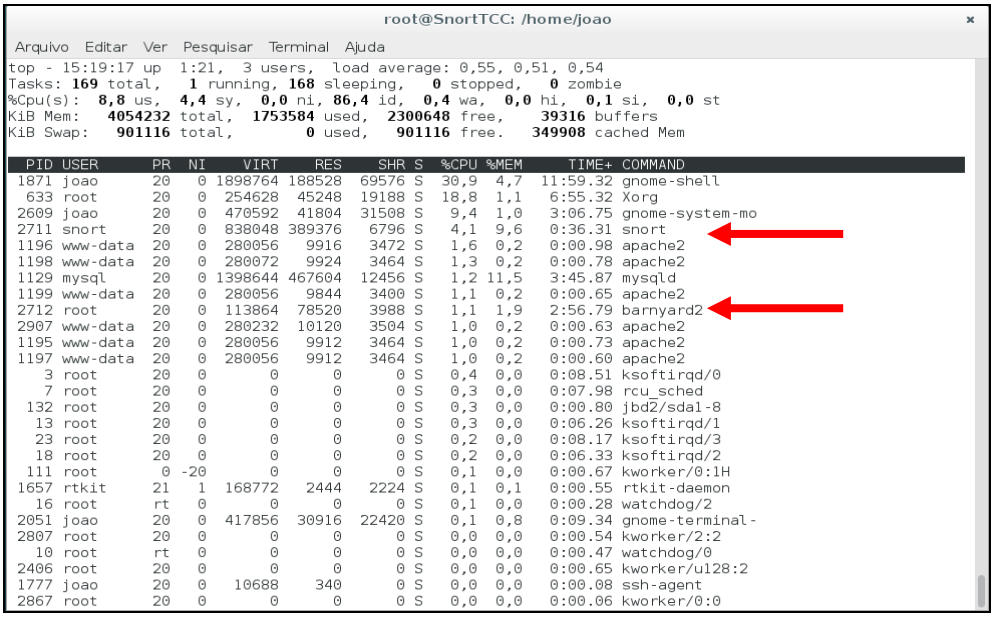
Figura 31 – Informações do Monitor do sistema (scan de vulnerabilidades).



Fonte: do Autor

A variação do uso da CPU, durante esta simulação, pode ser visualizada na tela do Top, como mostra a Figura 32, juntamente com a taxa de uso da CPU relativa ao conjunto de processos do IDS.

Figura 32 – Informações do sistema com o Top (scan de vulnerabilidades).



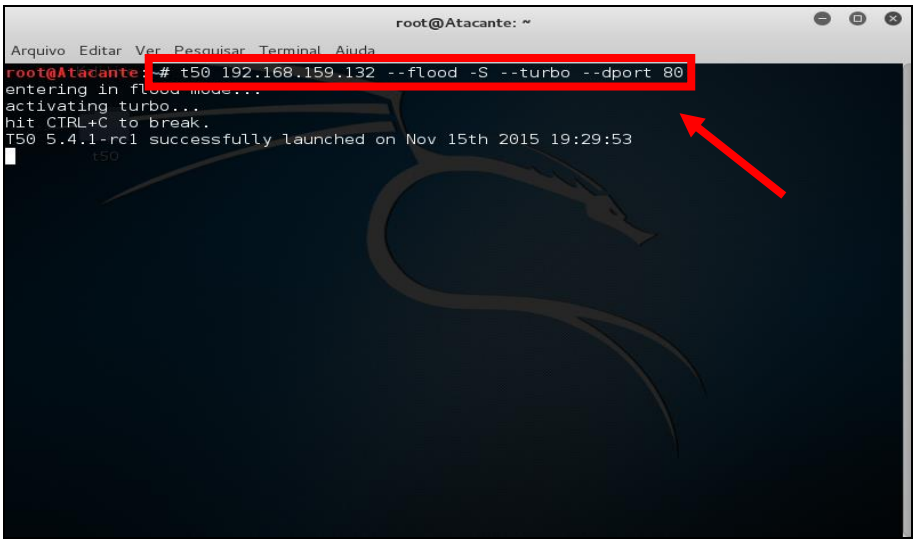
Fonte: do Autor

4.4 Simulação de ataque DoS

Com a utilização da ferramenta T50, foi simulado um ataque do tipo DoS, onde a máquina Atacante envia várias requisições ao servidor SnortTCC com a finalidade de que ele comece a negar o serviço.

Para a realização do teste, no terminal da máquina Atacante, foi executado o comando que dispara o ataque. Na Figura 33, é possível visualizar o sucesso da intrusão.

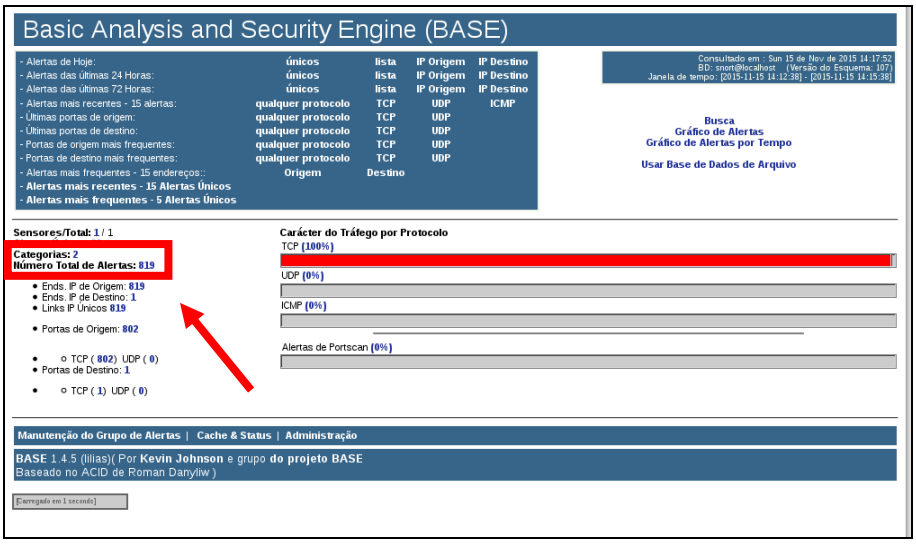
Figura 33 – Tela do T50 sendo executado.



Fonte: do Autor

A detecção da simulação do ataque de DoS foi bem sucedida. Na tela do BASE, na Figura 34, é exibido um total de 819 alertas gerados pelo Snort durante a simulação. Diferentemente das outras simulações, o IDS detectou duas categorias de alertas, isso porque a ferramenta T50 simula vários tipos de requisições. Todas utilizaram o protocolo TCP.

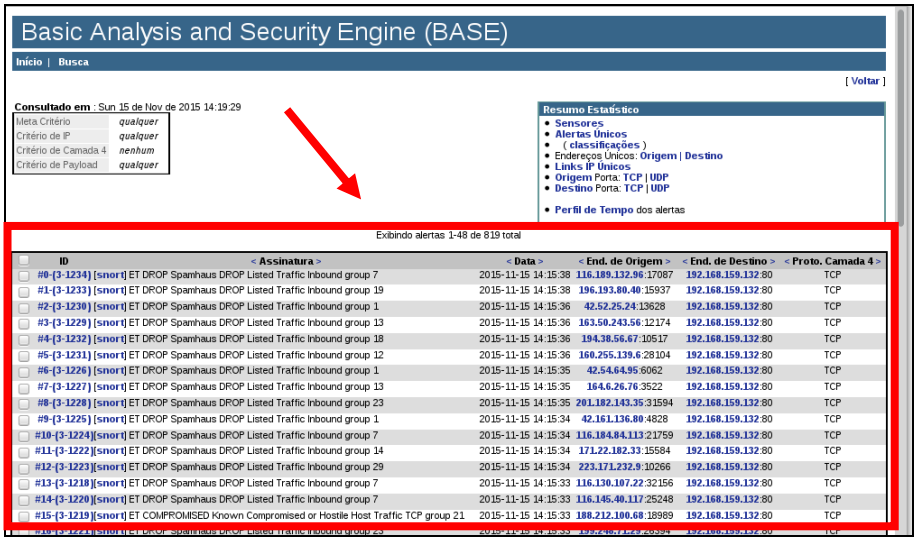
Figura 34 – Tela inicial do BASE (DoS).



Fonte: do Autor

A Figura 35 apresenta os alertas gerados pelo Snort resultantes desta simulação, com as informações detalhadas de cada um, incluindo as diferentes assinaturas que geraram os mesmos.

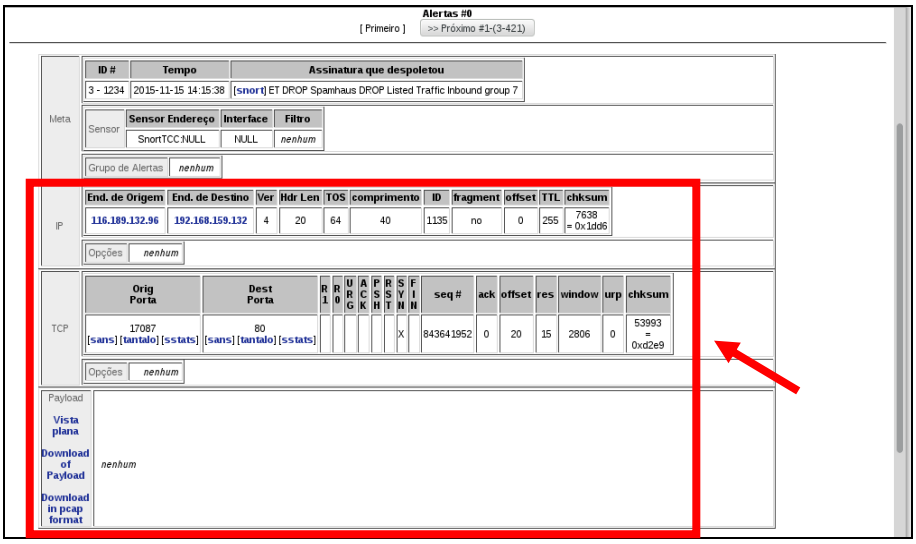
Figura 35 – Detalhamento dos alertas gerados (DoS).



Fonte: do Autor

No detalhe do alerta ID #0 não há o *payload*, pois o objetivo do ataque não é efetuar um GET no servidor, e sim inundar o mesmo com *flags* SYN, sem se importar com o SYN-ACK. Além disso, são apresentadas na Figura 36, outras informações sobre o IP e protocolo TCP do alerta.

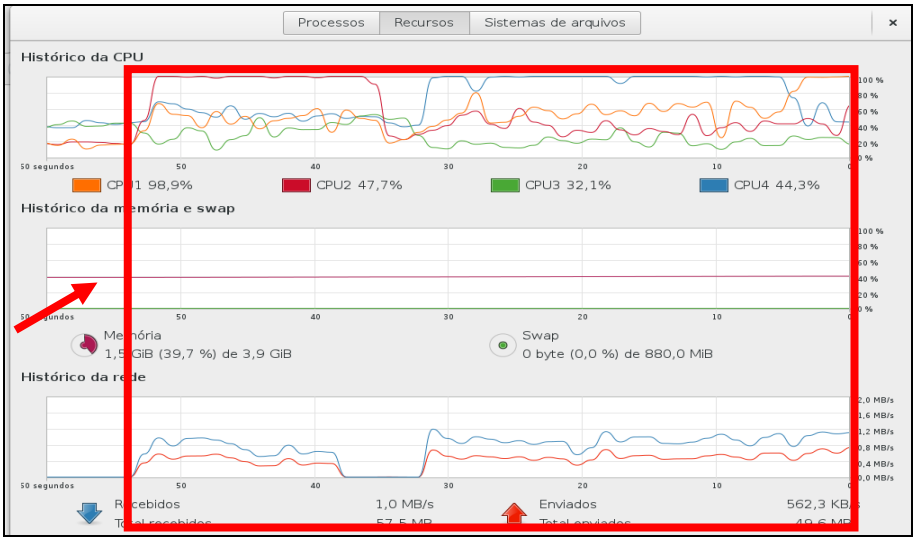
Figura 36 – Especificações individualizadas dos alertas (DoS).



Fonte: do Autor

O Monitor do sistema indicou um grande aumento no uso da CPU durante a realização da simulação de DoS, também mostrou uma variação significativa no fluxo da rede, conforme a Figura 37.

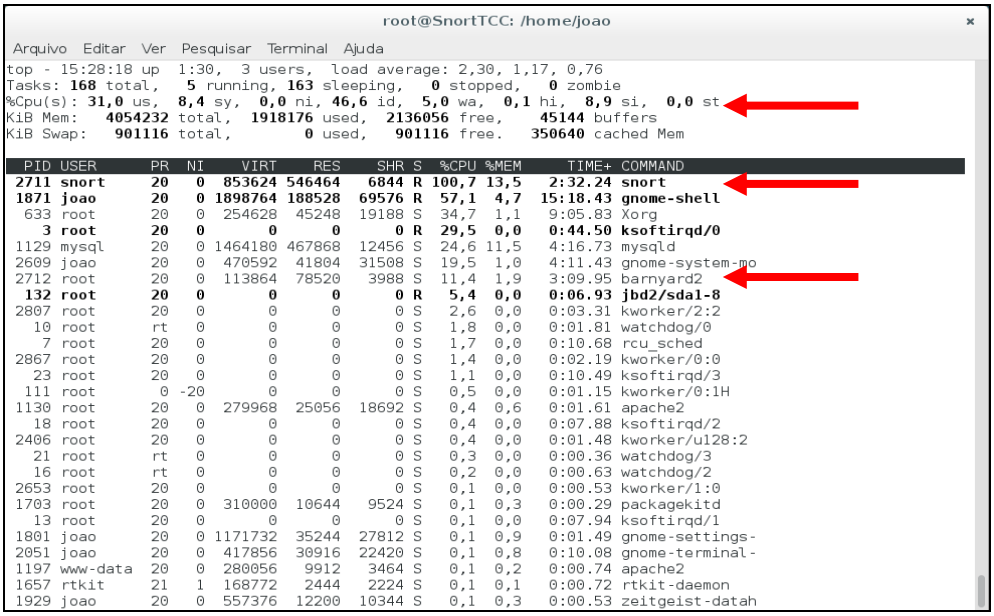
Figura 37 – Informações do Monitor do sistema (DoS).



Fonte: do Autor

Pode ser visualizada na tela do Top, como mostra a Figura 38, a variação da taxa de uso da CPU, e a variação de uso da CPU, relativa ao conjunto de processos do IDS usados na detecção.

Figura 38 – Informações do sistema com o Top (DoS).



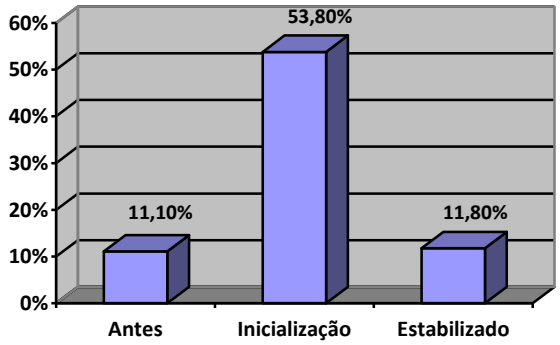
Fonte: do Autor

4.5 Análise dos resultados

O IDS Snort se mostrou eficaz na detecção das simulações feitas no presente trabalho, detectando as intrusões propostas e gerando alertas nas três situações testadas.

Durante a inicialização do conjunto de ferramentas do IDS, verificou-se um aumento significativo na taxa de uso da CPU, obtida através do Top, que antes do início do processo, era de aproximadamente 11,1% e passou para 53,8% em sua variação máxima. Após a estabilização das ferramentas que constituem o IDS, a taxa ficou reduzida para aproximadamente 11,8%, conforme a Figura 39. Com isso é possível observar que o IDS Snort consome aproximadamente 0,7% de uso da CPU estando ativado, sem nenhuma detecção concretizada.

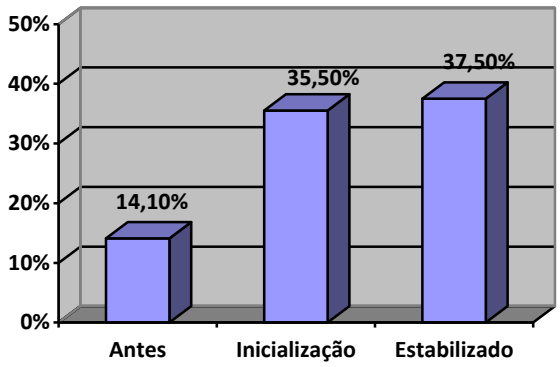
Figura 39 – Taxa aproximada do uso da CPU.



Fonte: do Autor

O monitor do sistema mostrou ainda que, antes da inicialização do IDS, a demanda aproximada por memória era de 14,1%, durante a inicialização aumentou para 35,5% e depois de estabilizado o conjunto do sistema do IDS, permaneceu em aproximadamente 37,5%, conforme a Figura 40. Dessa forma, verificou-se que a ativação do conjunto de ferramentas necessárias para que o IDS funcione, acarretou em um aumento em torno de 23,4% no uso de memória.

Figura 40 – Taxa aproximada de uso de memória.



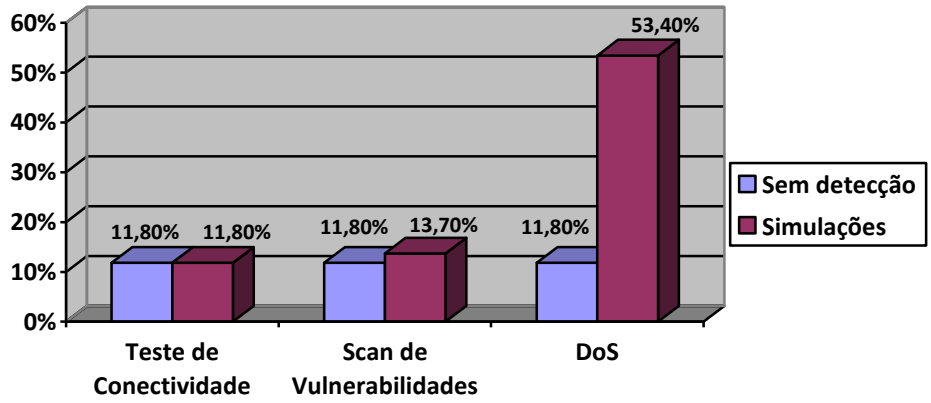
Fonte: do Autor

Foi observado também, que o fluxo de dados na rede, após a inicialização do IDS, teve um incremento em seu valor, oscilando em torno de 0,2 KB/s.

Como pode ser visualizado na Figura 41, durante a detecção do teste de conectividade, não houve alteração no uso da CPU, quando comparado ao sistema estabilizado. Já na detecção do Scan de vulnerabilidades, ocorreu um pequeno aumento, de aproximadamente

1,9%, no uso total da CPU. Diferentemente das duas primeiras simulações, o teste com o DoS, ocasionou um grande aumento no uso da CPU, que após o monitoramento por tempo determinado, chegou a cerca de 53,4%. Portanto, durante este ataque, a diferença de uso foi de 41,6% em relação ao sistema estabilizado.

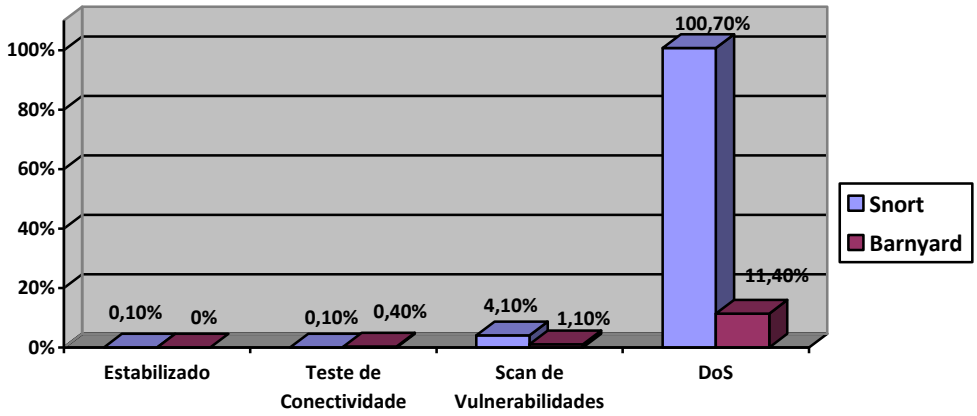
Figura 41 – Taxa aproximada do uso da CPU durante as simulações.



Fonte: do Autor

Levando-se em conta, os dois processos principais que compões o IDS aqui analisado, isto é, o Snort e o Barnyard, a Figura 42 expõe o uso da CPU de forma individualizada pelos dois processos.

Figura 42 – Taxa aproximada do uso da CPU – Snort e Barnyard.



Fonte: do Autor

É possível constatar que o processo do Snort, na simulação do DoS, acarretou em uma sobrecarga de uso da CPU durante o período analisado.

5 CONSIDERAÇÕES FINAIS

A elaboração deste estudo apresentou uma opção já existente, porém, agora de forma atualizada, na área de segurança de redes, através da implementação de um sistema de detecção de intrusão (IDS). Este sistema, composto pelo Snort e BASE, reforça a segurança, no que se refere à detecção de intrusões em uma rede, durante as tentativas de ataques, e facilita a leitura dos *logs* gerados.

No presente trabalho foi realizado um estudo, com grande ganho de conhecimento, a respeito de segurança de redes, tendo como foco o IDS, e as principais ferramentas *open source* existentes. Durante a construção do mesmo, sentiu-se a necessidade de implementar o sistema de forma atualizada, desde o sistema operacional até o interpretador de *logs*, o que exigiu grande esforço, pois houveram inúmeros problemas entre versões, uma vez que o sistema interage com várias ferramentas. Com isso, obteve-se como resultado deste esforço, uma implementação atualizada do referido sistema.

A escolha do Snort para implantação do sistema de detecção de intrusão no ambiente virtual criado revelou-se positiva, pois diante das simulações a que ele foi submetido, se mostrou eficiente, detectando as intrusões e gerando alertas como esperado. A influência que o Snort exerceu no desempenho do servidor, baseado nos resultados obtidos com as simulações aplicadas, pode ser considerada pequena, isso após o término de sua inicialização, pois a variação no uso da CPU foi baixa. Porém, nesta proposta, durante a inicialização do conjunto de ferramentas que compõem o IDS e no decorrer da detecção do ataque de DoS, exigiu uma grande taxa no uso da CPU, bem como na demanda por memória, mostrando que a configuração de hardware deve ser adequada a estas demandas, pois caso contrário, poderá haver comprometimento no desempenho do servidor.

A realização deste trabalho contribuiu de forma significativa na aquisição de conhecimentos na área de segurança de redes. Dessa forma, é possível concluir que, embora o sistema detecte os ataques realizados, somente a implementação do sistema aqui apresentado, não é suficiente para se ter uma total segurança, e sim, que ele deve compor algo maior, com outros sistemas de segurança, vindo assim a somar recursos na dura batalha pela obtenção de segurança em redes.

Na tentativa de aprimorar o sistema de detecção de intrusão, por este trabalho apresentado, é pertinente a continuação deste estudo, na busca por maior eficiência em termos de segurança. Com isso, sugere-se avançar na pesquisa com os seguintes temas:

- Utilização do Snort em conjunto com a ferramenta Guardian, desta forma o sistema atuará também como um IPS. O Guardian é um programa que permite atualizar as regras do firewall, baseado nos alertas gerados pelo Snort, possibilitando assim bloquear as intrusões detectadas;
- Aprofundar a análise de desempenho do sistema aqui apresentado, juntamente com um gerador de tráfego de rede, com isso obtendo um resultado mais próximo da realidade. Um exemplo de gerador de tráfego a ser utilizado é o Ostinato, ferramenta *open source* que cria e envia, automaticamente, pacotes de vários *streams*, com diferentes protocolos e em taxas diferentes.
- Aplicar o sistema IDS proposto no trabalho em um IP real, juntamente com a utilização do protocolo orientado a pacotes *Simple Network Management Protocol* (SNMP). Este protocolo possibilita ao administrador, entre outras coisas, o gerenciamento remoto do que acontece na rede, verificação do desempenho da rede, detectar problemas de rede e acompanhar quem a usa e como é usada.

REFERÊNCIAS

ASSUNÇÃO, Marcos Flávio Araújo. *Honeypots e Honeynets Aprenda a detectar e enganar invasores*. Florianópolis: Editora Visual Books, 2009.

BAKER, Andrew R.; ESLER, Joel; et al. *Snort Intrusion Detection and Prevention Toolkit*. Burlington, MA: Syngress Publishing Inc., 2007. 730p.

BARNYARD2. Disponível em: < http://forensicswiki.org/wiki/Barnyard2> Acesso em: set. 2015

CARVALHO, Luciano Gonçalves de. *Segurança de Redes*. Rio de Janeiro: Editora Ciência Moderna Ltda., 2005.

CENTRO DE ESTUDOS, RESPOSTA E TRATAMENTO DE INCIDENTES DE SEGURANÇA NO BRASIL. *Estatísticas dos Incidentes Reportados ao CERT.br – janeiro a dezembro de 2014*. Disponível em < http://www.cert.br/stats/incidentes/>. Acesso em: out. 2015.

CENTRO DE ESTUDOS, RESPOSTA E TRATAMENTO DE INCIDENTES DE SEGURANÇA NO BRASIL. *Cartilha de Segurança para Internet*. Disponível em <http://cartilha.cert.br/>. Acesso em: 15 jun. 2014.

CHESWICK, William R.; BELLOVIN, Steven M.; RUBIN, Aviel D. *Firewalls e Segurança na Internet Repelindo o hacker ardiloso*. 2. ed. Porto Alegre: Bookman, 2005.

CID, Daniel B.. *Log Analysis using OSSEC*. 2007. Disponível em: < http://www.ossec.net/ files/auscert-2007-dcid.pdf/> Acesso em: 22 ago. 2014.

DOHERTY, Jim; ANDERSON, Neil; MAGGIORA, Paul Della. *Cisco Networking Simplified*. 2. ed. Indianapolis: Cisco Press, 2008.

GUIMARÃES, Alexandre Guedes; LINS, Rafael Lins; OLIVEIRA, Raimundo. *Segurança com Redes Privadas Virtuais VPNs*. Rio de Janeiro: Brasport, 2006.

HLBR. Documentação HLBR. Disponível em: < http://hlbr.sourceforge.net/> Acesso em: 12 jun. 2014

INTERNATIONAL TELECOMMUNICATION UNION. *Recommendation X.800. Security Architecture for Open Systems Interconnection for CCITT Applications*. Geneva, 1991. 46f.

LEBLOND, Eric. *Suricata, to 10 Gbps and beyond.* 2012. Disponível em: < https://home.regit .org/tag/performance/> Acesso em: 10 set. 2014.

MEDEIROS, Thyago dos Santos. Proposta de uma Metodologia para Geração de Dados para Avaliação das Ferramentas de Detecção de Intrusão. 2008. 74f. Trabalho de Conclusão (Curso

de Especialização em Tecnologias, Gerência e Segurança de Redes de Computadores) – UFRGS, Porto Alegre, 2008.

MIGUEL, Daniel Fonseca Mendes. *Implantação do Sistema de Detecção de Intrusão Snort na Segurança de Rede de uma Organização Militar*. Revista Científica Escola de Formação Complementar do Exército. s.d. Disponível em:< http://www.esfcex.ensino.eb.br/revista/producaocientifica/arquivo/534_Artigo.pdf>. Acesso em: 5 jun 2014.

MONTEIRO, Emiliano Soares. *Segurança no Ambiente Corporativo*. Florianópolis: Visual Books Editora, 2003.

MONTORO, Rodrigo. *Introdução ao Snort – Serie Snortando (Parte 1)*. 2012. Disponível em: < http://spookerlabs.blogspot.com.br/2012/01/introducao-ao-snort-serie-snortando.html/> Acesso em: 02 out. 2014.

MORENO, Daniel. *Introdução ao Pentest*. São Paulo: Novatec Editora Ltda., 2015

NAKAMURA, Emílio Tissato; GEUS, Paulo Lício de. *Segurança de Redes em Ambientes Cooperativos*. 2. ed. Novatec, 2007.

NETO, Daniel José da; MAURICIO, Lucas Henrique; COSTA, Victor Pereira da. *SDI Sistemas de Detecção de Intrusão*. Disponível em: <http://www.gta.ufrj.br/grad/11_1/sdi/index.html>. Acesso em: 20 jun. 2014.

NIKTO2. Documentação Nikto2. Disponível em:< https://cirt.net/Nikto2>. Acesso em: nov. 2015.

OSSEC. Documentação OSSEC. Disponível em: <http://ossec-docs.readthedocs.org/en/latest/>. Acesso em: 15 set. 2014.

SAMHAIM. Documentação SAMHAIM. Disponível em: <http://la-samhna.de/samhain/index.html> Acesso em: 18 jun. 2014.

SCHÜRMANN, Tim. O sistema de detecção de intrusão Samhain em guarda. *Linux Magazine* #64, Mar 2010. Disponível em: < https://www.linuxnewmedia.com.br/images/uploads/pdf_aberto/LM_64_39_45_07_capa_samhain.pdf>. Acesso em: set 2014.

SILVA, Crístian Alves; FONSECA, Glauco Alves. *Sistema de Detecção de Intrusos para Redes Locais*. 2007. 91f. Monografia (Pós-Graduação em Segurança de Redes de Computadores) – Faculdade Salesiana, Vitória, 2007.

SILVA, Edelberto Franco e JULIO, Eduardo Pagani. Sistema de Detecção de Intrusão. Revista Infra Magazine 1. Disponível em: < http://www.devmedia.com.br/sistema-de-deteccao-de-intrusao-artigo-revista-infra-magazine-1/20819>. Acesso em Out 2015

SILVA, Gleydson Mazioli da. *Guia Foca GNU/Linux*. 2010. Disponível em: <http://www.ccuec.unicamp.br/biti/download/focalinux2.pdf>. Acesso em: nov. 2015.

SILVA, Lilia de Sá. *Uma Metodologia para Detecção de Ataques no Tráfego de Redes Baseada em Redes Neurais*. 2008. 256f. Tese (Doutorado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2008.

SNORT. Documentação Snort. Disponível em: <https://www.snort.org/docs>. Acesso em: 5 jun. 2014.

STALLINGS, William. *Criptografia e Segurança de Redes – Princípios e Práticas* - 4ª ed. Pearson, 2008.

SURICATA. Documentação Suricata. Disponível em: <https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata_User_Guide>. Acesso em: 10 jun. 2014.

TANENBAUM, Andrew S.. *Redes de Computadores*. 4. ed. Rio de Janeiro: Elsevier Brasil, 2003.

TRUHAN, Nathan Daniel. *Intrusion Detection for 0-Day Vulnerabilities*. 2011. Dissertação (Mestrado) – Kent State University. 2011. Disponível em: < https://etd.ohiolink.edu/! etd. send_file?accession=kent1310682482&disposition=inline>. Acesso em: set 2015.

VACCA, John A. *Network and System Security*. Burlington: Syngress, 2010.

APÊNDICE A – Roteiro de instalação.

Primeiramente foi necessário acrescentar ao arquivo **/etc/apt/sources.list** um repositório que contem versões atualizadas dos pacotes do PHP e MySQL, para isso acrescentou-se as seguintes linhas:

```
deb http://packages.dotdeb.org jessie all
deb-src http://packages.dotdeb.org jessie all
```

Feito isso, foi necessário instalar as chaves GnuPG dotdeb:

```
# cd /usr/src && wget http://www.dotdeb.org/dotdeb.gpg
# cat dotdeb.gpg | apt-key add -
```

Em seguida, foram instalados o MySQL, PHP e Apache. A senha requerida durante a instalação para o root do MySQL, foi a mesma de root do sistema:

```
# apt-get update && apt-get -y install apache2 apache2-doc autoconf automake bison ca-certificates ethtool flex g++ gcc gcc-4.9 libapache2-modphp5 libcrypt-ssleay-perl libmysqlclient-dev libnet1 libnet1-dev libpcre3 libpcre3-dev libphp-adodb libssl-dev libtool libwww-perl make mysqlclient mysql-common mysql-server ntp php5-cli php5-gd php5-mysql php-pear sendmail sendmail-bin sysstat usbmount
```

Foi necessário desabilitar *Large Receive Offload* (LRO) e o *Generic Receive Offload* (GRO) da interface de rede para não haver problemas no reagrupamento dos pacotes. Para isso foi editado o arquivo **/etc/rc.local**, adicionando as seguintes linhas depois de "**exit 0**":

```
ethtool --offload eth0 rx off tx off
ethtool -K eth0 gso off
ethtool -K eth0 gro off
```

A seguir foram instalados os três pré-requisitos para a implementação do Snort:

- libpacp:

```
# cd /usr/src && wget http://www.tcpdump.org/release/libpcap-1.7.4.tar.gz
# tar -zxf libpcap-1.7.4.tar.gz && cd libpcap-1.7.4
# ./configure --prefix=/usr && make && make install
```

- libdnet:

```
# cd /usr/src && wget http://libdnet.googlecode.com/files/libdnet-1.12.tar.gz
# tar -zxf libdnet-1.12.tar.gz && cd libdnet-1.12
# ./configure --prefix=/usr --enable-shared && make && make install
```

- DAQ:

```
# cd /usr/src && wget https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz
# tar -zxf daq-2.0.6.tar.gz && cd daq-2.0.6
# ./configure && make && make install
```

Depois foi necessário atualizar a biblioteca compartilhada:

```
# echo >> /etc/ld.so.conf /usr/lib
# echo >> /etc/ld.so.conf /usr/local/lib && ldconfig
```

Foi instalado e configurado o Snort utilizando os comandos listados a seguir:

```
# cd /usr/src && wget https://www.snort.org/documents/185
# mv 185 snort.conf
# wget https://www.snort.org/downloads/snort/snort-2.9.7.6.tar.gz
# tar -zxf snort-2.9.7.6.tar.gz && cd snort-2.9.7.6
# ./configure --enable-sourcefire && make && make install
# mkdir /usr/local/etc/snort /usr/local/etc/snort/rules /var/log/snort /var/log/barnyard2 /usr/local/lib/snort_dynamicrules
# touch /usr/local/etc/snort/rules/white_list.rules /usr/local/etc/snort/rules/black_list.rules /usr/local/etc/snort/sid-msg.map
# groupadd snort && useradd -g snort snort
# chown snort:snort /var/log/snort /var/log/barnyard2
```

 # cp /usr/src/snort-2.9.7.4/etc/*.conf* /usr/local/etc/snort

 # cp /usr/src/snort-2.9.7.4/etc/*.map /usr/local/etc/snort

 # cp /usr/src/snort.conf /usr/local/etc/snort

Em seguida foi editado o arquivo **/usr/local/etc/snort/snort.conf** (arquivo completo listado no Apêndice B):

Linha #45 - **ipvar HOME_NET 192.168.159.0/24**

Linha #48 - **ipvar EXTERNAL_NET !$HOME_NET**

Linha #104 - **var RULE_PATH /usr/local/etc/snort/rules**

Linha #105 - **var SO_RULE_PATH /usr/local/etc/snort/rules/so_rules**

Linha #106 - **var PREPROC_RULE_PATH /usr/local/etc/snort/rules/preproc_rules**

Linha #109 - **var WHITE_LIST_PATH /usr/local/etc/snort/rules**

Linha #110 - **var BLACK_LIST_PATH /usr/local/etc/snort/rules**

Linha #293 - adicionado no final após **"decompress_depth 65535" max_gzip_mem 104857600**

Linha #521 - adicionado esta linha **output unified2: filename snort.log, limit 128**

Linha #543 - comentado todas as linhas **"include $RULE_PATH"** exceto **"snort.rules"**

Foi instalado e configurado o Barnyard2, seguindo os comandos:

 # cd /usr/src && wget https://github.com/firnsy/barnyard2/archive/master.tar.gz

 # tar -zxf master.tar.gz && cd barnyard2-*

 # autoreconf -fvi -I ./m4 && ./configure --with-mysql --with-mysql-libraries=/usr/lib/x86_64-linux-gnu && make && make install

 # mv /usr/local/etc/barnyard2.conf /usr/local/etc/snort

 # cp schemas/create_mysql /usr/src

No arquivo **/usr/local/etc/snort/barnyard2.conf**, as seguintes linhas foram editadas:

Linha #27 modificado para **/usr/local/etc/snort/reference.config**

Linha #28 modificado para **/usr/local/etc/snort/classification.config**

Linha #29 modificado para **/usr/local/etc/snort/gen-msg.map**

Linha #30 modificado para **/usr/local/etc/snort/sid-msg.map**

Linha #227 modificado para **output alert_fast**

No final do arquivo foi adicionada esta linha:

output database: log, mysql, user=snort password=<senha diferente da senha do root do MySQL> **dbname=snort host=localhost**

Para a criação do banco de dados, foram realizados os seguintes passos:

A senha requerida para acessar o MySQL é a mesma que foi criada durante sua instalação, ou seja, a senha de root do MySQL.

 # mysql -u root -p

mysql> create database snort; mysql> grant CREATE, INSERT, SELECT, DELETE, UPDATE on snort.* to snort@localhost;

A senha que é solicitada a seguir no comando SQL é a mesma que foi definida no arquivo barnyard2.conf.

mysql> SET PASSWORD FOR snort@localhost=PASSWORD('senha'**);**

mysql> use snort;

mysql> source /usr/src/create_mysql

O próximo comando SQL apresenta a lista das novas tabelas recém importadas.

mysql> show tables;

mysql> exit

Para fim de teste, foi inicializado o snort e barnyard com os seguintes comandos:

 # /usr/local/bin/snort -q -u snort -g snort -c /usr/local/etc/snort/snort.conf -i eth0 &

 # /usr/local/bin/barnyard2 -c /usr/local/etc/snort/barnyard2.conf -d /var/log/snort -f snort.log –w /usr/local/etc/snort/bylog.waldo -C /usr/local/etc/snort/ classification.config &

O comando a seguir mostrou que o barnyard estava inserindo eventos corretamente no banco de dados, foi solicitada a senha de root MySQL novamente:

 # mysql -u root -p -D snort -e "select count(*) from event"

Para a configuração do Apache e PHP, primeiramente foi executado o seguinte comando:

 # cp /etc/apache2/sites-available/default-ssl.conf /etc/apache2/sites-enabled

Feito isso, foi necessário editar o arquivo **/etc/php5/apache2/php.ini**:

Foi alterada a linha:

Linha #452 - **error_reporting = E_ALL & ~E_NOTICE**

Logo após os comandos abaixo foram realizados:

 # a2enmod ssl

 # pear config-set preferred_state alpha && pear channel-update pear.php.net

 # pear install --alldeps Image_Color2 Image_Canvas Image_Graph

 # /etc/init.d/apache2 restart

Para instalar o BASE foram seguidos os seguintes passos:

 # cd /usr/src && wget http://sourceforge.net/projects/secureideas/files/latest/download

 # tar -zxf download && cp -r base-1.4.5 /var/www/html/base

Momentaneamente foi dada uma permissão ao diretório **/base**, para que se pudesse configurar o BASE:

 # chmod 777 /var/www/html/base

Para proceder a configuração do BASE, foi acessado no browser o endereço **http://"ip do servidor"/base.**

Feito isso, foi acessada a tela inicial para configuração, e os seguintes passos foram executados:

Foi clicado em **Continue**, e escolhido o idioma **Português**

Informado o caminho para o adodb: **/usr/share/php/adodb**

Clicado em **Continue**

Database Name: **snort**

Database Host: **localhost**

Database Port: **leave blank**

Database User Name: **snort**

Database Password: **(senha)** mesma que foi definida no arquivo barnyard2.conf.

Foram informados os dados para a autenticação do sistema e clicado em **Continue.**

Foi clicado em **Create baseag** que estende o banco de dados para suportar o BASE.

Para terminar a configuração, foi feito o *login* no botão **Step 5**.

Em seguida foi criado um *script* de inicialização para o Snort e Barnyard, criando o arquivo **snortbarn** para adicionar o *script* em **/etc/init.d/snortbarn,** conforme listado no Apêndice C.

Para torná-lo executável e criar os *links* simbólicos de inicialização foi executado os seguintes comandos:

 # chmod +x /etc/init.d/snortbarn

 # insserv -f -v snortbarn

Para iniciar a instalação do pulledpork, foram executados os seguintes comandos:

 # cd /usr/src && wget https://pulledpork.googlecode.com/files/pulledpork-0.7.0.tar.gz

 # tar -zxf pulledpork-0.7.0.tar.gz && cd pulledpork-0.7.0

 # cp pulledpork.pl /usr/local/bin && cp etc/*.conf /usr/local/etc/snort

Depois foi editado o arquivo **/usr/local/etc/snort/pulledpork.conf**:

Linha 19: foi comentada esta linha

Linha 26: foi comentada esta linha

Linha 27: descomentada para usar o conjunto de regras **Emerging Threats**

Linha 131: alterada para: **distro=Debian-8**

Linha 139: comentada esta opção **Blacklist**

Linhas 194-197: descomentadas todas essas linhas

Para desabilitar todas as regras de bloqueio (**fwsam**), foi usado o comando:

 # echo pcre:fwsam >> /usr/local/etc/snort/disablesid.conf

Para rodar o Pulledpork foi usado:

```
# chmod +x /usr/local/bin/pulledpork.pl
# /usr/local/bin/pulledpork.pl -c /usr/local/etc/snort/pulledpork.conf -T –l
```

Para concluir foram feitas as últimas configurações para o correto funcionamento do Snort:

```
# chmod 755 /var/www/base
# pkill snort && pkill barnyard2
# rm -rf /var/log/snort/* /var/log/barnyard2/*
```

Foi editado o arquivo **/usr/local/etc/snort/snort.conf** e na linha 542 foi adicionado:

```
include $RULE_PATH/snort.rules
```

Para inicializar o conjunto de ferramentas que compõe esse IDS, foi executado o seguinte comando:

```
# /etc/init.d/snortbarn restart
```

APÊNDICE B – Arquivo de configuração do Snort.

```
#--------------------------------------------------
#   VRT Rule Packages Snort.conf
#
#   For more information visit us at:
#     http://www.snort.org                Snort Website
#     http://vrt-blog.snort.org/    Sourcefire VRT Blog
#
#     Mailing list Contact:      snort-sigs@lists.sourceforge.net
#     False Positive reports:    fp@sourcefire.com
#     Snort bugs:                bugs@snort.org
#
#     Compatible with Snort Versions:
#     VERSIONS : 2.9.7.5
#
#     Snort build options:
#     OPTIONS : --enable-gre --enable-mpls --enable-targetbased --enable-ppm --enable-perfprofiling --enable-zlib --enable-active-response --enable-normalizer --enable-reload --enable-react --enable-flexresp3
#
#     Additional information:
#     This configuration file enables active response, to run snort in
#     test mode -T you are required to supply an interface -i <interface>
#     or test mode will fail to fully validate the configuration and
#     exit with a FATAL error
#--------------------------------------------------

###################################################
# This file contains a sample snort configuration.
# You should take the following steps to create your own custom configuration:
#
#  1) Set the network variables.
#  2) Configure the decoder
#  3) Configure the base detection engine
#  4) Configure dynamic loaded libraries
#  5) Configure preprocessors
#  6) Configure output plugins
#  7) Customize your rule set
#  8) Customize preprocessor and decoder rule set
#  9) Customize shared object rule set
###################################################

###################################################
# Step #1: Set the network variables.  For more information, see README.variables
###################################################

# Setup the network addresses you are protecting
ipvar HOME_NET 192.168.159.0/24

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET !$HOME_NET

# List of DNS servers on your network
ipvar DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
ipvar SMTP_SERVERS $HOME_NET

# List of web servers on your network
ipvar HTTP_SERVERS $HOME_NET

# List of sql servers on your network
ipvar SQL_SERVERS $HOME_NET

# List of telnet servers on your network
ipvar TELNET_SERVERS $HOME_NET

# List of ssh servers on your network
ipvar SSH_SERVERS $HOME_NET

# List of ftp servers on your network
ipvar FTP_SERVERS $HOME_NET
```

```
# List of sip servers on your network
ipvar SIP_SERVERS $HOME_NET

# List of ports you run web servers on
portvar HTTP_PORTS
[36,80,81,82,83,84,85,86,87,88,89,90,311,383,555,591,593,631,801,808,818,901,972,1158,1220,1414,1533,1741,1830,1942,2231,2301,238
1,2578,2809,2980,3029,3037,3057,3128,3443,3702,4000,4343,4848,5000,5117,5250,5600,5814,6080,6173,6988,7000,7001,7005,7071,714
4,7145,7510,7770,7777,7778,7779,8000,8001,8008,8014,8015,8020,8028,8040,8080,8081,8082,8085,8088,8090,8118,8123,8180,8181,818
2,8222,8243,8280,8300,8333,8344,8400,8443,8500,8509,8787,8800,8888,8899,8983,9000,9002,9060,9080,9090,9091,9111,9290,9443,944
7,9710,9788,9999,10000,11371,12601,13014,15489,19980,29991,33300,34412,34443,34444,40007,41080,44449,50000,50002,51423,5333
1,55252,55555,56712]

# List of ports you want to look for SHELLCODE on.
portvar SHELLCODE_PORTS !80

# List of ports you might see oracle attacks on
portvar ORACLE_PORTS 1024:

# List of ports you want to look for SSH connections on:
portvar SSH_PORTS 22

# List of ports you run ftp servers on
portvar FTP_PORTS [21,2100,3535]

# List of ports you run SIP servers on
portvar SIP_PORTS [5060,5061,5600]

# List of file data ports for file inspection
portvar FILE_DATA_PORTS [$HTTP_PORTS,110,143]

# List of GTP ports for GTP preprocessor
portvar GTP_PORTS [2123,2152,3386]

# other variables, these should not be modified
ipvar AIM_SERVERS
[64.12.24.0/23,64.12.28.0/23,64.12.161.0/24,64.12.163.0/24,64.12.200.0/24,205.188.3.0/24,205.188.5.0/24,205.188.7.0/24,205.188.9.0/24,2
05.188.153.0/24,205.188.179.0/24,205.188.248.0/24]

# Path to your rules files (this can be a relative path)
# Note for Windows users:  You are advised to make this an absolute path,
# such as:  c:\snort\rules
var RULE_PATH /usr/local/etc/snort/rules
var SO_RULE_PATH /usr/local/etc/snort/rules/so_rules
var PREPROC_RULE_PATH /usr/local/etc/snort/rules/preproc_rules

# If you are using reputation preprocessor set these
var WHITE_LIST_PATH /usr/local/etc/snort/rules
var BLACK_LIST_PATH /usr/local/etc/snort/rules

##################################################
# Step #2: Configure the decoder.  For more information, see README.decode
##################################################

# Stop generic decode events:
config disable_decode_alerts

# Stop Alerts on experimental TCP options
config disable_tcpopt_experimental_alerts

# Stop Alerts on obsolete TCP options
config disable_tcpopt_obsolete_alerts

# Stop Alerts on T/TCP alerts
config disable_tcpopt_ttcp_alerts

# Stop Alerts on all other TCPOption type events:
config disable_tcpopt_alerts

# Stop Alerts on invalid ip options
config disable_ipopt_alerts

# Alert if value in length field (IP, TCP, UDP) is greater th elength of the packet
# config enable_decode_oversized_alerts

# Same as above, but drop packet if in Inline mode (requires enable_decode_oversized_alerts)
```

```
# config enable_decode_oversized_drops

# Configure IP / TCP checksum mode
config checksum_mode: all

# Configure maximum number of flowbit references.  For more information, see README.flowbits
# config flowbits_size: 64

# Configure ports to ignore
# config ignore_ports: tcp 21 6667:6671 1356
# config ignore_ports: udp 1:17 53

# Configure active response for non inline operation. For more information, see REAMDE.active
# config response: eth0 attempts 2

# Configure DAQ related options for inline operation. For more information, see README.daq
#
# config daq: <type>
# config daq_dir: <dir>
# config daq_mode: <mode>
# config daq_var: <var>
#
# <type> ::= pcap | afpacket | dump | nfq | ipq | ipfw
# <mode> ::= read-file | passive | inline
# <var> ::= arbitrary <name>=<value passed to DAQ
# <dir> ::= path as to where to look for DAQ module so's

# Configure specific UID and GID to run snort as after dropping privs. For more information see snort -h command line options
#
# config set_gid:
# config set_uid:

# Configure default snaplen. Snort defaults to MTU of in use interface. For more information see README
#
# config snaplen:
#

# Configure default bpf_file to use for filtering what traffic reaches snort. For more information see snort -h command line options (-F)
#
# config bpf_file:
#

# Configure default log directory for snort to log to.  For more information see snort -h command line options (-l)
#
# config logdir:


##################################################
# Step #3: Configure the base detection engine.  For more information, see  README.decode
##################################################

# Configure PCRE match limitations
config pcre_match_limit: 3500
config pcre_match_limit_recursion: 1500

# Configure the detection engine  See the Snort Manual, Configuring Snort - Includes - Config
config detection: search-method ac-split search-optimize max-pattern-len 20

# Configure the event queue.  For more information, see README.event_queue
config event_queue: max_queue 8 log 5 order_events content_length

##################################################
## Configure GTP if it is to be used.
## For more information, see README.GTP
##################################################

# config enable_gtp

##################################################
# Per packet and rule latency enforcement
# For more information see README.ppm
##################################################

# Per Packet latency configuration
#config ppm: max-pkt-time 250, \
#   fastpath-expensive-packets, \
```

```
#   pkt-log

# Per Rule latency configuration
#config ppm: max-rule-time 200, \
#   threshold 3, \
#   suspend-expensive-rules, \
#   suspend-timeout 20, \
#   rule-log alert

################################################
# Configure Perf Profiling for debugging
# For more information see README.PerfProfiling
################################################

#config profile_rules: print all, sort avg_ticks
#config profile_preprocs: print all, sort avg_ticks

################################################
# Configure protocol aware flushing
# For more information see README.stream5
################################################
config paf_max: 16000

################################################
# Step #4: Configure dynamic loaded libraries.
# For more information, see Snort Manual, Configuring Snort - Dynamic Modules
################################################

# path to dynamic preprocessor libraries
dynamicpreprocessor directory /usr/local/lib/snort_dynamicpreprocessor/

# path to base preprocessor engine
dynamicengine /usr/local/lib/snort_dynamicengine/libsf_engine.so

# path to dynamic rules libraries
dynamicdetection directory /usr/local/lib/snort_dynamicrules

################################################
# Step #5: Configure preprocessors
# For more information, see the Snort Manual, Configuring Snort - Preprocessors
################################################

# GTP Control Channle Preprocessor. For more information, see README.GTP
# preprocessor gtp: ports { 2123 3386 2152 }

# Inline packet normalization. For more information, see README.normalize
# Does nothing in IDS mode
preprocessor normalize_ip4
preprocessor normalize_tcp: block, rsv, pad, urp, req_urg, req_pay, req_urp, ips, ecn stream
preprocessor normalize_icmp4
preprocessor normalize_ip6
preprocessor normalize_icmp6

# Target-based IP defragmentation.  For more inforation, see README.frag3
preprocessor frag3_global: max_frags 65536
preprocessor frag3_engine: policy windows detect_anomalies overlap_limit 10 min_fragment_length 100 timeout 180

# Target-Based stateful inspection/stream reassembly.  For more inforation, see README.stream5
preprocessor stream5_global: track_tcp yes, \
   track_udp yes, \
   track_icmp no, \
   max_tcp 262144, \
   max_udp 131072, \
   max_active_responses 2, \
   min_response_seconds 5
preprocessor stream5_tcp: policy windows, detect_anomalies, require_3whs 180, \
   overlap_limit 10, small_segments 3 bytes 150, timeout 180, \
    ports client 21 22 23 25 42 53 70 79 109 110 111 113 119 135 136 137 139 143 \
        161 445 513 514 587 593 691 1433 1521 1741 2100 3306 6070 6665 6666 6667 6668 6669 \
        7000 8181 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779, \
    ports both 36 80 81 82 83 84 85 86 87 88 89 90 110 311 383 443 465 563 555 591 593 631 636 801 808 818 901 972 989 992 993 994
995 1158 1220 1414 1533 1741 1830 1942 2231 2301 2381 2578 2809 2980 3029 3037 3057 3128 3443 3702 4000 4343 4848 5000 5117
5250 5600 5814 6080 6173 6988 7907 7000 7001 7005 7071 7144 7145 7510 7802 7770 7777 7778 7779 \
        7801 7900 7901 7902 7903 7904 7905 7906 7908 7909 7910 7911 7912 7913 7914 7915 7916 \
        7917 7918 7919 7920 8000 8001 8008 8014 8015 8020 8028 8040 8080 8081 8082 8085 8088 8090 8118 8123 8180 8181 8182 8222
8243 8280 8300 8333 8344 8400 8443 8500 8509 8787 8800 8888 8899 8983 9000 9002 9060 9080 9090 9091 9111 9290 9443 9447 9710
```

```
9788 9999 10000 11371 12601 13014 15489 19980 29991 33300 34412 34443 34444 40007 41080 44449 50000 50002 51423 53331 55252
55555 56712
preprocessor stream5_udp: timeout 180

# performance statistics.  For more information, see the Snort Manual, Configuring Snort - Preprocessors - Performance Monitor
# preprocessor perfmonitor: time 300 file /var/snort/snort.stats pktcnt 10000

# HTTP normalization and anomaly detection.  For more information, see README.http_inspect
preprocessor http_inspect: global iis_unicode_map unicode.map 1252 compress_depth 65535 decompress_depth 65535 max_gzip_mem
104857600
preprocessor http_inspect_server: server default \
    http_methods { GET POST PUT SEARCH MKCOL COPY MOVE LOCK UNLOCK NOTIFY POLL BCOPY BDELETE BMOVE
LINK UNLINK OPTIONS HEAD DELETE TRACE TRACK CONNECT SOURCE SUBSCRIBE UNSUBSCRIBE PROPFIND
PROPPATCH BPROPFIND BPROPPATCH RPC_CONNECT PROXY_SUCCESS BITS_POST CCM_POST SMS_POST
RPC_IN_DATA RPC_OUT_DATA RPC_ECHO_DATA } \
    chunk_length 500000 \
    server_flow_depth 0 \
    client_flow_depth 0 \
    post_depth 65495 \
    oversize_dir_length 500 \
    max_header_length 750 \
    max_headers 100 \
    max_spaces 200 \
    small_chunk_length { 10 5 } \
    ports { 36 80 81 82 83 84 85 86 87 88 89 90 311 383 555 591 593 631 801 808 818 901 972 1158 1220 1414 1533 1741 1830 1942 2231
2301 2381 2578 2809 2980 3029 3037 3057 3128 3443 3702 4000 4343 4848 5000 5117 5250 5600 5814 6080 6173 6988 7000 7001 7005
7071 7144 7145 7510 7770 7777 7778 7779 8000 8001 8008 8014 8015 8020 8028 8040 8080 8081 8082 8085 8088 8090 8118 8123 8180
8181 8182 8222 8243 8280 8300 8333 8344 8400 8443 8500 8509 8787 8800 8888 8899 8983 9000 9002 9060 9080 9090 9091 9111 9290
9443 9447 9710 9788 9999 10000 11371 12601 13014 15489 19980 29991 33300 34412 34443 34444 40007 41080 44449 50000 50002
51423 53331 55252 55555 56712 } \
    non_rfc_char { 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 } \
    enable_cookie \
    extended_response_inspection \
    inspect_gzip \
    normalize_utf \
    unlimited_decompress \
    normalize_javascript \
    apache_whitespace no \
    ascii no \
    bare_byte no \
    directory no \
    double_decode no \
    iis_backslash no \
    iis_delimiter no \
    iis_unicode no \
    multi_slash no \
    utf_8 no \
    u_encode yes \
    webroot no

# ONC-RPC normalization and anomaly detection.  For more information, see the Snort Manual, Configuring Snort - Preprocessors - RPC
Decode
preprocessor rpc_decode: 111 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779 no_alert_multiple_requests
no_alert_large_fragments no_alert_incomplete

# Back Orifice detection.
preprocessor bo

# FTP / Telnet normalization and anomaly detection.  For more information, see README.ftptelnet
preprocessor ftp_telnet: global inspection_type stateful encrypted_traffic no check_encrypted
preprocessor ftp_telnet_protocol: telnet \
    ayt_attack_thresh 20 \
    normalize ports { 23 } \
    detect_anomalies
preprocessor ftp_telnet_protocol: ftp server default \
    def_max_param_len 100 \
    ports { 21 2100 3535 } \
    telnet_cmds yes \
    ignore_telnet_erase_cmds yes \
    ftp_cmds { ABOR ACCT ADAT ALLO APPE AUTH CCC CDUP } \
    ftp_cmds { CEL CLNT CMD CONF CWD DELE ENC EPRT } \
    ftp_cmds { EPSV ESTA ESTP FEAT HELP LANG LIST LPRT } \
    ftp_cmds { LPSV MACB MAIL MDTM MIC MKD MLSD MLST } \
    ftp_cmds { MODE NLST NOOP OPTS PASS PASV PBSZ PORT } \
    ftp_cmds { PROT PWD QUIT REIN REST RETR RMD RNFR } \
    ftp_cmds { RNTO SDUP SITE SIZE SMNT STAT STOR STOU } \
```

```
    ftp_cmds { STRU SYST TEST TYPE USER XCUP XCRC XCWD } \
    ftp_cmds { XMAS XMD5 XMKD XPWD XRCP XRMD XRSQ XSEM } \
    ftp_cmds { XSEN XSHA1 XSHA256 } \
    alt_max_param_len 0 { ABOR CCC CDUP ESTA FEAT LPSV NOOP PASV PWD QUIT REIN STOU SYST XCUP XPWD } \
    alt_max_param_len 200 { ALLO APPE CMD HELP NLST RETR RNFR STOR STOU XMKD } \
    alt_max_param_len 256 { CWD RNTO } \
    alt_max_param_len 400 { PORT } \
    alt_max_param_len 512 { SIZE } \
    chk_str_fmt { ACCT ADAT ALLO APPE AUTH CEL CLNT CMD } \
    chk_str_fmt { CONF CWD DELE ENC EPRT EPSV ESTP HELP } \
    chk_str_fmt { LANG LIST LPRT MACB MAIL MDTM MIC MKD } \
    chk_str_fmt { MLSD MLST MODE NLST OPTS PASS PBSZ PORT } \
    chk_str_fmt { PROT REST RETR RMD RNFR RNTO SDUP SITE } \
    chk_str_fmt { SIZE SMNT STAT STOR STRU TEST TYPE USER } \
    chk_str_fmt { XCRC XCWD XMAS XMD5 XMKD XRCP XRMD XRSQ } \
    chk_str_fmt { XSEM XSEN XSHA1 XSHA256 } \
    cmd_validity ALLO < int [ char R int ] > \
    cmd_validity EPSV < [ { char 12 | char A char L char L } ] > \
    cmd_validity MACB < string > \
    cmd_validity MDTM < [ date nnnnnnnnnnnnnn[.n[n[n]]] ] string > \
    cmd_validity MODE < char ASBCZ > \
    cmd_validity PORT < host_port > \
    cmd_validity PROT < char CSEP > \
    cmd_validity STRU < char FRPO [ string ] > \
    cmd_validity TYPE < { char AE [ char NTC ] | char I | char L [ number ] } >
preprocessor ftp_telnet_protocol: ftp client default \
    max_resp_len 256 \
    bounce yes \
    ignore_telnet_erase_cmds yes \
    telnet_cmds yes


# SMTP normalization and anomaly detection.  For more information, see README.SMTP
preprocessor smtp: ports { 25 465 587 691 } \
    inspection_type stateful \
    b64_decode_depth 0 \
    qp_decode_depth 0 \
    bitenc_decode_depth 0 \
    uu_decode_depth 0 \
    log_mailfrom \
    log_rcptto \
    log_filename \
    log_email_hdrs \
    normalize cmds \
    normalize_cmds { ATRN AUTH BDAT CHUNKING DATA DEBUG EHLO EMAL ESAM ESND ESOM ETRN EVFY } \
    normalize_cmds { EXPN HELO HELP IDENT MAIL NOOP ONEX QUEU QUIT RCPT RSET SAML SEND SOML } \
    normalize_cmds { STARTTLS TICK TIME TURN TURNME VERB VRFY X-ADAT X-DRCP X-ERCP X-EXCH50 } \
    normalize_cmds { X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA XTRN XUSR } \
    max_command_line_len 512 \
    max_header_line_len 1000 \
    max_response_line_len 512 \
    alt_max_command_line_len 260 { MAIL } \
    alt_max_command_line_len 300 { RCPT } \
    alt_max_command_line_len 500 { HELP HELO ETRN EHLO } \
    alt_max_command_line_len 255 { EXPN VRFY ATRN SIZE BDAT DEBUG EMAL ESAM ESND ESOM EVFY IDENT NOOP RSET
} \
    alt_max_command_line_len 246 { SEND SAML SOML AUTH TURN ETRN DATA RSET QUIT ONEX QUEU STARTTLS TICK
TIME TURNME VERB X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA XTRN XUSR } \
    valid_cmds { ATRN AUTH BDAT CHUNKING DATA DEBUG EHLO EMAL ESAM ESND ESOM ETRN EVFY } \
    valid_cmds { EXPN HELO HELP IDENT MAIL NOOP ONEX QUEU QUIT RCPT RSET SAML SEND SOML } \
    valid_cmds { STARTTLS TICK TIME TURN TURNME VERB VRFY X-ADAT X-DRCP X-ERCP X-EXCH50 } \
    valid_cmds { X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA XTRN XUSR } \
    xlink2state { enabled }

# Portscan detection.  For more information, see README.sfportscan
# preprocessor sfportscan: proto  { all } memcap { 10000000 } sense_level { low }

# ARP spoof detection.  For more information, see the Snort Manual - Configuring Snort - Preprocessors - ARP Spoof Preprocessor
# preprocessor arpspoof
# preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00

# SSH anomaly detection.  For more information, see README.ssh
preprocessor ssh: server_ports { 22 } \
                autodetect \
                max_client_bytes 19600 \
                max_encrypted_packets 20 \
```

```
            max_server_version_len 100 \
            enable_respoverflow enable_ssh1crc32 \
            enable_srvoverflow enable_protomismatch

# SMB / DCE-RPC normalization and anomaly detection.  For more information, see README.dcerpc2
preprocessor dcerpc2: memcap 102400, events [co ]
preprocessor dcerpc2_server: default, policy WinXP, \
    detect [smb [139,445], tcp 135, udp 135, rpc-over-http-server 593], \
    autodetect [tcp 1025:, udp 1025:, rpc-over-http-server 1025:], \
    smb_max_chain 3, smb_invalid_shares ["C$", "D$", "ADMIN$"]

# DNS anomaly detection.  For more information, see README.dns
preprocessor dns: ports { 53 } enable_rdata_overflow

# SSL anomaly detection and traffic bypass.  For more information, see README.ssl
preprocessor ssl: ports { 443 465 563 636 989 992 993 994 995 5061 7801 7802 7900 7901 7902 7903 7904 7905 7906 7907 7908 7909
7910 7911 7912 7913 7914 7915 7916 7917 7918 7919 7920 }, trustservers, noinspect_encrypted

# SDF sensitive data preprocessor.  For more information see README.sensitive_data
preprocessor sensitive_data: alert_threshold 25

# SIP Session Initiation Protocol preprocessor.  For more information see README.sip
preprocessor sip: max_sessions 40000, \
  ports { 5060 5061 5600 }, \
  methods { invite \
            cancel \
            ack \
            bye \
            register \
            options \
            refer \
            subscribe \
            update \
            join \
            info \
            message \
            notify \
            benotify \
            do \
            qauth \
            sprack \
            publish \
            service \
            unsubscribe \
            prack }, \
  max_uri_len 512, \
  max_call_id_len 80, \
  max_requestName_len 20, \
  max_from_len 256, \
  max_to_len 256, \
  max_via_len 1024, \
  max_contact_len 512, \
  max_content_len 2048

# IMAP preprocessor.  For more information see README.imap
preprocessor imap: \
  ports { 143 } \
  b64_decode_depth 0 \
  qp_decode_depth 0 \
  bitenc_decode_depth 0 \
  uu_decode_depth 0

# POP preprocessor. For more information see README.pop
preprocessor pop: \
  ports { 110 } \
  b64_decode_depth 0 \
  qp_decode_depth 0 \
  bitenc_decode_depth 0 \
  uu_decode_depth 0

# Modbus preprocessor. For more information see README.modbus
preprocessor modbus: ports { 502 }

# DNP3 preprocessor. For more information see README.dnp3
preprocessor dnp3: ports { 20000 } \
  memcap 262144 \
```

```
  check_crc

# Reputation preprocessor. For more information see README.reputation
preprocessor reputation: \
  memcap 500, \
  priority whitelist, \
  nested_ip inner, \
  whitelist $WHITE_LIST_PATH/white_list.rules, \
  blacklist $BLACK_LIST_PATH/black_list.rules

##################################################
# Step #6: Configure output plugins
# For more information, see Snort Manual, Configuring Snort - Output Modules
##################################################

# unified2
# Recommended for most installs
# output unified2: filename merged.log, limit 128, nostamp, mpls_event_types, vlan_event_types

# Additional configuration for specific types of installs
# output alert_unified2: filename snort.alert, limit 128, nostamp
# output log_unified2: filename snort.log, limit 128, nostamp

output unified2: filename snort.log, limit 128

# syslog
# output alert_syslog: LOG_AUTH LOG_ALERT

# pcap
# output log_tcpdump: tcpdump.log

# metadata reference data.  do not modify these lines
include classification.config
include reference.config


##################################################
# Step #7: Customize your rule set
# For more information, see Snort Manual, Writing Snort Rules
#
# NOTE: All categories are enabled in this conf file
##################################################

# site specific rules
include $RULE_PATH/local.rules

#include $RULE_PATH/app-detect.rules
#include $RULE_PATH/attack-responses.rules
#include $RULE_PATH/backdoor.rules
#include $RULE_PATH/bad-traffic.rules
#include $RULE_PATH/blacklist.rules
#include $RULE_PATH/botnet-cnc.rules
#include $RULE_PATH/browser-chrome.rules
#include $RULE_PATH/browser-firefox.rules
#include $RULE_PATH/browser-ie.rules
#include $RULE_PATH/browser-other.rules
#include $RULE_PATH/browser-plugins.rules
#include $RULE_PATH/browser-webkit.rules
#include $RULE_PATH/chat.rules
#include $RULE_PATH/content-replace.rules
#include $RULE_PATH/ddos.rules
#include $RULE_PATH/dns.rules
#include $RULE_PATH/dos.rules
#include $RULE_PATH/experimental.rules
#include $RULE_PATH/exploit-kit.rules
#include $RULE_PATH/exploit.rules
#include $RULE_PATH/file-executable.rules
#include $RULE_PATH/file-flash.rules
#include $RULE_PATH/file-identify.rules
#include $RULE_PATH/file-image.rules
#include $RULE_PATH/file-java.rules
#include $RULE_PATH/file-multimedia.rules
#include $RULE_PATH/file-office.rules
#include $RULE_PATH/file-other.rules
#include $RULE_PATH/file-pdf.rules
#include $RULE_PATH/finger.rules
```

```
#include $RULE_PATH/ftp.rules
#include $RULE_PATH/icmp-info.rules
#include $RULE_PATH/icmp.rules
#include $RULE_PATH/imap.rules
#include $RULE_PATH/indicator-compromise.rules
#include $RULE_PATH/indicator-obfuscation.rules
#include $RULE_PATH/indicator-scan.rules
#include $RULE_PATH/indicator-shellcode.rules
#include $RULE_PATH/info.rules
#include $RULE_PATH/malware-backdoor.rules
#include $RULE_PATH/malware-cnc.rules
#include $RULE_PATH/malware-other.rules
#include $RULE_PATH/malware-tools.rules
#include $RULE_PATH/misc.rules
#include $RULE_PATH/multimedia.rules
#include $RULE_PATH/mysql.rules
#include $RULE_PATH/netbios.rules
#include $RULE_PATH/nntp.rules
#include $RULE_PATH/oracle.rules
#include $RULE_PATH/os-linux.rules
#include $RULE_PATH/os-mobile.rules
#include $RULE_PATH/os-other.rules
#include $RULE_PATH/os-solaris.rules
#include $RULE_PATH/os-windows.rules
#include $RULE_PATH/other-ids.rules
#include $RULE_PATH/p2p.rules
#include $RULE_PATH/phishing-spam.rules
#include $RULE_PATH/policy-multimedia.rules
#include $RULE_PATH/policy-other.rules
#include $RULE_PATH/policy.rules
#include $RULE_PATH/policy-social.rules
#include $RULE_PATH/policy-spam.rules
#include $RULE_PATH/pop2.rules
#include $RULE_PATH/pop3.rules
#include $RULE_PATH/protocol-dns.rules
#include $RULE_PATH/protocol-finger.rules
#include $RULE_PATH/protocol-ftp.rules
#include $RULE_PATH/protocol-icmp.rules
#include $RULE_PATH/protocol-imap.rules
#include $RULE_PATH/protocol-nntp.rules
#include $RULE_PATH/protocol-other.rules
#include $RULE_PATH/protocol-pop.rules
#include $RULE_PATH/protocol-rpc.rules
#include $RULE_PATH/protocol-scada.rules
#include $RULE_PATH/protocol-services.rules
#include $RULE_PATH/protocol-snmp.rules
#include $RULE_PATH/protocol-telnet.rules
#include $RULE_PATH/protocol-tftp.rules
#include $RULE_PATH/protocol-voip.rules
#include $RULE_PATH/pua-adware.rules
#include $RULE_PATH/pua-other.rules
#include $RULE_PATH/pua-p2p.rules
#include $RULE_PATH/pua-toolbars.rules
#include $RULE_PATH/rpc.rules
#include $RULE_PATH/rservices.rules
#include $RULE_PATH/scada.rules
#include $RULE_PATH/scan.rules
#include $RULE_PATH/server-apache.rules
#include $RULE_PATH/server-iis.rules
#include $RULE_PATH/server-mail.rules
#include $RULE_PATH/server-mssql.rules
#include $RULE_PATH/server-mysql.rules
#include $RULE_PATH/server-oracle.rules
#include $RULE_PATH/server-other.rules
#include $RULE_PATH/server-samba.rules
#include $RULE_PATH/server-webapp.rules
#include $RULE_PATH/shellcode.rules
#include $RULE_PATH/smtp.rules
#include $RULE_PATH/snmp.rules
#include $RULE_PATH/specific-threats.rules
#include $RULE_PATH/spyware-put.rules
#include $RULE_PATH/sql.rules
#include $RULE_PATH/telnet.rules
#include $RULE_PATH/tftp.rules
#include $RULE_PATH/virus.rules
#include $RULE_PATH/voip.rules
```

```
#include $RULE_PATH/web-activex.rules
#include $RULE_PATH/web-attacks.rules
#include $RULE_PATH/web-cgi.rules
#include $RULE_PATH/web-client.rules
#include $RULE_PATH/web-coldfusion.rules
#include $RULE_PATH/web-frontpage.rules
#include $RULE_PATH/web-iis.rules
#include $RULE_PATH/web-misc.rules
#include $RULE_PATH/web-php.rules
#include $RULE_PATH/x11.rules

##################################################
# Step #8: Customize your preprocessor and decoder alerts
# For more information, see README.decoder_preproc_rules
##################################################

# decoder and preprocessor event rules
# include $PREPROC_RULE_PATH/preprocessor.rules
# include $PREPROC_RULE_PATH/decoder.rules
# include $PREPROC_RULE_PATH/sensitive-data.rules

##################################################
# Step #9: Customize your Shared Object Snort Rules
# For more information, see http://vrt-blog.snort.org/2009/01/using-vrt-certified-shared-object-rules.html
##################################################

# dynamic library rules
# include $SO_RULE_PATH/browser-ie.rules
# include $SO_RULE_PATH/browser-other.rules
# include $SO_RULE_PATH/exploit-kit.rules
# include $SO_RULE_PATH/file-flash.rules
# include $SO_RULE_PATH/file-image.rules
# include $SO_RULE_PATH/file-java.rules
# include $SO_RULE_PATH/file-multimedia.rules
# include $SO_RULE_PATH/file-office.rules
# include $SO_RULE_PATH/file-other.rules
# include $SO_RULE_PATH/file-pdf.rules
# include $SO_RULE_PATH/indicator-shellcode.rules
# include $SO_RULE_PATH/malware-cnc.rules
# include $SO_RULE_PATH/malware-other.rules
# include $SO_RULE_PATH/netbios.rules
# include $SO_RULE_PATH/os-linux.rules
# include $SO_RULE_PATH/os-other.rules
# include $SO_RULE_PATH/os-windows.rules
# include $SO_RULE_PATH/policy-social.rules
# include $SO_RULE_PATH/protocol-dns.rules
# include $SO_RULE_PATH/protocol-nntp.rules
# include $SO_RULE_PATH/protocol-other.rules
# include $SO_RULE_PATH/protocol-snmp.rules
# include $SO_RULE_PATH/protocol-voip.rules
# include $SO_RULE_PATH/pua-p2p.rules
# include $SO_RULE_PATH/server-apache.rules
# include $SO_RULE_PATH/server-iis.rules
# include $SO_RULE_PATH/server-mail.rules
# include $SO_RULE_PATH/server-mysql.rules
# include $SO_RULE_PATH/server-oracle.rules
# include $SO_RULE_PATH/server-other.rules
# include $SO_RULE_PATH/server-webapp.rules

# legacy dynamic library rule files
# include $SO_RULE_PATH/bad-traffic.rules
# include $SO_RULE_PATH/browser-ie.rules
# include $SO_RULE_PATH/chat.rules
# include $SO_RULE_PATH/dos.rules
# include $SO_RULE_PATH/exploit.rules
# include $SO_RULE_PATH/file-flash.rules
# include $SO_RULE_PATH/icmp.rules
# include $SO_RULE_PATH/imap.rules
# include $SO_RULE_PATH/misc.rules
# include $SO_RULE_PATH/multimedia.rules
# include $SO_RULE_PATH/netbios.rules
# include $SO_RULE_PATH/nntp.rules
# include $SO_RULE_PATH/p2p.rules
# include $SO_RULE_PATH/smtp.rules
# include $SO_RULE_PATH/snmp.rules
# include $SO_RULE_PATH/specific-threats.rules
```

```
# include $SO_RULE_PATH/web-activex.rules
# include $SO_RULE_PATH/web-client.rules
# include $SO_RULE_PATH/web-iis.rules
# include $SO_RULE_PATH/web-misc.rules

# Event thresholding or suppression commands. See threshold.conf
include threshold.conf
```

APÊNDICE C – Script de inicialização do Snort e Barnyard.

```
#! /bin/sh
#
### BEGIN INIT INFO
# Provides: snortbarn
# Required-Start: $remote_fs $syslog mysql
# Required-Stop: $remote_fs $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# X-Interactive: true
# Short-Description: Start Snort and Barnyard
### END INIT INFO

. /lib/init/vars.sh
. /lib/lsb/init-functions

mysqld_get_param() {
        /usr/sbin/mysqld --print-defaults | tr " " "\n" | grep -- "--$1" | tail -n 1 | cut -d= -f2
}

do_start()
{
        log_daemon_msg "Starting Snort and Barnyard" ""
        # Make sure mysql has finished starting
        ps_alive=0
        while [ $ps_alive -lt 1 ];
        do
        pidfile=`mysqld_get_param pid-file`
        if [ -f "$pidfile" ] && ps `cat $pidfile` >/dev/null 2>&1; then ps_alive=1; fi
        sleep 1
        done
        /sbin/ifconfig eth1 up
        /usr/local/bin/snort -q -u snort -g snort -c /usr/local/etc/snort/snort.conf -i eth1 &
         /usr/local/bin/barnyard2 -q -c /usr/local/etc/snort/barnyard2.conf -d /var/log/snort -f snort.log -w /usr/local/etc/snort/bylog.waldo -
        C /usr/local/etc/snort/classification.
        config 2> /dev/null &
        log_end_msg 0
        return 0
}

do_stop()
{
        log_daemon_msg "Stopping Snort and Barnyard" ""
        kill $(pidof snort) 2> /dev/null
        kill $(pidof barnyard2) 2> /dev/null
        log_end_msg 0
        return 0
}
```

```
case "$1" in
start)
        do_start
        ;;
stop)
        do_stop
        ;;
restart)
        do_stop
        do_start
        ;;
*)
        echo "Usage: snort-barn {start|stop|restart}" >&2
        exit 3
        ;;
esac
exit 0
```