MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO TÉCNICO INDUSTRIAL – CTISM / UFSM
CURSO SUPERIOR DE TECNOLOGIA EM REDES DE COMPUTADORES

IMPLEMENTAÇÃO DE LOGS E GERAÇÃO DE GRÁFICOS PARA GESTÃO DE INFORMAÇÕES DO FIREWALL DE APLICAÇÃO UniscanWAF

TRABALHO DE CONCLUSÃO DE CURSO

Bruno Gonçalves Lucena

Santa Maria, RS, Brasil
2013

CTISM / UFSM, RS	GONÇALVES LUCENA, Bruno	Graduado	2013

IMPLEMENTAÇÃO DE LOGS E GERAÇÃO DE GRÁFICOS PARA GESTÃO DE INFORMAÇÕES DO FIREWALL DE APLICAÇÃO UniscanWAF

Bruno Gonçalves Lucena

Trabalho de Conclusão de Curso (TCC) apresentado ao Curso de Graduação em Tecnologia em Redes de Computadores, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Tecnólogo em Redes de Computadores.

Orientador: Prof. Me. Rogério Correa Turchetti

Santa Maria, RS, Brasil
2013

Ministério da Educação
Universidade Federal de Santa Maria
Colégio Técnico Industrial – CTISM/UFSM
Curso Superior de Tecnologia em Redes de Computadores

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Conclusão de Curso.

Implementação de logs e geração de gráficos para gestão do firewall de aplicação UniscanWAF.

elaborado pelo acadêmico
Bruno Gonçalves Lucena

como requisito para
Conclusão de Curso de Graduação.

COMISSÃO EXAMINADORA:

Rogério Correa Turchetti, Me.
(Presidente/Orientador)

Eugênio de Oliveira Simonetto, Dr. (UFSM)

Tiago Antônio Rizzetti, Me. (UFSM)

Santa Maria, 25 de Janeiro de 2013

RESUMO

Trabalho de Conclusão de Curso
Curso Superior de Tecnologia em Redes de Computadores
Universidade Federal de Santa Maria

IMPLEMENTAÇÃO DE LOGS E GERAÇÃO DE GRÁFICOS PARA GESTÃO DO FIREWALL DE APLICAÇÃO UniscanWAF

AUTOR: LUCENA, G. B.

ORIENTADOR: ROGÉRIO CORREA TURCHETTI

Data e Local do Projeto: Santa Maria, 25 de Janeiro de 2013.

Este trabalho tem como objetivo apresentar um estudo sobre a implementação de registros de *logs* e a criação de gráficos para gerenciamento do *firewall* de aplicação UniscanWAF de forma a tratar toda a saída resultante do monitoramento da ferramenta, auxiliando o gerente de rede na tomada de decisão. Sendo assim, esta pesquisa é centrada na área de tratamento de dados brutos extraídos de *logs*, possibilitando visualizar de forma clara os dados do UniscanWAF. Para tanto, são gerados gráficos, através da biblioteca RRDtool. Desta forma, o presente estudo pretende auxiliar o gerente de segurança da informação na tomada de decisões, através do tratamento das informações geradas pela ferramenta UniscanWAF. Facilitando, assim, a visualização e interpretação de *logs* de segurança com o auxílio de bibliotecas que permitem a geração de gráficos para gerenciamento de segurança. Sendo possível também, a análise de gerenciamento de contabilidade em geral.

Palavras-chave: *Logs*; RRDtool; Gerenciamento de Informações; UniscanWAF.

ABSTRACT

Completion of Course Work
Superior Course of Technology in Computer Network
Federal University of Santa Maria

DEPLOYMENT OF LOGS AND GRAPHS FOR MANAGEMENT OF APPLICATION FIREWALL UniscanWAF

AUTHOR: LUCENA, G. B.

ADVISER: ROGÉRIO CORREA TURCHETTI

Date and Place Project: Santa Maria, January 25th, 2013.

This work aims to present a study about the development of logs and graphs for management of the application firewall UniscanWAF, in order to treat all the resulting output from monitoring of tool, helping the network manager in decision making. Thus, this research is focused on the area of treatment of raw data extracted from logs, allowing a clear view of the data UniscanWAF. In this way, graphics are generated by RRDtool library. Thus, this study intends to help the information security manager in making decisions, through the processing of information generated by the tool UniscanWAF. Thereby facilitating the visualization and interpretation of security logs with the aid of libraries that allow you to generate graphs to security management. Also being possible, analysis of accounting management in general.

Keywords: Logs; RRDtool; Management of Informations; UniscanWAF.

AGRADECIMENTOS

A elaboração deste trabalho tornou-se possível, graças ao Professor Rogério Correa Turchetti, agradeço-lhe pelas boas orientações. Ao colega e amigo Alfredo Del Fabro Neto, desenvolvedor do UniscanWAF, que disponibilizou a ferramenta, bem como um ótimo suporte, auxiliando para o desenvolvimento deste estudo.

Uma pessoa que mostrou-se interessada em auxiliar, contribuindo na reta final do presente trabalho foi o grande Professor Walter Priesnitz Filho, neste momento agradeço-lhe pela dedicação prestada.

Agradeço pelo apoio familiar sempre presente, meus pais e irmãos, em todas minhas decisões, e pela compreensão de não poder estar presente em confraternizações.

Agradeço à minha amada Raissa Monego, a qual é uma pessoa que eu amo tanto. Aos meus também colegas e amigos Anderson Petry e Rafael Boufleuer, pela força e apoio sempre presente, nos momentos necessários. Agradeço enfim, a todos que colaboraram de alguma forma para a realização deste trabalho.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ASP	- *Active Server Pages*
CSV	- *Comma-separated values*
CRLF	- *Carraige Line Feed*
GPL	- *General Public License*
HTML	- *Hypertext Markup Language*
HTTP	- *HyperText Transfer Protocol*
HTTPS	- *HyperText Transfer Protocol Secure*
IDS	- *Intrusion Detection Systems*
IP	- *Internet Protocol*
ISO	- *International Organization for Standardization*
JSP	- *JavaServer Pages*
LDAP	- *Lightweight Directory Access Protocol*
LFI	- *Local File Include*
MIB	- *Base Information Management*
NMS	- *Network Management System*
PHP	- *Hypertext Prepocessor*
RCE	- *Remote Command Execution*
RFC	- *Request for Comments*
RFI	- *Remote File Include*
SDLC	- *Software Development Life Cycle*
SQL	- *Structured Query Language*
SMI	- *Structure of Management Information*
SNMP	- *Simple Network Management Protocol*
UDP	- *User Datagram Protocol*
WAF	- *Web Application Firewall*
WIMU	- *Web Interface Management* UniscanWAF
XML	- *Extensible Markup Language*
XSS	- *Cross Site Scripting*

SUMÁRIO

1. INTRODUÇÃO

A segurança da informação sempre foi um tópico que necessitou cuidados por parte das pessoas em praticamente todos os lugares, principalmente em grandes organizações. Com a expansão do fluxo de informações trocadas em uma rede, a preocupação em manter todo este volume de dados seguro é um bom motivo para alertar a comunidade dos administradores de rede. Dentre outros fatores, com a evolução das tecnologias, exige-se um maior grau à implementações de níveis de segurança para acompanhar tais avanços tecnológicos. Surgiu assim, segundo as exigências de segurança, a ferramenta principal utilizada neste trabalho, o UniscanWAF (KUROSE; ROSS, 2010).

Não é uma tarefa simples definir o número de informações que uma aplicação *Web* pode tratar diariamente, a escalabilidade é um fator importante para conseguir oferecer um serviço adequado mesmo nos horários com maior número de acessos. Conforme cresce o número de acessos aumenta-se o número de informações recebidas pela aplicação *Web*, tendo a necessidade de haver um tratamento adequado destes dados para possíveis ações ativas ou, melhor ainda, pró-ativas.

A acessibilidade é um parâmetro cada vez mais comum no ramo da informática. Sendo assim, devemos tomar cuidado com a imensa gama de informações que nos cercam, pois muitas destas, são geradas com intenções maliciosas e objetivam atacar uma possível vítima. Portanto, aspectos de segurança devem ter especial atenção por parte de equipes de desenvolvimento e administradores de rede. Dentre os principais fatores para preocupação com segurança estão, que novas vulnerabilidades surgem diariamente dando origem à novos ataques e, também que, a defesa é mais complexa do que o ataque, pois para o *hacker*, basta que ele consiga explorar apenas um ponto de falha em um alvo. Por outro lado, a defesa é muito mais complexa, pois exige que todos os pontos sejam defendidos (NAKAMURA; GEUS, 2007).

O objetivo deste trabalho é auxiliar o gerente de segurança da informação na tomada de decisões, através do tratamento das informações geradas pela

ferramenta UniscanWAF. Isto é, facilitar a visualização e interpretação de *logs* de segurança com o auxílio de bibliotecas que permitem a geração de gráficos para gerenciamento de segurança. Sendo possível também, a análise de gerenciamento de contabilidade em geral. Desta forma, serão desenvolvidos *logs* e gráficos para gestão das informações do *firewall* de aplicação UniscanWAF de forma a tratar toda a saída resultante do monitoramento da ferramenta, auxiliando o gerente de rede na tomada de decisão.

A organização deste trabalho está segmentada em capítulos. No capítulo dois é feita a revisão bibliográfica, contendo uma abordagem sobre conceitos importantes, isto é, o gerenciamento em rede de computadores, o *firewall* de aplicação *web* (WAF), bem como trabalhos relacionados. Enfim, neste capítulo serão centrados, os alicerces utilizados para elaboração deste estudo. Na sequência, no capítulo três, é destinado à arquitetura proposta para este trabalho, contendo objetivos da proposta, arquitetura da aplicação de gerenciamento e ferramentas utilizadas. Por conseguinte, o quarto capítulo, é designado às contribuições geradas, como por exemplo, a implementação dos registros de *logs* e a geração de gráficos para o gerenciamento das informações do UniscanWAF. No capítulo cinco, propõem-se os trabalhos futuros. Os capítulos seguintes, seis e sete, trazem os resultados obtidos e as considerações finais, respectivamente. E por fim, o capítulo oito, é destinado as referências bibliográficas do presente estudo.

2. REVISÃO BIBLIOGRÁFICA

Neste capítulo serão abordados os principais alicerces para o desenvolvimento do presente trabalho, focando-se portanto em ferramentas que auxiliam de alguma forma o administrador a gerenciar seus ativos de redes e consequentemente tomar decisões. Além do exposto, é objetivo deste capítulo apresentar e estudar técnicas de gerenciamento existentes nas aplicações estudadas. Esta análise possibilitará a implementação das funcionalidades de gerenciamento para o presente trabalho.

Sendo primeiramente apresentados os conceitos importantes para elaboração deste trabalho e logo em seguida a apresentação das ferramentas comparativas, a subdivisão desdes tópicos será disposta em seções, onde os mesmos serão detalhados.

2.1. Gerenciamento em Redes de Computadores

A gerência de redes pode ser entendida como todas as atividades, métodos, procedimentos, e ferramentas que permitem a operação (para manter a rede ativa e funcionando sem problemas), manutenção (capacidade de efetuar reparos, melhorias, prevenir e agir de forma proativa), administração (forma de manter o controle dos recursos na rede e como eles são atribuídos) e disponibilidade (configuração de recursos para determinados tipos de serviços) de sistemas de redes (CLEMM, 2006).

Um sistema de gerenciamento em redes de computadores, segundo Stallings (1999), é um conjunto de ferramentas que permitem administração, monitoramento, auditoria, configuração, segurança, contabilidade, desempenho, controle. Para tanto, é de se imaginar que esta é uma tarefa amplamente complexa. Sendo assim, basicamente, o gerenciamento de redes significa o conjunto de ações e procedimentos necessários para manter uma rede sempre funcionando, de

preferência à contento.

Para melhor clareza, a ISO (*International Organization for Standardization*), dividiu a grande área de gerenciamento de redes em cinco subáreas: 1) Gerenciamento de Desempenho, 2) Gerenciamento de Configuração; 3) Gerenciamento de Falhas; 4) Gerenciamento de Segurança e 5) Gerenciamento de Contabilidade. Estas duas últimas serão as abordadas na implementação de técnicas para o gerenciamento do *firewall* de aplicação *web* UniscanWAF, o qual será descrito no capítulo 3. As seções seguintes demonstrarão as cinco subáreas anteriormente citadas, o protocolo SNMP (Simple Network Management Protocol), bem como ferramentas de *firewalls* de aplicações *web*, com suas respectivas considerações preliminares.

2.1.1. Gerenciamento de Desempenho

Segundo diversos autores, dentre eles ((DANTAS, 2002) e (KUROSE; ROSS, 2010)), o objetivo da gerência de desempenho é analisar, administrar, medir, informar, quantificar e controlar a performance de diversos componentes na rede, como por exemplo picos de utilização e vazão em um determinado terminal. Entre os dispositivos que englobam o gerenciamento de desempenho estão roteadores, enlaces, servidores, dentre outros nodos da rede. Para isto, o protocolo SNMP tem papel fundamental neste tipo de gerenciamento na *web*.

2.1.2. Gerenciamento de Configuração

O gerenciamento de configuração permite que um gerente de rede saiba quais dispositivos fazem parte da rede administrada e quais são suas configurações de *software* e de *hardware.* Desta maneira, esta subárea de gerenciamento é responsável pelas configurações física e lógica dos componentes de uma rede

(KUROSE; ROSS, 2010). Sendo, a física o tratamento da localização física dos nodos e da configuração de *hardware* da rede e, a lógica correspondendo à nomeação, parâmetros de portas e serviços, bem como a definição de protocolos à utilizar.

Segundo Freitas e Monteiro (2004), faz parte do escopo do gerenciamento de configuração manter uma descrição minuciosa atualizada para gerar relatórios, auxiliando em mudanças de configurações, quando necessárias, bem como, promovendo facilidade de acesso à documentos de configuração dos equipamentos. Sucintamente, permite ainda controlar os objetos gerenciados, identificá-los, coletar e disponibilizar dados sobre estes.

2.1.3. Gerenciamento de Falhas

A meta do gerenciamento de falhas é detectar anomalias, registrar condições de funcionamento inadequado de um dispositivo e, por fim, tratar e solucionar estes eventos em uma rede. Distinguir gerenciamento de falhas, de gerenciamento de desempenho é uma tarefa que exige um pouco de atenção. Pode-se considerar a gerência de falhas como o tratamento e correção imediata do problema, como por exemplo, interrupção de serviços em enlaces, em hospedeiros, em *hardwares* e/ou *softwares,* bem como, em roteadores. Nota-se, desta maneira, que estes são geralmente problemas críticos para o funcionamento adequado da rede. Já a gerência de desempenho adota uma abordagem de mais longo prazo ao tratamento dos erros, ou seja, não é necessária uma intervenção imediata do administrador da rede (KUROSE; ROSS, 2010).

A gerência de falhas tem como responsabilidade instituir decisões que devem ser tomadas para restaurar elementos do sistema que possam ter apresentado problema. Compete à ela monitorar os estados dos recursos da rede e dar manutenção a cada um dos objetos gerenciados. Estas informações, juntamente com um mapa da rede, podem identificar se algum elemento está com defeito, ou não está funcionando. Caso haja detecção de defeitos, o administrador da rede deve

solucionar o problema de imediato, evitando o desencadeamento de maiores danos (FREITAS; MONTEIRO, 2004).

2.1.4. Gerenciamento de Segurança

A gerência de segurança é, conforme (KUROSE; ROSS, 2010), um conceito em crescente evolução nos últimos anos, tornando-se um fator indispensável às organizações de grande porte. O gerenciamento de segurança tem como objetivo utilizar-se de técnicas para controle e monitoramento de mecanismos para segurança. Dentre os métodos para se implementar a gerência de segurança estão: a) Controlar e monitorar o acesso aos recursos de rede de acordo com alguma política definida; b) Gerar, distribuir, e armazenar chaves criptográficas; c) Prover mecanismos para proteção dos recursos da rede e informações de usuário; d) Verificar se os recursos de segurança da rede estão disponíveis somente para usuários autorizados e, e) Coletar, armazenar, e examinar os registros de auditoria e *logs* de segurança. Este último aspecto, bastante relevante para elaboração deste trabalho.

Pode-se exemplificar a implementação de técnicas de segurança em várias situações com diversos ambientes. Dentre os exemplos estão: a proteção de sistemas e arquivos com senhas, promovendo mecanismos de controle de acesso aos sistemas computacionais; as centrais de distribuição de chaves e as autoridades certificadoras; controle de informações sigilosas que trafegam nos circuitos de dados, implementando codificação através de criptografia; o uso de *firewalls* para monitorar e controlar pontos extremos de acesso à rede; criação de *logs* e registros para análise e monitoramento de aplicações.

Não confundir a gerência de segurança com seus próprios mecanismos de segurança, na prática, não é algo simples, isto é, conhecer a diferença entre estas duas coisas é muito difícil (NETO, 2009). De forma simplificada, pode-se dizer que a gerência de segurança usa técnicas de autenticação e políticas de autorização, auditoria e manutenção de *logs* de segurança, controle de chaves criptográficas, enfim, todos os tipos de mecanismos para serviços de segurança, visando garantir a

disponibilidade, confiabilidade e integridade de sistemas (PINHEIRO, 2002).

2.1.5. Gerenciamento de Contabilidade

Atribui-se à gerência de contabilidade a aferição do uso da rede, estabelecimento de métricas de utilização da rede para melhor distribuição dos seus recursos, determinação de custos, checagem e verificação de quotas (PINHEIRO, 2002). A partir destas informações, o administrador da rede pode responsabilizar os usuários, por exemplo, em função do uso indevido da rede. Estas informações ainda permitem determinar uma provável necessidade de expansão da rede, ou seja, se a utilização dos recursos de rede estão aumentando além de determinados limites, pode-se indicar a necessidade de expansão ou reconfiguração da rede. Três operações básicas podem ser feitas para ajudar no desempenho da rede: 1) Coletar dados da rede; 2) Analisar os dados coletados e 3) Contabilizar por usuários, estabelecer métricas, verificar quotas e determinar custos.

Segundo Kurose e Ross (2010), define-se o gerenciamento de contabilidade, como aquele que permite ao gerente da rede especificar, registrar e controlar o acesso de usuários e dispositivos aos recursos da rede. Tendo por funcionalidade determinar quotas de utilização, cobranças por utilização e alocação de acesso privilegiado a recursos.

2.1.6. Protocolo de Gerenciamento SNMP

Gerenciar redes de computadores, para muitos profissionais na área de TI, é uma tarefa complexa, pois, além de ser uma atividade que necessita extremo cuidado, vem-se tornando uma função cada vez mais essencial. O SNMP é um protocolo para gerenciamento de rede simples, ou seja, possibilita a gerência com simplicidade. Este é um protocolo da camada de aplicação criado para transportar

informações que possibilitem o gerenciamento da rede. É através deste que ferramentas de gerenciamento (NMS) comunicam-se com os dispositivos gerenciados. Este possibilita que administradores de rede gerenciem o desempenho de uma rede, monitorem equipamentos, interfaces, processadores, memórias de dispositivos, como por exemplo comutadores de pacotes, servidores e computadores (MAURO; SCHMIDT, 2005).

Com este protocolo, administradores de redes conseguem visualizar o *status* atual da rede, manter um histórico de atividades, bem como receber avisos de forma imediata para ajudar na resolução de problemas.

Conforme ((MAURO; SCHMIDT, 2005) e (KUROSE; ROSS, 2010)), a topologia de uma rede gerenciada por meio de SNMP inclui três elementos sendo eles: **a)** Dispositivos Gerenciados (elementos da rede que serão gerenciados e que possuem suporte ao protocolo SNMP, como roteadores e *switches*); **b)** Agentes: Módulos de *software* que armazenam informações dos dispositivos gerenciados em uma base de dados (MIB). Estes agentes também podem armazenar informações como processamento, uso de memória, temperatura do dispositivo, quantidade de mensagens de erro, número de *bytes* de pacotes recebidos e enviados e **c)** Gerente da Rede (NMS): Sistema responsável pelo monitoramento e controle dos dispositivos gerenciados. Permite que os administradores de redes visualizem as informações de leitura SNMP, seja por meio de gráfico, tabelas, relatórios ou alertas por e-mail. Como exemplo de gerente de rede pode-se citar o Cacti, o MRTG e o CiscoWork.

Segundo Stallings (1999), o SNMP utiliza o protocolo UDP da camada de transporte, para realizar a comunicação entre gerentes e dispositivos gerenciados. Existem três tipos de mensagens comuns durante o processo de gerenciamento: a) *Get* – Permite que a estação de gerenciamento recupere o valor de objetos da MIB (*Base Information Management*) do agente; b) *Set* – Permite que a estação de gerenciamento defina o valor de objetos da MIB do agente e c) *Trap* – Permite que o agente notifique a estação de gerenciamento sobre eventos significativos.

Sucintamente, o SNMP é um protocolo que permite ao gerente verificar a performance da rede, detectar e solucionar problemas, e fazer planejamentos de escalabilidade. Conforme Kurose e Ross (2010), o protocolo SNMP é utilizado para

transportar informações da MIB entre entidades gerenciadoras e agentes. Ainda segundo os autores, a forma mais comum de uso do SNMP é o modo “comando-resposta”, onde o gerente envia uma requisição à um agente, que realiza alguma ação e envia uma resposta. Outro método comum de utilização do SNMP é para enviar uma mensagem TRAP (mensagem não solicitada) à entidade gerenciadora, notificando de uma situação excepcional que resultou em mudança nos valores dos objetos na MIB. As mensagens SNMP são codificadas através da SMI que, por sua vez, utiliza um subconjunto da ASN.1 (STALLINGS ,1999). O SNMP é padronizado pela RFC 1157, que posteriormente teve atualizações para SNMPv2, e SNMPv3, esta última versão focada para questões de segurança, pois a centralização na segurança resultou em um uso mais adequado do protocolo, como uma ferramenta não só para monitoramento, mas também para controle de redes.

2.1.7. *Logs* e Auditoria

A importância de se ter um arquivo de rastreabilidade de eventos ocorridos em um sistema, ou neste caso em determinada aplicação (servidor *web*), é algo essencial para se ter um serviço executando perfeitamente (livre de possíveis erros). Encontrar um problema, aparentemente em redes saturadas (como ocorre geralmente com as que possuem servidores *web*), sem a análise minuciosa de um registro contendo todos os acontecimentos, torna-se um processo fatigante. É neste ponto, que entram os *logs*, os quais são basicamente são conjuntos destes registros de acontecimentos e/ou eventos de um serviço ou aplicação.

Em um sistema operacional os registros de eventos ficam armazenados no *Syslog*. Este é um mecanismo utilizado pelos sistemas operacionais, processos e aplicativos que permite o envio de mensagens de atividades e erros para uma estação de gerenciamento. Conforme descrito na RFC 5424, existem 8 (oito) níveis de *log* que variam de 0 (zero) à 7 (sete). Onde quanto menor o nível, maior o impacto do evento registrado. A Tabela 1 demonstra a importância das mensagens, os *levels* (níveis) seguem ordem decrescente de importância. Portanto, o nível de

gravidade 0 (zero) é o evento mais importante e/ou pior caso, sendo por conseguinte, o 7 (sete) o menos crítico.

LEVEL **(Nível)**	***MESSAGE*** **(Mensagem)**	**DESCRIÇÃO** **(Breve Especificação da Gravidade)**
0	LOG_EMERG	Emergência: o sistema está inutilizável
1	LOG_ALERT	Alerta: uma ação deve ser tomada imediatamente
2	LOG_CRIT	Crítico: condições críticas
3	LOG_ERR	Erros: condições de erros
4	LOG_WARNING	Avisos: condições de advertências
5	LOG_NOTICE	Notificações: condição normal, mas significativa
6	LOG_INFO	Informações: mensagem informativa
7	LOG_DEBUG	Depuração: mensagem de nível de depuração

Tabela 1: Classificação de níveis de gravidade do *Syslog*.
Fonte: Adaptação da RFC-5424.

Logs são muito importantes. Sua análise pode ser uma ferramenta bastante útil para se fazer o gerenciamento e auditoria de uma determinada aplicação, como exemplo, um servidor *Web*. Os *logs* podem ser gravados em diversos lugares simultaneamente. O registro pode ocorrer tanto no próprio equipamento que hospeda a aplicação *web*, quanto fora da máquina servidora, sendo este último método uma ferramenta utilizada em larga escala, por ser mais confiável, pois em um invasão bem sucedida, geralmente o primeiro passo do atacante é fazer a destruição dos *logs* do sistema, porém se estes dados estiverem em outro lugar (em um outro computador protegido) evita-se este tipo de problema.

Conforme os autores Cheswick, Bellovin e Rubin (2005), é comprovado que invasores usufruem, muitas vezes, destes recursos de registros de eventos desleixados, por administradores, como mecanismos prelúdios para um ataque. Desta forma, é uma boa prática manter os arquivos de *log* fora do servidor que disponibiliza o serviço, por questões de segurança. Vem daí a necessidade de possuir um cuidado especial à estes importantes recursos, pois, dentre outros mecanismos, este é uma das munições que fazem parte do arsenal dos *hackers*.

Enquanto que o *syslog* é um ótimo recurso para se trabalhar com registros, ou seja, é um programa útil para coletar *logs* em um sistema, o Netstat[1] por outro lado está entre as melhores ferramentas de auditoria (CHESWICK; BELLOVIN; RUBIN, 2005). Quando quer se testar a aplicabilidade destes recursos em um ambiente hostil, ou seja, em aplicações *web* na Internet.

2.2. WAF: *Firewall* de Aplicação *Web*

A disseminação das redes de computadores é um parâmetro considerável à segurança de informações. Hoje, com o acúmulo de usuários e tráfego de dados formou-se o que conhecemos por *Internet,* com proporções alarmantes de serviços e aplicações migrando para a *Web*. Nestas condições, diversas aplicações optam por usufruírem dos variados recursos que este ambiente oferece, como portabilidade, acessibilidade e compartilhamento. No entanto, por tratar-se de um grande volume de dados, todos os serviços que trafegam informações na *Web* devem saber que a segurança neste ambiente está sujeita à diversos tipos de ataques provindos de vulnerabilidades nela existentes.

Ciente de que ataques sempre irão existir, uma forma para minimizar esse problema são os testes de segurança durante o desenvolvimento do *software*. Todavia, segundo (FABRO NETO; TURCHETTI, 2012), nem todas vulnerabilidades são previstas no SDLC (*Software Development Life Cycle*) de uma aplicação, visto que podem ser descobertas novas vulnerabilidades após a disponibilização da aplicação ao usuário final. Pensando nisso, foi concebida a ideia de implementar um WAF (*Web Application Firewall*), para monitorar em tempo real os acessos à aplicação e determinar quais requisições estão aptas a serem enviadas para o servidor *Web* onde está a aplicação. Portanto, só são enviados ao servidor *Web*, requisições que, a priori, não representem risco ao sistema. Deste modo, a ideia é agregar um outro nível de segurança ao sistema, protegendo-o de ataques *Web*, e eventualmente, de novos ataques não previstos durante o desenvolvimento e

1 **Netstat** (*Network Statistics*): é uma ferramenta de 'linha de comando' que exibe informações sobre conexões de rede (entrada e saída), tabelas de roteamento, estatísticas numéricas e interfaces de rede. Tem suporte tanto para plataformas Unix quanto Windows (NETSTAT, 2012).

implantação do sistema.

Um WAF é, como define a organização (WAFEC, 2006), uma nova tecnologia de segurança que tem o papel de proteger aplicações *web* de ataques. As soluções de um WAF são capazes de prevenir e neutralizar um ataque a uma aplicação, conseguindo filtrar de forma eficiente o que, geralmente, um IDS (*Intrusion Detection Systems*) e *firewalls* que atuam na camada de rede não conseguem. Isso se deve ao fato de um WAF agir diretamente na camada de aplicação, filtrando os dados e principalmente parâmetros utilizados em uma transação HTTP.

Este trabalho será desenvolvido, a partir da utilização de um WAF, o qual será o cerne para esta pesquisa. Esta ferramenta é o UniscanWAF, que do ponto de vista de um administrador de rede, é carente de um sistema que possibilite fácil gerenciamento. Entre os requisitos estão, a visualização e interpretação de *logs* de segurança de maneira estruturada para possibilitar um raciocínio analítico dos dados. Atualmente, o UniscanWAF tem a capacidade de detectar vulnerabilidades *Web*. Entretanto, há uma necessidade de se tratar adequadamente as informações capturadas pela ferramenta, uma vez que, são gerados *logs* que impossibilitam a análise detalhada dos dados capturados. Informações detalhadas sobre a ferramenta UniscanWAF serão descritas mais adiante, no capítulo 3.

2.2.1. Trabalhos Relacionados

Com a finalidade de fazer um levantamento do estado das funcionalidades de gerenciamento oferecidas pelos detectores de vulnerabilidades, e ferramentas que auxiliam no tratamento de grandes volumes de registros em *logs* (fazendo auditoria e gerenciamento de determinadas bases de dados em *logs)*, efetuou-se uma pesquisa, onde foram analisadas descrições e funcionalidades de *softwares,* com características similares. Puderam ser selecionados os principais programas relacionados ao gênero deste trabalho, isto é, ferramentas que trabalham com vulnerabilidades *web,* bem como aplicações voltadas à gerência e tratamento de bases de dados em *logs*.

Desta maneira, a escolha das ferramentas teve por objetivo a visualização, análise e verificação das técnicas e/ou métodos de gerenciamento em redes implantadas, bem como a auditoria e consulta aos armazenamentos em *logs* gerados pelas ferramentas instrumentadas nesta pesquisa. Sendo assim, os *scanners* de vulnerabilidades *web,* juntamente com as aplicações que oferecem serviços para gerenciamento de *logs,* serão comparados e avaliados nos aspectos relevantes para a realização deste estudo.

Existe no mercado uma ampla variedade de ferramentas voltadas à segurança de aplicações *web*, no entanto o tratamento e gerência dos *logs* gerados nestas aplicações são considerados um ponto fraco destas ferramentas, que muitas vezes mostram-se aparentemente eficientes (HAKING, 2012). A princípio, foram feitos testes de funcionalidades, para checagem de aspectos relevantes para geração de novas estratégias para implementação de técnicas para gerenciamento de uma aplicação, tornando-se assim, ferramentas realmente interessantes, colaborando com o desenvolvimento para uma interface de gerenciamento do UniscanWAF, denominada WIMU (*Web Interface Management UniscanWAF*), a qual será vista no capítulo 7, em trabalhos futuros. A partir deste levantamento foram selecionadas as seguintes ferramentas: **1)** Acunetix Web Vulnerability Scanner Version 8.0 Free Edition, **2)** Nikto, **3)** ModSecurity, **4)** Wapiti - Web Application Vulnerability Scanner / Security Auditor, **5)** Loggly, e, **6)** Logstash. Sendo que estas duas últimas ferramentas são próprias para tratamento e gerência de bases de dados de registros em *log.*

2.2.1.1. Acunetix

Segundo o *site* do proprietário do *software* (ACUNETIX, 2012), o Acunetix é um *scanner* de vulnerabilidades *web*, *que* detecta de modo automatizado vulnerabilidades XSS, SQL Injection, verificador de arquivos e diretórios, e outras em aplicações *web*. São algumas características do Acunetix: a) Analisador automático de Java *script,* permitindo testes de segurança de Ajax e em aplicações

Web 2.0; b) Possui sistema de testes de *cross-site scripting* (XSS) e SQL-injection; c) É um *scanner multi-threads*; d) Contém um rastreador que detecta o tipo de servidor *web* e linguagem da aplicação; e) Rastreia e analisa *sites*, incluindo conteúdo em Flash, SOAP e AJAX; f) Não é *open source,* sendo um *software* proprietário, porém possui versão *free*[2], e funciona nas plataformas Microsoft Windows {2000, 2003, Server, Server 2008, XP, Vista e Seven}.

Esta ferramenta possui uma interface gráfica bastante interessante, com um bom método de navegação sobre ela. Levando-se em consideração aspectos de segurança, o gerenciamento das informações coletadas pelo Acunetix contém: a) O detalhamento de vulnerabilidades encontradas, como o tipo e local (diretório e/ou arquivo) onde se encontra na aplicação *web* testada; b) A classificação da vulnerabilidade em três níveis de ameaça (alto, médio e baixo); c) Exibe a estrutura do *site* alvo, bem como, os resultados encontrados na busca com seus respectivos *status*; d) Mostra informações da aplicação *web* testada, como por exemplo, sistema operacional (UNIX Ubuntu), nome e versão do servidor *web* (Apache/2.2.22), tecnologias utilizada (PHP) e suscetibilidade da aplicação (sim/não); e) Estatísticas da aplicação, como tempo de verificação e número de requisições efetuadas no *site* alvo; f) Exibe uma janela de atividades, dividida em *logs* da aplicação e registros de erros. (ACUNETIX, 2012).

Sendo um *software* proprietário, foi utilizado o Acunetix Web Vulnerability Scanner v8.0 Free Edition (versão gratuita da ferramenta). Foi citado anteriormente que a ferramenta continha uma interface gráfica amigável, segue exemplo na Figura 1, uma demostração da interface da ferramenta Acunetix em um ambiente real de testes, na distribuição Microsoft Windows XP SP3.

2 A versão *free* possui as seguintes limitações: **a)** Os *sites* serão verificados apenas para vulnerabilidades XSS (apenas nos *sites* de teste do Acunetix são verificados todos os tipos de vulnerabilidades); **b)** O relatório padrão que é gerado pela ferramenta pode ser apenas visualizado, não sendo permitindo imprimi-lo ou exportá-lo; **c)** Os resultados do *scanner* não podem ser salvos (ACUNETIX, 2012).

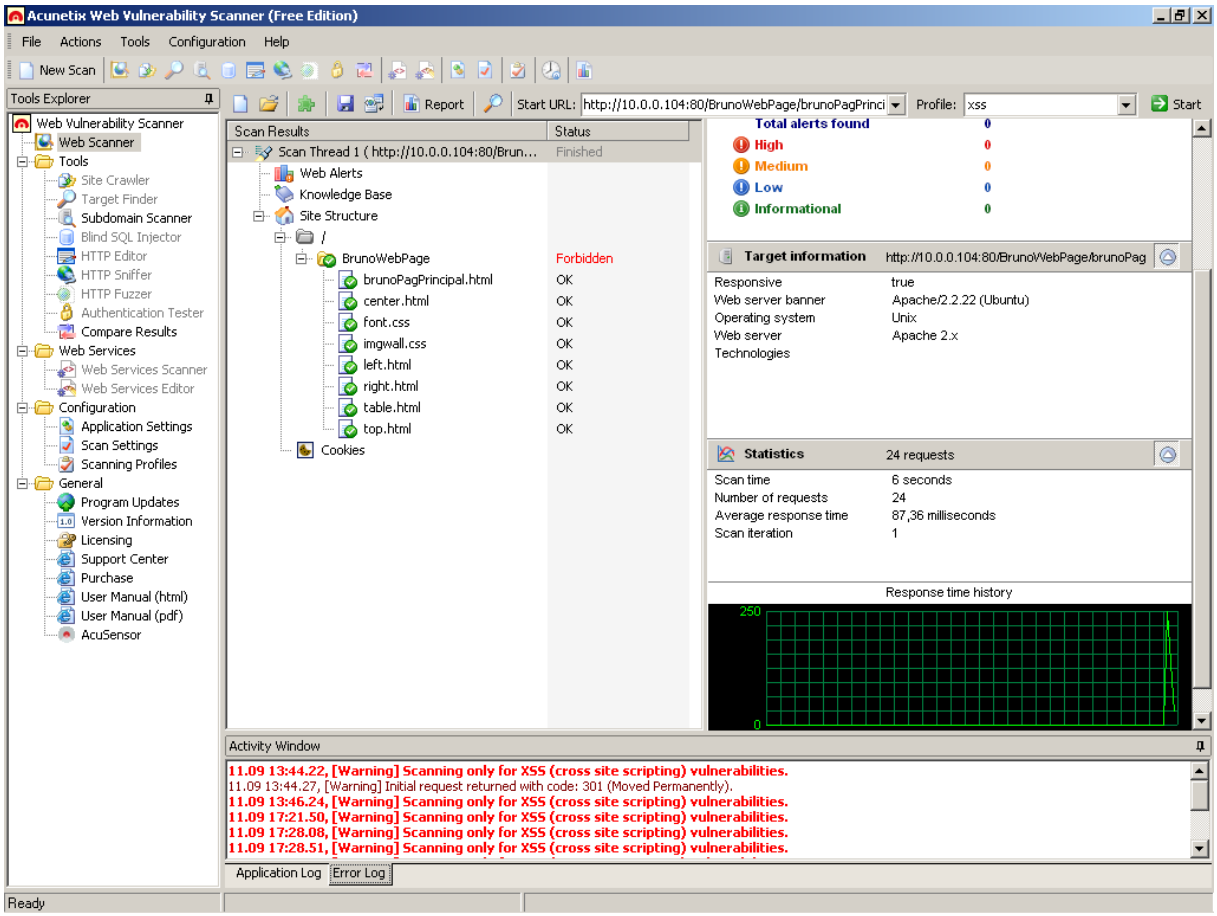


Figura 1: Interface do Acunetix Web Vulnerability Scanner 8.0 Free Edition.
Fonte: do Autor.

2.2.1.2. Nikto

Conforme descrito em (CIRT, 2012), Nikto é uma ferramenta para detectar vulnerabilidades. Tem como característica ser um *scanner* escrito em Perl e especialmente desenvolvido para ser usado em detecções de vulnerabilidades em servidores *web*. Localiza padrões de vários arquivos inseguros, configurações e programas. É simples de ser utilizado e de fácil atualização. tendo como característica gerar registros de *logs* das informações coletadas nos seguintes formatos: a) Relatórios de texto (.txt), b) Registros em formato HTML (*Hypertext Markup Language*), c) Relatórios em XML (*Extensible Markup Language*) e, d) Relatórios em um arquivo CSV (*Comma-separated values*).

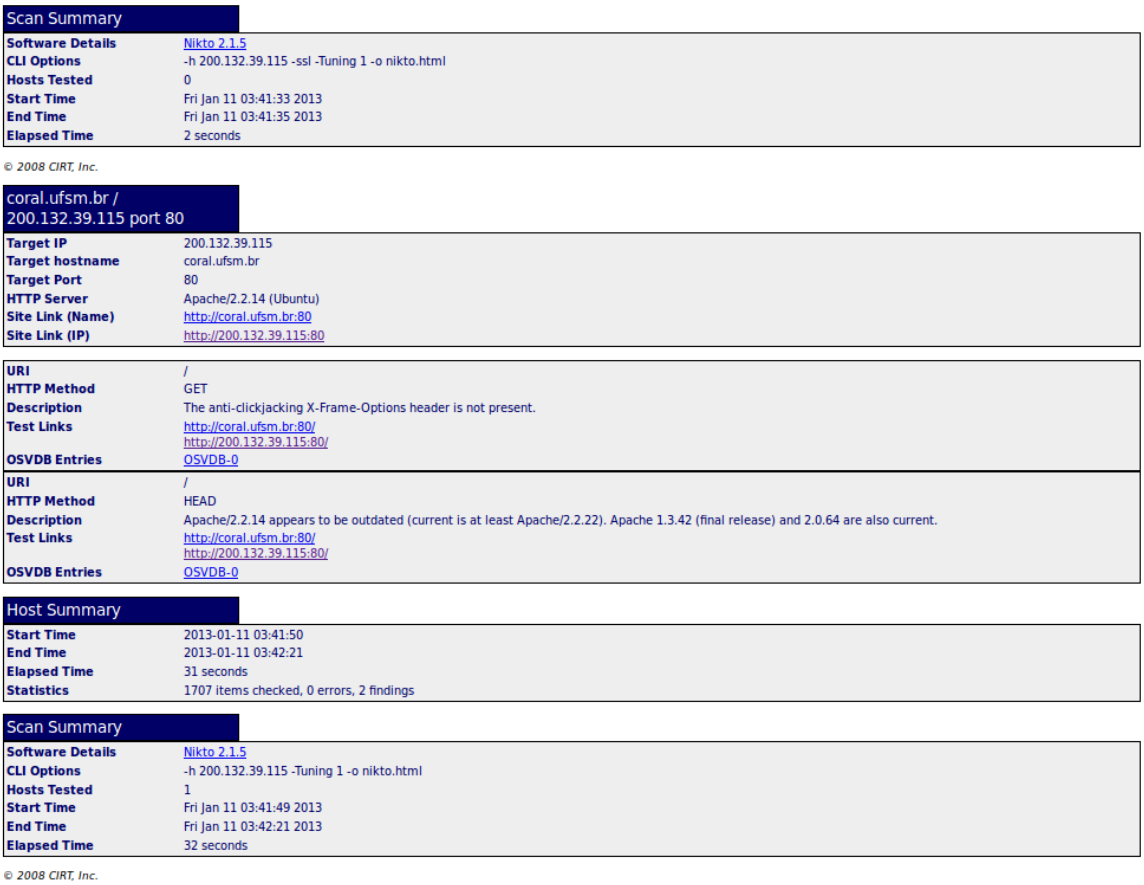
Quanto a avaliação da ferramenta, sobre os aspectos observados referentes

ao tratamento da saída do *scanner*, é que podem ser gerados arquivos de relatórios das informações encontradas na aplicação verificada. Por padrão, registros de uma verificação não são gerados. Tornando-se um fator importante ao pensar em desempenho, pois na existência de um ambiente com grande volume de dados, a verificação de um *site* pode ser uma tarefa demorada. Com o armazenamento de todos estes registros acentua-se ainda mais o problema, demandando mais tempo no processo de verificação. Contudo estes registros de *logs* podem ser gerados a partir da customização do modo de execução da ferramenta, segue um exemplo de comando para execução deste modo para gerar os registros em um dos formatos supra citados.

```
$ perl nikto.pl -h 200.132.39.115 -Tuning 1 -o log_nikto.html
```

Onde *perl nikto.pl* refere-se à execução da classe principal da linguagem de programação Perl da ferramenta Nikto. O parâmetro *-h* é o endereço IP do servidor *web* à ser verificado. O parâmetro -*Tuning* é para definir a opção de ajuste de controle de teste do Nikto, onde o 1 é para atividades de *logs.* E por fim, o parâmetro *-o* refere-se à saída para armazenar os registros de eventos da ferramenta, lembrando que pode-se salvar o arquivo apenas nos formatos suportados.

Foi utilizada a versão 2.1.5 do Nikto, sendo o ambiente de testes da ferramenta executado na plataforma Linux, distribuição Ubuntu 12.04LTS. Porém quanto à aspectos visuais, a ferramenta não possui interface gráfica, a vantagem é que, a mesma pode gerar um arquivo, como por exemplo o arquivo gerado com o comando acima mostrado na Figura 2.



Scan Summary

Software Details	Nikto 2.1.5
CLI Options	-h 200.132.39.115 -ssl -Tuning 1 -o nikto.html
Hosts Tested	0
Start Time	Fri Jan 11 03:41:33 2013
End Time	Fri Jan 11 03:41:35 2013
Elapsed Time	2 seconds

© 2008 CIRT, Inc.

coral.ufsm.br / 200.132.39.115 port 80

Target IP	200.132.39.115
Target hostname	coral.ufsm.br
Target Port	80
HTTP Server	Apache/2.2.14 (Ubuntu)
Site Link (Name)	http://coral.ufsm.br:80
Site Link (IP)	http://200.132.39.115:80

URI	/
HTTP Method	GET
Description	The anti-clickjacking X-Frame-Options header is not present.
Test Links	http://coral.ufsm.br:80/ http://200.132.39.115:80/
OSVDB Entries	OSVDB-0

URI	/
HTTP Method	HEAD
Description	Apache/2.2.14 appears to be outdated (current is at least Apache/2.2.22). Apache 1.3.42 (final release) and 2.0.64 are also current.
Test Links	http://coral.ufsm.br:80/ http://200.132.39.115:80/
OSVDB Entries	OSVDB-0

Host Summary

Start Time	2013-01-11 03:41:50
End Time	2013-01-11 03:42:21
Elapsed Time	31 seconds
Statistics	1707 items checked, 0 errors, 2 findings

Scan Summary

Software Details	Nikto 2.1.5
CLI Options	-h 200.132.39.115 -Tuning 1 -o nikto.html
Hosts Tested	1
Start Time	Fri Jan 11 03:41:49 2013
End Time	Fri Jan 11 03:42:21 2013
Elapsed Time	32 seconds

© 2008 CIRT, Inc.

Figura 2: Arquivo de registro gerado com o Nikto 2.1.5.
Fonte: do Autor.

2.2.1.3. ModSecurity

ModSecurity é um *firewall* de aplicações *web open source*, este é um dos projetos presentes na comunidade aberta de segurança de *software* livre (OWASP, 2012). Este *software* foi implementado para estabelecer uma camada de segurança externa. Sendo assim, este *firewall* tem por objetivo, aumentar a segurança, detectar e prevenir possíveis ataques, antes mesmo que estes ataques acabem atingindo seus respectivos alvos, neste caso a aplicação *web*.

Esta ferramenta tem como característica ser um *firewall* de aplicação *web* embutido e com adaptabilidade, o que significa que ele pode ser implantado como parte de uma infra-estrutura existente do servidor *web* (Apache, IIS 7 e Nginx). Outra característica relevante é que o ModSecurity é multiplataforma, suportando variados

sistemas operacionais, dentre eles: Linux; Windows; Solaris; FreeBSD; OpenBSD; NetBSD; AIX; Mac OS X; HP-UX (MODSECURITY, 2012).

Segundo o *site* da ferramenta (MODSECURITY, 2012), com o ModSecurity é possível fazer *logs* completos de transações HTTP, permitindo que sejam registradas tanto as requisições quanto as respostas. Sua facilidade para criação de *logs* também permite refinadas decisões a serem tomadas sobre o que exatamente é registrado e quando. Este ainda garante que somente dados relevantes são registrados.

A vantagem do ModSecurity é que, o mesmo possui um bom suporte, com uma grande comunidade envolvida (até pelo vasto tamanho do projeto). O ModSecurity possui ferramentas que podem ser acopladas como *plug-ins* para trabalharem como ferramentas dos registros de *logs* do ModSecurity, fazendo-se uma analogia é como se fossem os módulos do Uniscan (ROCHA, 2012).

A AuditConsole (AUDITCONSOLE, 2012) é uma delas, a Figura 3 ilustra os *logs* com esta ferramenta. Seu acesso é via interface *web.* Sendo assim, pela grande implantação de ferramentas já consolidadas no mercado para se trabalhar com gerenciamento de informações, foi definido que a aplicação implementada seguirá no mesmo caminho, com acesso via *browser.* Aspecto bastante interessante, por se tratar de uma aplicação que deverá futuramente funcionar sobre uma distribuição servidora, além do mais, tendo um grande número de possibilidades de se trabalhar com o gerenciamento via *browser*, como demonstram a maioria das ferramentas de gerenciamento atuais. Com esta ferramenta, cria-se a possibilidade de carregar os arquivos de *logs* do *firewall* de aplicação ModSecurity, ideia esta que será adaptada à interface de gerenciamento do UniscanWAF, a WIMU.

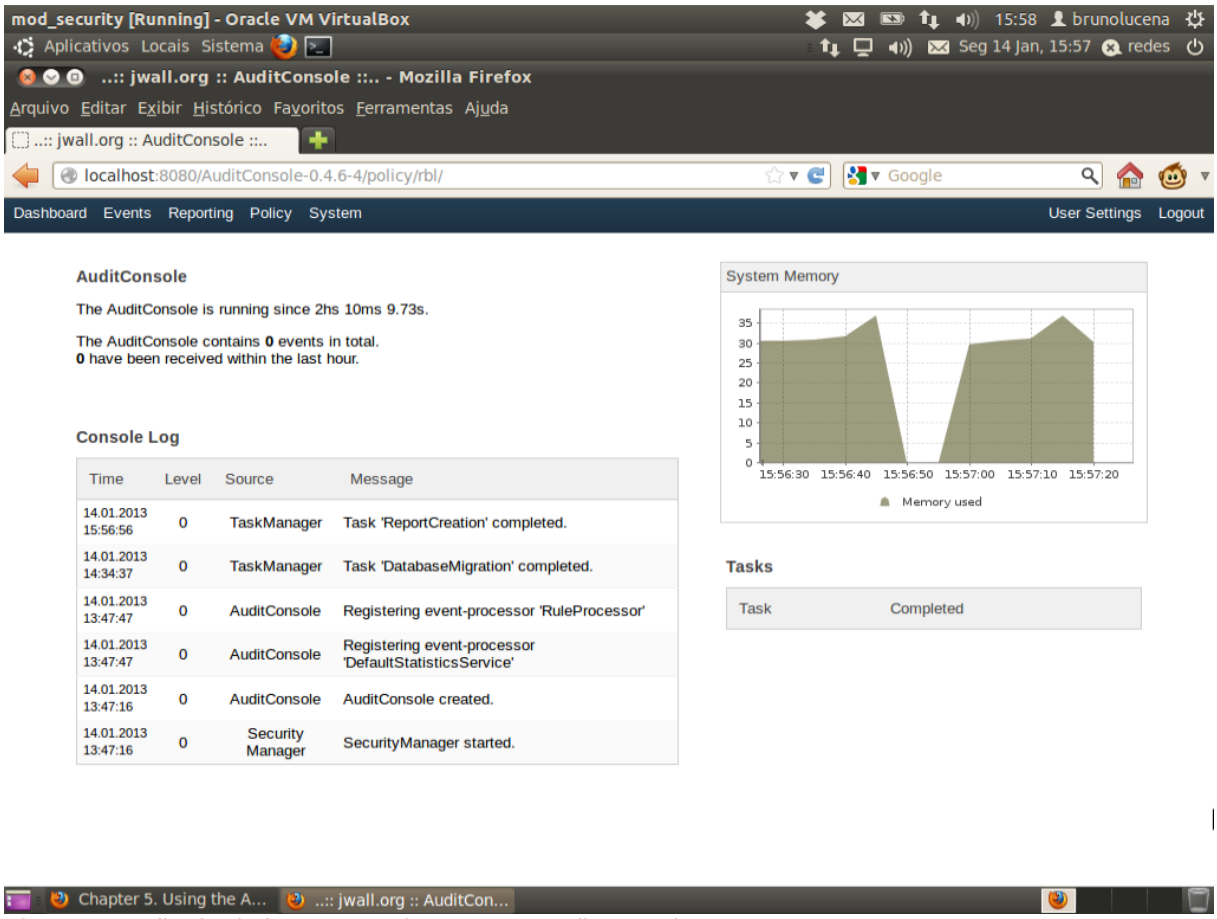


Figura 3: Auditoria de logs com a ferramenta AuditConsole.
Fonte: do Autor.

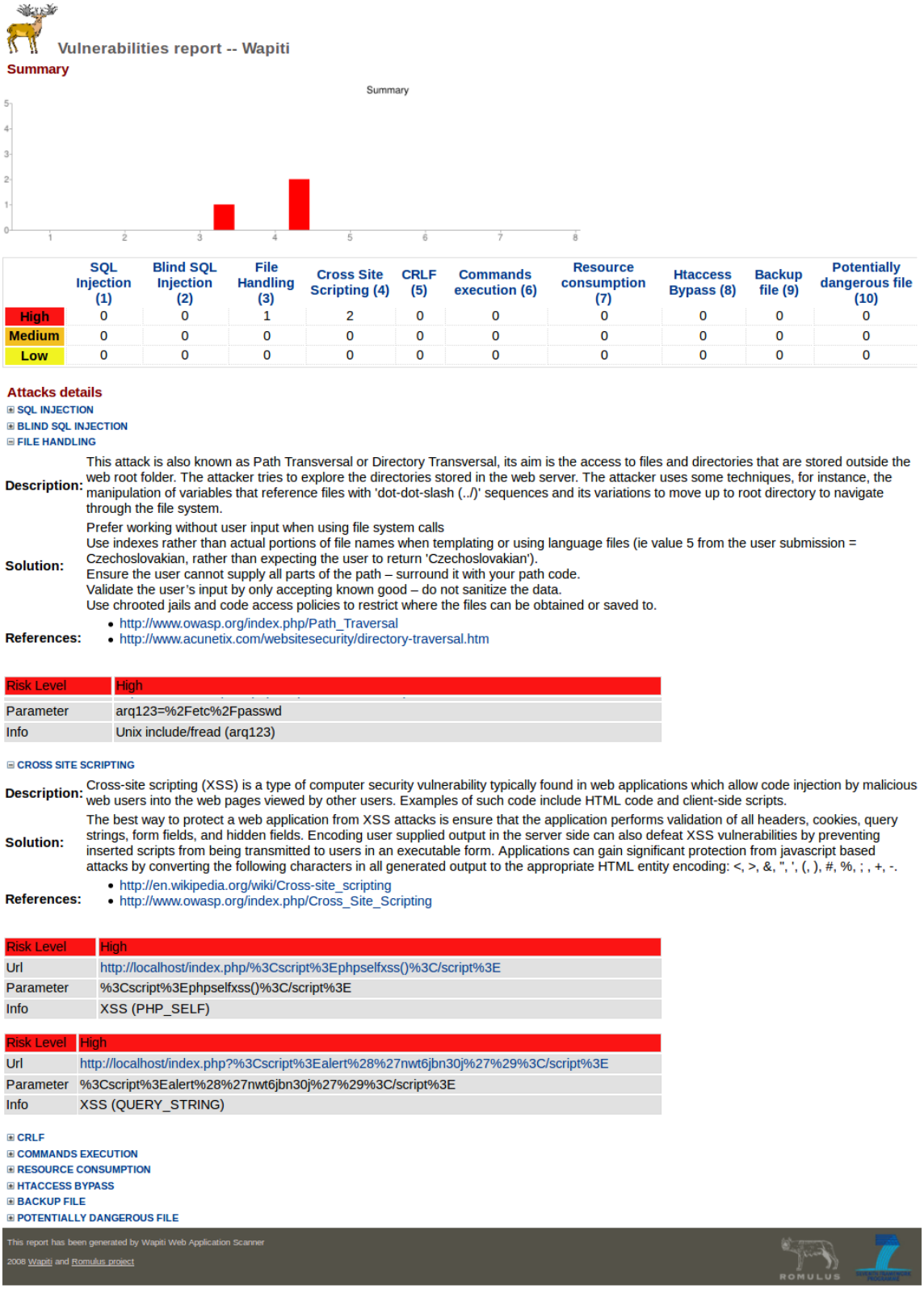
2.2.1.4. Wapiti

Wapiti é uma ferramenta licenciada por GNU *General Public License Version 2* (GPLv2). Ela permite auditar a segurança de suas aplicações *web*. O *scanner* faz varreduras "*Black-box*", ou seja, não examina o código fonte da aplicação, mas verifica as páginas *web* da aplicação *web* implantada, procurando os *scripts* e as formas em que podem-se injetar dados. Logo que recebe uma dada lista para checagem, o Wapiti age como um *fuzzer*, injetando cargas para ver se um *script* é vulnerável. Esta ferramenta pode detectar as seguintes vulnerabilidades: a)Erros de manipulação de arquivos (LFI, RFI, *require, fopen, readfile*); b) Injeção de banco de dados (injeções PHP, JSP, ASP, SQL e XPath); c)Injeções XSS (*Cross Site*

Scripting); d) Injeções LDAP; e) Detecção de comando de execução (*eval (), system(), passtru()*) e, f)Injeções CRLF. (WAPITI, 2012).

Segundo os desenvolvedores (WAPITI, 2012), esta ferramenta imprime um aviso de prevenção cada vez que é gerado um *script* HTTP carregado, bem como, quando é exibido o código de erro 500 HTTP (útil para ASP / IIS). O Wapiti não depende de um banco de dados de vulnerabilidades, pois, este visa encontrar vulnerabilidades desconhecidas em aplicações *web*. Atualmente a ferramenta não fornece uma interface gráfica e deve-se utilizá-la a partir de *text-line* (linha de comando) no terminal.

Para execução da ferramenta foi utilizado um cenário real, onde foi verificado um servidor *web* na *Internet*, mostrando a seguir o comando efetuado no terminal e em seguida é mostrado na Figura 4 os resultados obtidos, para questões referentes à registros possibilitando análise que a ferramenta proporciona. O Wapiti foi executado em um sistema operacional Linux, distribuição Ubuntu 12.04LTS.



Vulnerabilities report -- Wapiti

Summary

Summary

	SQL Injection (1)	Blind SQL Injection (2)	File Handling (3)	Cross Site Scripting (4)	CRLF (5)	Commands execution (6)	Resource consumption (7)	Htaccess Bypass (8)	Backup file (9)	Potentially dangerous file (10)
High	0	0	1	2	0	0	0	0	0	0
Medium	0	0	0	0	0	0	0	0	0	0
Low	0	0	0	0	0	0	0	0	0	0

Attacks details

SQL INJECTION

BLIND SQL INJECTION

FILE HANDLING

Description: This attack is also known as Path Transversal or Directory Transversal, its aim is the access to files and directories that are stored outside the web root folder. The attacker tries to explore the directories stored in the web server. The attacker uses some techniques, for instance, the manipulation of variables that reference files with 'dot-dot-slash (../)' sequences and its variations to move up to root directory to navigate through the file system.

Solution: Prefer working without user input when using file system calls
Use indexes rather than actual portions of file names when templating or using language files (ie value 5 from the user submission = Czechoslovakian, rather than expecting the user to return 'Czechoslovakian').
Ensure the user cannot supply all parts of the path – surround it with your path code.
Validate the user's input by only accepting known good – do not sanitize the data.
Use chrooted jails and code access policies to restrict where the files can be obtained or saved to.

References:
- http://www.owasp.org/index.php/Path_Traversal
- http://www.acunetix.com/websitesecurity/directory-traversal.htm

Risk Level	High
Parameter	arq123=%2Fetc%2Fpasswd
Info	Unix include/fread (arq123)

CROSS SITE SCRIPTING

Description: Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications which allow code injection by malicious web users into the web pages viewed by other users. Examples of such code include HTML code and client-side scripts.

Solution: The best way to protect a web application from XSS attacks is ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields. Encoding user supplied output in the server side can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding: <, >, &, ", ', (,), #, %, ; , +, -.

References:
- http://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.owasp.org/index.php/Cross_Site_Scripting

Risk Level	High
Url	http://localhost/index.php/%3Cscript%3Ephpselfxss()%3C/script%3E
Parameter	%3Cscript%3Ephpselfxss()%3C/script%3E
Info	XSS (PHP_SELF)

Risk Level	High
Url	http://localhost/index.php?%3Cscript%3Ealert%28%27nwt6jbn30j%27%29%3C/script%3E
Parameter	%3Cscript%3Ealert%28%27nwt6jbn30j%27%29%3C/script%3E
Info	XSS (QUERY_STRING)

CRLF

COMMANDS EXECUTION

RESOURCE CONSUMPTION

HTACCESS BYPASS

BACKUP FILE

POTENTIALLY DANGEROUS FILE

This report has been generated by Wapiti Web Application Scanner

2008 Wapiti and Romulus project

ROMULUS

Figura 4: Resultados para auditoria gerados com o Wapiti.
Fonte: do Autor.

2.2.1.5. Loggly

Conforme o *site* da ferramenta (LOGGLY, 2012), Loggly é um serviço baseado em nuvem, sendo classificado como SaaS (*Software as a Service*). Sendo assim, é um *software* como serviço para gerenciamento de grandes volumes de informações de *logs.* Tem como principal característica atrativa a simplicidade para tal tarefa (gerenciamento de registros de *logs*), sem a necessidade de instalação de nenhum *software*, sendo toda a aplicação executada na nuvem.

Para se ter este tipo de serviço é muito simples, basta criar uma conta no *site* da aplicação <http://www.loggly.com> e usufruir dos recursos. Não é uma ferramenta gratuita, mas a vantagem é que novos usuários têm trinta dias de serviço grátis. Um aspecto importante é que a aplicação é acessada com a utilização do protocolo *https*, pois o acesso é via interface *web* tornando a aplicação mais segura e confiável.

Sendo assim, o Loggly é uma ferramenta bastante robusta, sendo completamente baseada em *cloud* (nuvem). Sem a preocupação com *hardware*, *software*, instaladores, *downloads* e armazenamento de dados. Desta maneira, a funcionalidade de gerenciar imensos registros de *logs* torna-se uma tarefa simples e agradável. Para se fazer gestão e auditoria destes dados, oriundos de uma determinada aplicação do usuário, o Loggly mostra-se uma boa ferramenta. O mesmo tem como característica possuir escalabilidade, permitindo que o usuário tome medidas pró-ativas para melhorar o desempenho na sua aplicação, a partir dos seus registros de *logs*.

Para o gerenciamento dos registros de *logs*, o Loggly conta com um conjunto de características, possibilitando a análise destas informações em painéis com gráficos embutidos, permitindo comparação de valores, análises numéricas, mecanismos de buscas e, mensagens de alertas, para auxílio do usuário administrador da sua base de dados de *logs* (LOGGLY, 2012). Na Figura 5, há uma demonstração da interface da ferramenta.

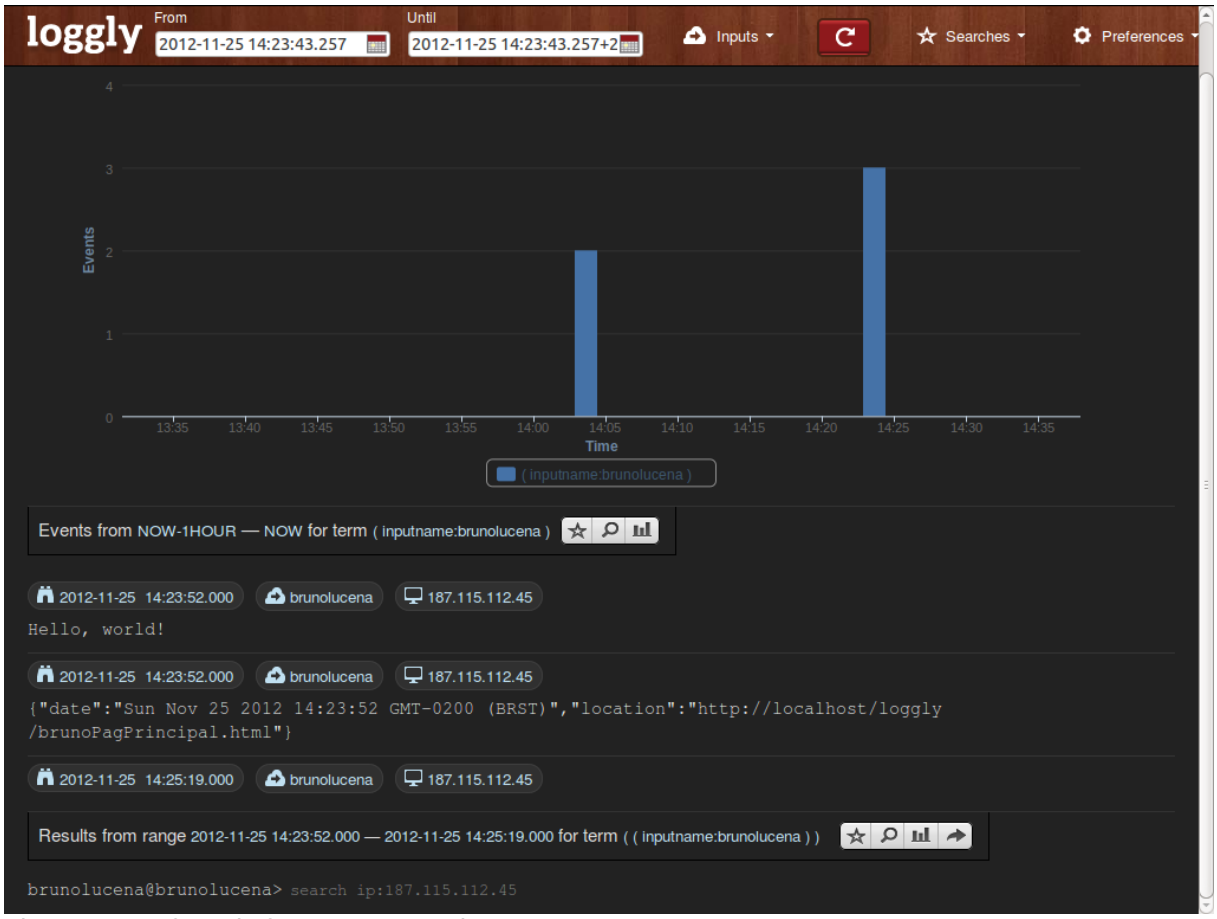


Figura 5: Interface da ferramenta Loggly.
Fonte: do Autor.

2.2.1.6. Logstash

Ferramentas completamente gratuitas e totalmente *open sources* são sempre bem vindas no mercado, Logstash é uma delas. Segundo seus desenvolvedores, no *site* da ferramenta (LOGSTASH, 2012), o objeto da Logstash é disponibilizar o gerenciamento de registros de *logs,* bem como, a gerência de eventos. Com esta ferramenta é possível trabalhar-se com os *logs* coletados, assim como, fazer análise dos mesmos, como uma auditoria destes dados e, armazená-los para, no caso, um uso posterior (como por exemplo, com seleção de registros de um determinado acontecimento que demanda bastante tempo para conclusão desta pesquisa).

A análise dos dados é dos seus pontos chaves, pois possui uma interface *web* que permite tal fator, com mecanismos para se obter um bom desempenho na

auditoria destes registros. O Logstash é uma ferramenta para auxiliar o gerente de um sistema, na implantação de registros de *logs* e, outros dados de eventos de diferentes aplicações, permitindo coletar tudo e centralizá-los em um único lugar (LOGSTASH, 2012). Segue na Figura 6, uma demonstração da interface desta ferramenta.

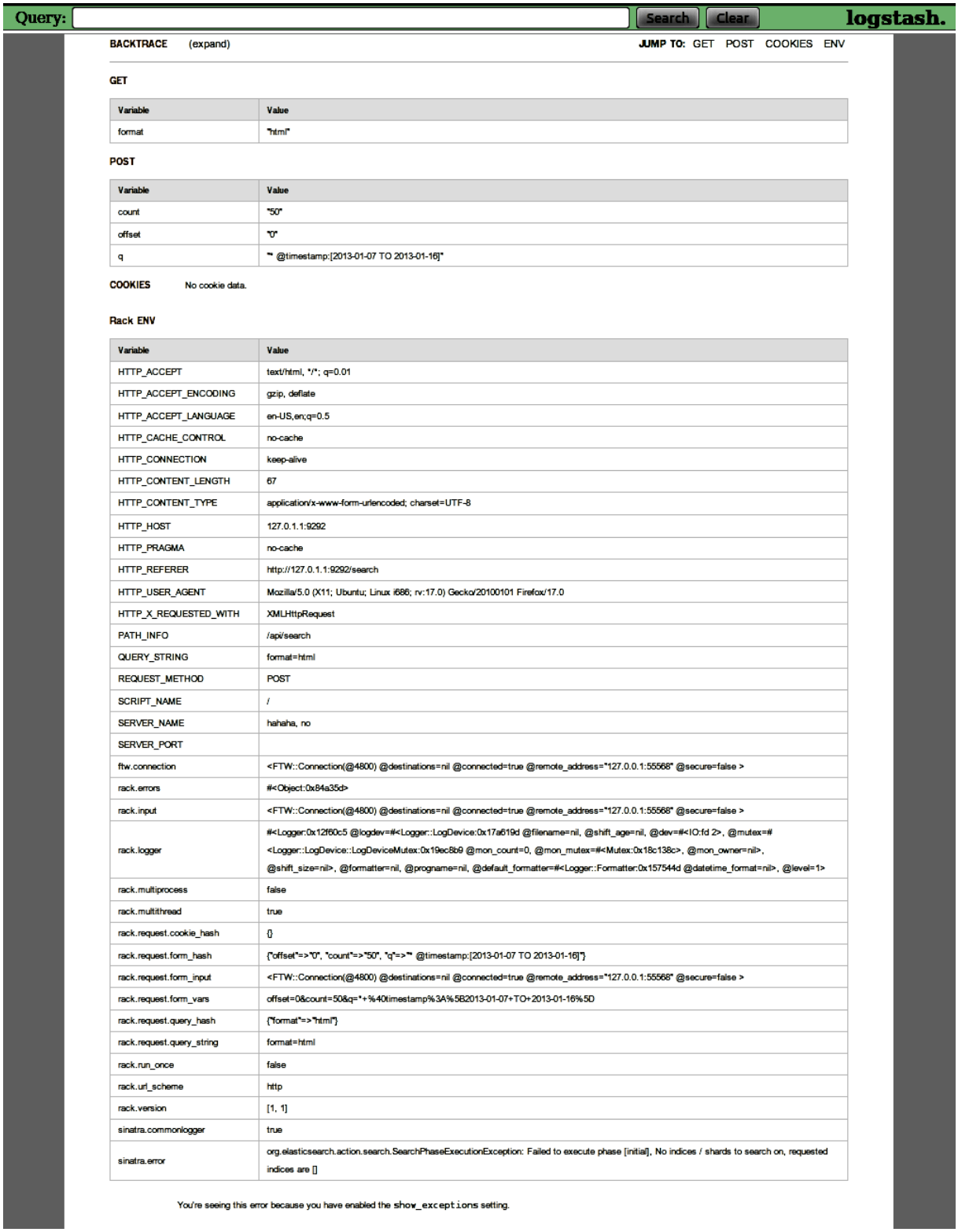


Figura 6: Interface da ferramenta Logstash.
Fonte: do Autor.

2.3. Conclusões Parciais

Com as ferramentas analisadas, pode-se concluir que a área de gerenciamento de segurança, para auditoria de *logs* e eventos em aplicações voltadas para a detecção de vulnerabilidades *web*, bem como, aplicações voltadas para a gerência de registros de *logs*, ainda é um aspecto bastante negligenciado por aplicações com estas características, pois é notável uma certa carência de utilização de recursos (existentes hoje) para se implantar boas maneiras de se aplicar o gerenciamento dessas ferramentas, pensando em auxiliar um gerente de rede.

Aplicações como Loggly e Logstash, as quais são voltadas para o tratamento de dados e gerenciamento de registros de *logs,* possuem bons mecanismos para auxílio na análise e auditoria das suas informações coletadas e/ou simplesmente importadas, como no caso da ferramenta AuditConsole (*Logging Tool*) do ModSecurity. Os serviços destas aplicações possuem uma considerável vantagem, em questões de gerenciamento, pois são ferramentas específicas para tal funcionalidade.

Geralmente, o que temos em cenários atuais, para área de gerenciamento já está praticamente fixado (não mudam as tecnologias e os métodos para se gerenciar aparentam despontar para uma constante). Subentende-se, desta forma que o serviço de um gerente de rede (principalmente em aplicações com grandes bases de dados) é uma tarefa bastante complexa e, um dos fatores que dificulta a gerência é que a maioria das aplicações *web* carece de boas técnicas de gerenciamento, que na maioria dos casos é bastante vulnerável a ataques, por conta do cenário hostil que a *Internet* proporciona.

Sendo assim, é de grande valia, todo e qualquer método para contribuir na tarefa complexa de um gerente. Com as ferramentas para gerenciamento existentes atualmente há como garantir certo nível de segurança, porém aplicações malignas sempre emergem na *web,* tornando a necessidade de buscar novas formas de proteção, surgiram desta maneira, as aplicações voltadas para a área de detecção de vulnerabilidades *web,* como parte das ferramentas analisadas neste trabalho.

Foi observado, em geral, uma certa complexidade nos meios de

gerenciamento para as detecções que as ferramentas resultaram. Por este motivo, este trabalho tem por objetivo, implementar um sistema de registros em *log*, bem como, desenvolver uma interface *web* para se aplicar técnicas de gerenciamento para tratar a saída do UniscanWAF, uma ferramenta utilizada para proteção de serviços *web*, no caso um *firewall* de aplicações *web*.

3. ARQUITETURA PROPOSTA

Este capítulo será dividido em três seções. Em um primeiro momento, na seção 3.1 será apresentada a proposta do estudo em questão. A seguir na seção 3.2 será apresentada a arquitetura de funcionamento da aplicação de gerenciamento. E por fim, na seção 3.3 serão descritas as ferramentas essenciais para elaboração deste trabalho, bem como suas características e funcionalidades.

3.1. Proposta de Estudo

A segurança da informação sempre foi um tópico que necessitou cuidados exclusivos por todas as pessoas, em diversos lugares, essencialmente nas grandes organizações. Com a expansão do fluxo de informações trocadas em uma rede, a preocupação em manter todo este volume de dados seguro é um bom motivo para alertar a comunidade dos administradores de rede. Dentre outros fatores, com a evolução das tecnologias, exige-se um maior grau à implementações de níveis de segurança para acompanhar tais avanços tecnológicos. Surgiu assim, segundo as exigências de segurança, a ferramenta principal utilizada neste trabalho, o UniscanWAF (KUROSE, 2010).

Não é uma tarefa simples definir o número de informações que uma aplicação *Web* pode tratar diariamente, a escalabilidade é um fator importante para conseguir oferecer um serviço adequado mesmo nos horários de maior número de acesso. Conforme cresce o número de acessos aumenta-se o número de informações recebidas pela aplicação *Web*, tendo a necessidade de haver um tratamento adequado destes dados para possíveis ações ativas ou, melhor ainda, pró-ativas. Hoje, o UniscanWAF tem a capacidade de detectar vulnerabilidades *Web.* Entretanto, há uma necessidade de se tratar adequadamente as informações capturadas pela ferramenta, uma vez que, atualmente são gerados *logs* que impossibilitam a análise detalhada dos dados capturados. Sendo assim, o

desenvolvimento de técnicas para gerenciamento do *firewall* de aplicação UniscanWAF, é de grande valia, possibilitando futuros estudos para implementação de novos métodos para a gerência das informações obtidas.

O presente estudo é centrado na área de tratamento de dados brutos extraídos do UniscanWAF, possibilitando visualizar de forma clara os dados da ferramenta. Para tanto, são gerados gráficos, através da biblioteca RRDtool. Não faz parte deste trabalho implementar novas regras de detecção para vulnerabilidades em aplicações *web,* tampouco avaliar a precisão dos resultados apresentados pela aplicação. Entretanto pretende-se divulgar os resultados da melhor forma possível ao gerente de segurança da informação, facilitando o compreendimento do gerente para auditoria e interpretação da funcionalidade do UniscanWAF, possibilitando este responsável fazer análises estatísticas.

A proposta inicial deste estudo era desenvolver técnicas para gerenciamento do *firewall* de aplicação UniscanWAF de forma a tratar toda a saída resultante do monitoramento da ferramenta, auxiliando o gerente de rede na tomada de decisão. Contudo estas técnicas ficaram limitadas à implementação de um registro de eventos *(logs),* geração de gráficos e análise com consultas permitindo gerar dados estatísticos dos acontecimentos registrados.

O projeto em estudo está estruturado em etapas, a parte inicial foi a busca de ferramentas que pudessem contribuir com a proposta do estudo, ou seja, ferramentas similares para se buscar ideias. Após esta etapa será a aplicação prática do projeto para à obtenção dos resultados. Para isto serão implantados e desenvolvidos métodos para facilitar a visualização da saída do UniscanWAF. A estruturação dos dados para a implementação dos *logs* em uma nova sintaxe será o primeiro procedimento prático, após ter-se os registros de *logs* serão desenvolvidas técnicas para auxiliar a auditoria e análise destes dados, com a possibilidade de se fazer estatísticas destas informações.

Ao obter os *logs* já estruturados para a leitura da biblioteca RRDtool, iniciará a geração e análise dos gráficos a partir da utilização dos recursos dísponíveis na biblioteca do presente projeto.

Por fim, serão aplicadas as implementações e métodos para gerenciamento do *Firewall* de Aplicação UniscanWAF, possibilitando consultas para gerar dados

estatísticos. Serão abordados aspectos de gerenciamento de segurança, e também serão relacionados aspectos de gerenciamento de contabilidade, com a utilização da análise dos resultados visualizados nos gráficos, encaminhando-se assim, com os resultados obtidos, a etapa final de estudos do projeto. A Figura 7, ilustra o funcionamento da aplicação de gerenciamento, no caso, futuramente, a interface a ferramenta WIMU.

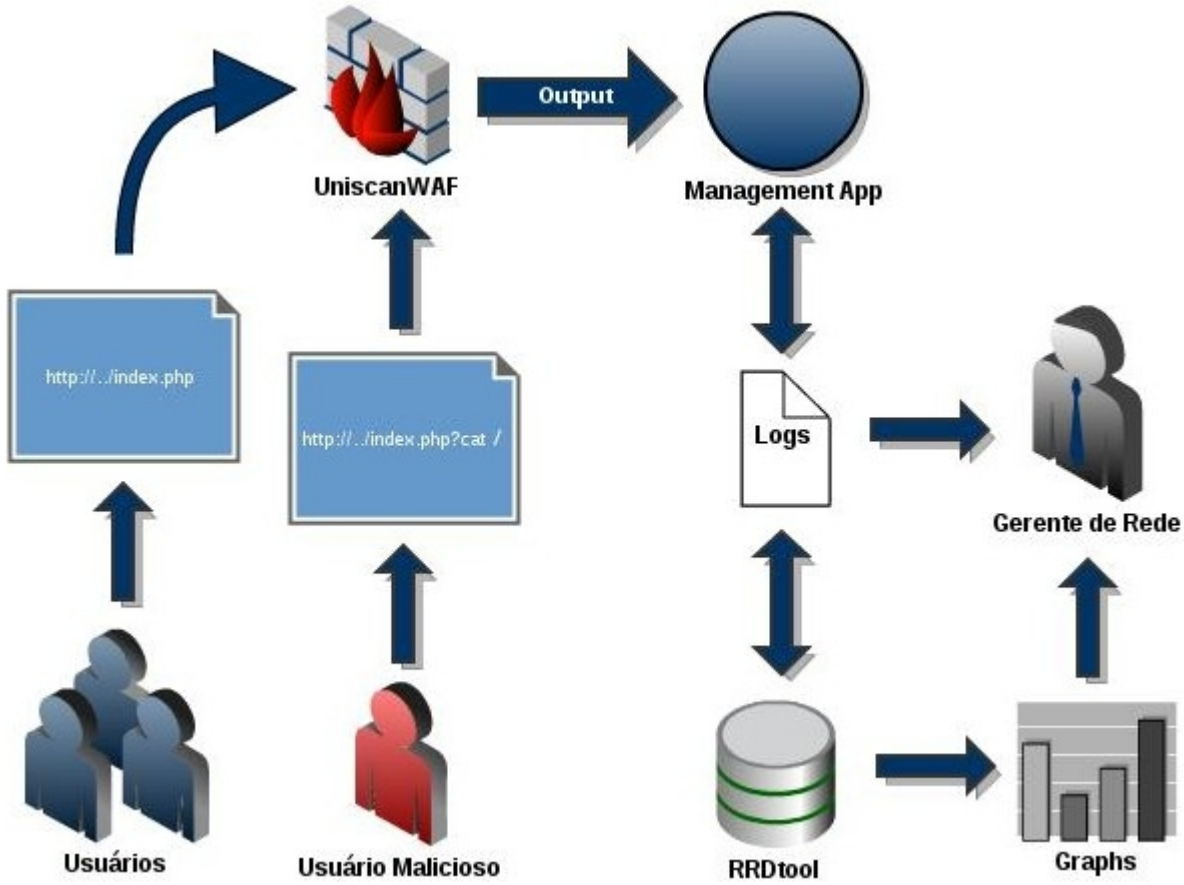


Figura 7: Funcionamento da Aplicação de Gerenciamento.
Fonte: do Autor.

Conforme visto na Figura 7, o funcionamento da aplicação de gerenciamento é bastante simples. Ocorrendo da seguinte forma: Os usuários fazem requisições ao servidor *web,* considerando que estes podem ser um usuário malicioso, enviando uma requisição contendo uma vulnerabilidade, como por exemplo na ilustração

“*http://../index.php?cat /*”. Todas as requisições com destino ao servidor *web* (porta 80) são processadas pelo *firewall* de aplicação, gerando uma saída (dados brutos) do UniscanWAF, na figura representada pela seta “*Output*”. A partir desta saída é feito o tratamento destes eventos na aplicação de gerenciamento. Resultando na implementação de registros em *Logs.* De posse destas informações contidas nos *logs*, a aplicação de gerenciamento permite que sejam gerados gráficos (transformando as informações contidas nos *logs* em uma sintaxe padrão de entrada de dados da biblioteca RRDtool). Por fim, tanto a análise dos arquivos de *logs*, quanto a visualização dos gráficos, podem ser gerenciadas pelo administrador da rede.

3.2. Arquitetura da Aplicação de Gerenciamento

O UniscanWAF tem a característica de possuir uma arquitetura modular. Este fator é um ponto positivo por facilitar a adição de novas funcionalidades, fazendo com que a ferramenta possa evoluir rapidamente. Partindo deste princípio, de contribuir para a evolução do UniscanWAF, é que foram desenvolvidos estes tratamentos da saída do Firewall, com a implementação de registros de *logs* e geração de gráficos. A arquitetura de funcionamento desta aplicação de gerenciamento (nomeou-se desta forma) é demostrada à seguir na Figura 8.

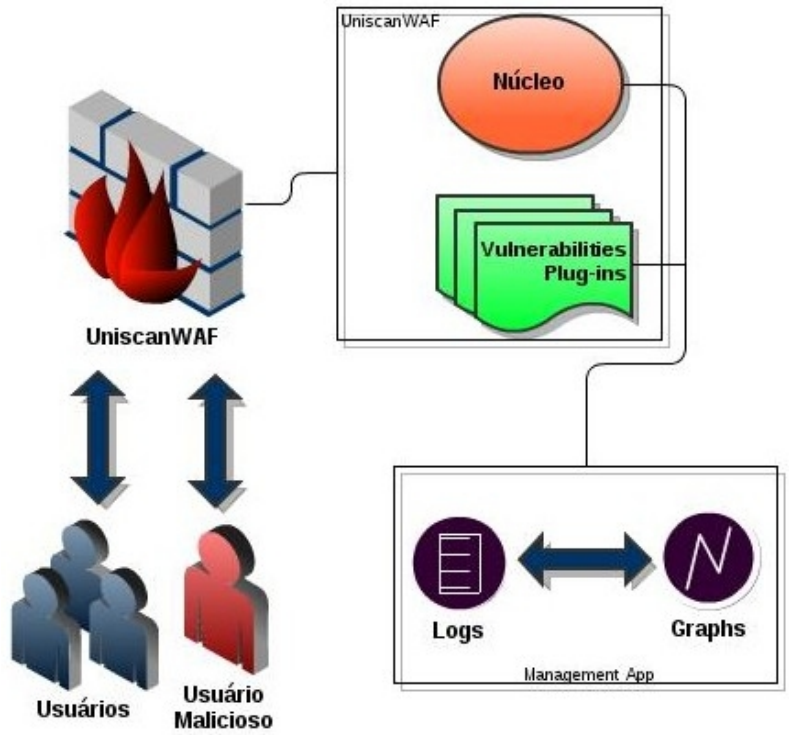


Figura 8: Arquitetura da Aplicação de Gerenciamento
Fonte: do Autor.

Onde são adicionados no núcleo do UniscanWAF, juntamente com adaptações nos *plug-ins* de tipos de vulnerabilidades, bem como, no módulo de inicialização do *Firewall*, trechos de códigos e adição de classes em Perl, para tais funcionalidades.

3.3. Ferramentas Utilizadas

As ferramentas essenciais para a elaboração deste estudo são, na subseção 3.3.1, o Firewall de Aplicação UniscanWAF, onde é feita a proposta principal deste trabalho é o aprimoramento desta ferramenta, para isto, são feitas as implementações descritas à seguir no capítulo 4. E na subseção 3.3.2 a biblioteca RRDtool (*Round Robin Database Tool*), utilizada para geração dos gráficos.

3.3.1. UniscanWAF: *Firewall* de Aplicação *Web*

Praticamente todas as grandes organizações, órgãos governamentais e, diversos cidadãos, possuem *sites Web.* O número de indivíduos e organizações com acesso à *Internet* está aumentando gradativamente. Em virtude disto, as organizações demonstram grande interesse em utilizar a *Web* para diversos tipos de serviços (STALLINGS, 2008). Entretanto, é sabido que ambientes *Web* são extremamente vulneráveis a riscos de vários tipos, dentre eles, os ataques à vulnerabilidades *Web.* A ferramenta UniscanWAF tem por objetivo fazer o tratamento destes problemas. À medida que as organizações percebem essa realidade, a procura pela segurança dos serviços *Web* aumenta.

Esta ferramenta atua de forma similar a um *proxy Web* com filtro de pacotes, pois, é capaz de detectar e descartar requisições maliciosas em tempo real. O UniscanWAF trabalha com duas maneiras para decidir se libera acesso a um determinado recurso *Web* (FABRO NETO; TURCHETTI, 2012). A primeira estratégia para proteção contra novos ataques, é dada a partir da utilização de uma *Whitelist,* permitindo que o usuário acesse apenas os recursos cadastrados na mesma. Já a segunda estratégia é um conjunto de *plug-ins* responsáveis pela detecção de vulnerabilidades, os quais atuam na verificação de ataques através de regras pré-definidas. Podemos chamar este conjunto de *plug-ins* (contabilizando cinco no total, sendo um para cada tipo de vulnerabilidade, as quais serão descritas mais adiante) de *Blacklist.*

Quanto ao funcionamento o UniscanWAF, este trabalha impedindo que uma requisição maliciosa alcance o servidor *Web.* Portanto, o mesmo fica esperando por requisições que tenham como destino a porta 80 e trata a requisição de maneira adequada. Quando o UniscanWAF recebe uma requisição, encaminha a mesma para verificação, onde é detectado o tipo de vulnerabilidade, se esta existir o pacote é descartado. Caso contrário, é encaminhado para o servidor *Web*, encarregado de responder as requisições não maliciosas do usuário que requisitou o recurso. Ressalta-se que o UniscanWAF trabalha com um sistema de regras ou assinatura de

ataques. Isso significa que ele detecta vulnerabilidades cadastradas no seu banco de vulnerabilidades. Essa característica reflete uma necessidade de sistemas baseados em assinaturas de ataque: a atualização constante das assinaturas, proporcionando longevidade ao WAF e maior proteção aos usuários que fazem uso do mesmo (FABRO NETO; TURCHETTI, 2012). A Seguir na Figura 9, é ilustrado o funcionamento básico da ferramenta.

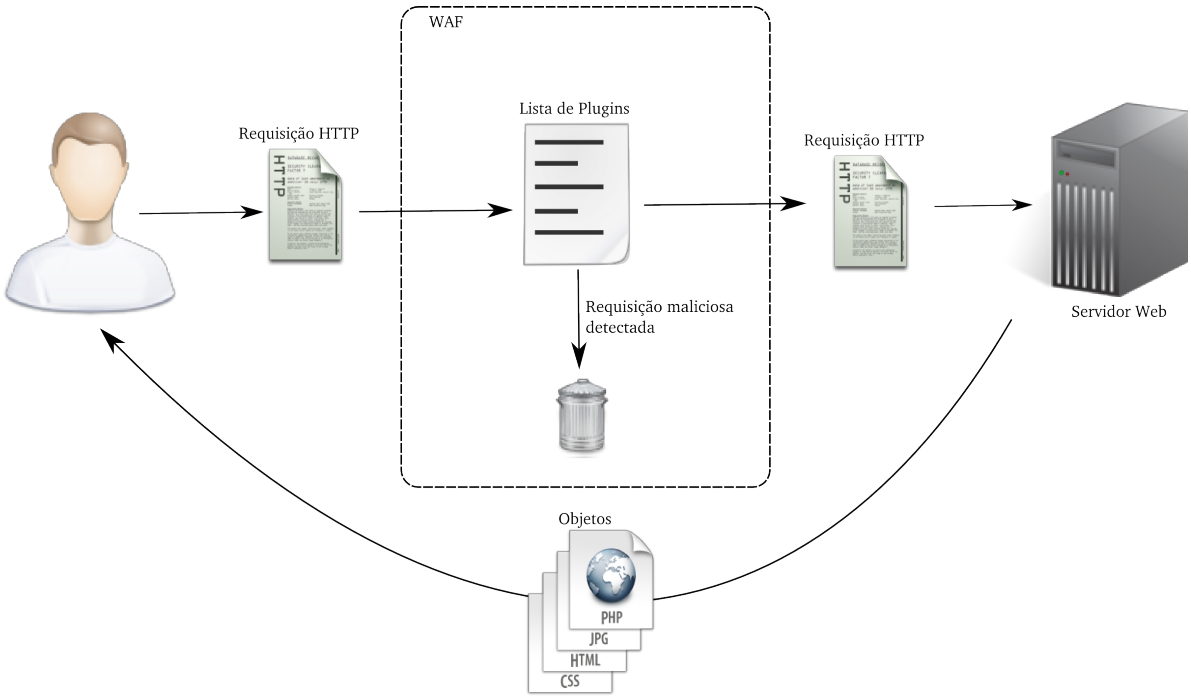


Figura 9: Funcionamento do UniscanWAF.
Fonte: (FABRO NETO; TURCHETTI, 2012).

Como pode-se observar, o *Firewall* é o responsável por fazer a ponte entre o usuário e a aplicação *Web*, monitorando as requisições com destino à porta 80 (HTTP). Na Figura 9, o usuário envia uma requisição deste tipo, onde o UniscanWAF intercepta essa requisição e verifica se a mesma esta presente na *Whitelist,* caso não estiver, a requisição é eliminada. Contudo, se a requisição estiver na *Whitelist*, é enviada para varredura de um possível ataque *Web* presente na *Blacklist*. É importante ressaltar, que a detecção depende do ataque estar cadastrado na base do UniscanWAF (*Blacklist*). Sendo assim, a requisição que não estiver presente em

nenhum dos tipos de ataques da base, é considerada livre de ataques, permitindo, só assim, que chegue à aplicação *Web*.

Quanto à sua arquitetura, a mesma pode ser designada como modular, aspecto bastante importante para organização e adição de novas funções à ferramenta.

Por fim, a utilização desta ferramenta para elaboração do presente estudo, é fruto da necessidade de auxílio à indivíduos voltados à administração de redes, promovendo o desenvolvimento de técnicas para gerenciamento da ferramenta em questão. Hoje, o que acontece, normalmente, é que as organizações só visam a utilização do que há de mais moderno das tecnologias existentes no mercado, não ligando para quem irá garantir a segurança disto tudo, e muito menos como (de que forma). Sendo assim, o auxílio ao gerente de rede é, muitas vezes, um dos processos de menor prioridade. Mas esta visão das organizações em geral está, lentamente, em processo de mudança, tendo este estudo como exemplo.

3.3.1.1. Tipos de Vulnerabilidades

Tanto para o desenvolvimento de *Logs,* quanto para geração de gráficos, é válido saber quais os tipos de vulnerabilidades *web* o *Firewall* de Aplicação UniscanWAF possui em sua base de dados. Sabe-se que para cada regra (tipo de vulnerabilidade) há um arquivo correspondente, que determina as ações a serem executadas pela ferramenta para notificar a existência ou não de uma vulnerabilidade na requisição analisada. Esta característica permite adicionar novas regras ao sistema sob-demanda, tornando um sistema relativamente simples e flexível. Atualmente, o UniscanWAF é capaz de analisar e processar os seguintes tipos de vulnerabilidades *web*: a) LFI (*Local File Include*), b) RFI (*Remote File Include*), c) RCE (*Remote Command Execution*), d) SQL-Injection e, e) XSS (*Cross-site Scripting*).

a) LFI: Segundo (FABRO NETO, TURCHETTI, 2012), é um tipo de vulnerabilidade que possibilita a inclusão de arquivos locais. Com este tipo de

ataque dados e arquivos dos *hosts* servidores *Web* podem ser expostos. Um exemplo de ataque deste tipo é a inclusão do arquivo “/arquivo.php?/etc/passwd”, expondo este arquivo na aplicação .

b) RFI: Parecida com a inclusão de arquivos locais. Porém, nesta vulnerabilidade a inclusão é de um arquivo, não localmente, mas hospedado em outro servidor *Web*. Nestes ataques os riscos de segurança predominantemente serão bem maior, pois o arquivo remoto pode ter sido intencionalmente desenvolvido por um possível atacante, aumentando suas possibilidades de ações maliciosas de maneira simplificada. A inclusão de arquivos remotos ocorre devido negligências de validação e tratamento inadequado das requisições ao servidor *web* (ROCHA, 2012). Um exemplo seria “/arquivo.php?arg=http://atacante.com/comandos.txt”.

c) RCE: Nesta vulnerabilidade um atacante executa comandos remotos em um sistema alvo. Com o comando *system(),* se não tratado, em uma aplicação PHP pode executar comandos do sistema. Um exemplo deste comando é “/rce.php? arg=user4;cat /etc/passwd; ” (FABRO NETO, TURCHETTI, 2012).

d) SQL-Injection: Segundo (FABRO NETO, TURCHETTI, 2012), esta vulnerabilidade é basicamente uma inserção de código SQL (*Structured Query Language*) malicioso com dados de entrada de uma aplicação. Esse tipo de ataque pode acessar um banco de dados, fazer modificações (dependendo, executar operações de administrador do banco) e, em alguns casos, executar comandos do sistema operacional . Exemplo: /show.php?username=user&password=1’ OR ’1 .

e) XSS: Ocorre quando pode-se gerar problemas de validação de usuário, ou seja, quando uma aplicação inclui dados fornecidos pelo usuário sem a validação desses dados, como resultados deste ataque temos o eventual roubo de sessão. Assim como na RFI, aplicações que não tratam de maneira adequada os dados de entrada estão vulneráveis ao ataque. Exemplo: “<ScRiPt >prompt(903975)</ScRiPt>” (ROCHA, 2012).

3.3.2. RRDtool: Round Robin Database Tool

Conforme em (OETIKER, 2009), a ferramenta RRDtool é uma biblioteca que permite ao usuário a criação de gráficos. Uma característica bastante relevante com relação ao RRDtool é a capacidade de integração com outras ferramentas para gerenciamento.

Ferramentas que trabalham com amplo volume de dados são, geralmente, bastante utilizadas por praticamente todas as grandes organizações em que há uma necessidade de informatização, pelo amplo fluxo de informações. Nestes ambientes não é uma tarefa viável reunir informações sobre o estado de diferentes tipos de dados (que vão desde a temperatura em um local de trabalho ao número de octetos que passaram por uma interface no seu roteador) manualmente. O RRDtool é uma ferramenta que permite armazenar estes dados de forma eficiente e sistemática, tendo assim, diversas utilidades. Permite registrar e analisar os dados coletados a partir de todos os tipos de fontes de dados (*Data-Sources*). A parte de análise de dados do RRDtool é baseada na capacidade de gerar representações gráficas de valores dos dados recolhidos em um período de tempo definido (OETIKER, 2009).

A ferramenta RRDtool (*Round Robin Database Tool*) é uma biblioteca que possibilita a geração de gráficos a partir de uma determinada sintaxe própria (padrão de entrada de dados para gerar sua base de dados). Isto é, ela gera gráficos e possui uma sintaxe padrão para a geração e manipulação do seu banco de dados. É bastante utilizada pela vantagem de possuir inúmeras funcionalidades quando integrada a outras ferramentas de gerenciamento em redes de computadores, esta característica é o grande diferencial do RRDtool. Portanto, a biblioteca RRDtool, é amplamente aproveitada pois possui uma característica de fácil interoperabilidade e portabilidade com demais ferramentas.

Desta maneira, conforme em (OETIKER, 2009), o RRDtool é uma ferramenta de banco de dados que foi desenvolvida objetivando suprir as necessidades para o conjunto de armazenamento de dados e geração de gráficos em determinado período. Esta ferramenta possibilita escrever sua própria rede personalizada, bem como rodar *scripts* de monitoramento de uma aplicação em determinado intervalo de tempo definido (como por exemplo a cada minuto) ou usar uma das muitas ferramentas pré-desenvolvidas baseadas em RRDtool (como por exemplo a ferramenta de monitamento CACTI, ou MRTG).

4. CONTRIBUIÇÕES

Sabe-se que proposta inicial do projeto era a de fazer a implementação de técnicas de gerenciamento da saída do UniscanWAF, neste ponto do estudo, delimita-se o escopo deste trabalho à tais aspectos de gerenciamento: **a)** Implementação de registros de *logs*; **b)** Geração de gráficos e, **c)** Análise com Consultas. Sabe-se que a ampla área de gerenciamento envolve um valor de complexidade progressivo, pois este conceito como viu-se anteriormente, dependendo do ambiente em que se têm e o nível de negligência do administrador, pode-se tornar um sistema massivamente vulnerável. No entanto, prevenir-se com o uso adequado de ferramentas é fundamental. Porém, quando envolve a *Internet*, todos estamos, de uma forma ou de outra, vulneráveis. Por esta questão ferramentas que protegem-nos destas ameaças são bem-vindas.

Os dados gerados por estas aplicações, geralmente carecem de métodos de facilitação de análise, como foi mostrado com as ferramentas selecionadas para análise destes requisitos. Porém, com este estudo pretende-se contribuir à solucionar estes tipos de problemas, ajudando, desta forma, os responsáveis por gerenciar estas aplicações e, contribuindo para se ter bons referenciais nas cinco subáreas de gerenciamento, também descritas no capítulo 2. Este trabalho, tem como contribuição a aplicação de duas destas subáreas, gerenciamento de segurança e gerência de contabilidade.

Este estudo objetiva fazer o tratamento da saída do *Firewall* de Aplicação *Web* denominado UniscanWAF. Para isto, este trabalho propõe implementar registros de *logs,* bem como fazer a análise e auditoria destes eventos registrados, fazendo as adequações e modificar o que for necessário, para o desenvolvimento desta nova funcionalidade, até então inexistente.

4.1. Implementação de *Logs*

Com o desenvolvimento dos arquivos de *logs* (agora com conhecimento prévio das vulnerabilidades descritas no *log*) pôde-se implementar os registros de todas as ocorrências de eventos, mostrados na saída do UniscanWAF, bem como, o contabilização de acontecimentos das detecções. Para isso, foram criados os arquivos "*LogFile.pm*" e "*LogGeral.pm*". Estes foram incluídos no núcleo do *Firewall* de Aplicação, sendo denominados como classes adicionais escritas em Perl. Cada uma, contendo uma sub-rotina para geração dos registros de *logs* característicos. Estas classes serão descritas à seguir. Seus respectivos códigos-fonte seguem nas Figuras 10 e 11.

```
1  package Uniscan::LogFile;
2
3  use strict;
4
5  sub Log() {
6     my $logdir="/var/log/";
7     my ($self,$s) = @_;
8     my ($logsec,$logmin,$loghour,$logmday,$logmon,$logyear,$logwday,$logyday,$logisdst)=localtime(time);
9     my $time_stamp = time();
10    my $logtimestamp = sprintf("[%4d-%02d-%02d %02d:%02d:%02d]",$logyear+1900,$logmon+1,$logmday,$loghour,
  $logmin,$logsec);
11    $logmon++;
12    my $logfile="$logdir$logmon-$logmday-AccessUniscanWAF.log";
13    my $fh;
14    open($fh, '>>', "$logfile") or die "$logfile: $!";
15    print $fh "$logtimestamp [Timestamp: $time_stamp] $s\n";
16    close($fh);
17 }
18
19 1;
```

Figura 10: Código-fonte da classe *LogFile.pm*.
Fonte: do Autor.

Esta classe, denominada *LogFile.pm*, é responsável pela implementação do registro de *logs*, onde são guardados todos os eventos e acontecimentos da ferramenta UniscanWAF. Em seu código-fonte o "*package Uniscan::LogFile;*" descrito na linha 1 corresponde ao nome da classe.

Na linha 3, o comando "*use strict;*" é responsável pela inclusão de uma biblioteca para debugar erros de código. Logo em seguida, na linha 5, é declarada a sub-rotina "*Log*" responsável pela geração dos registros de eventos. A variável "*Logdir*" é o diretório onde são salvos os arquivos de *logs*, no caso em /var/log.

Descrito na linha 8, tem-se a utilização do comando "*localtime(time)*" o qual

retorna o tempo no formato, em que as variáveis de atribuição estão criadas (por exemplo, o primeiro dado significa o tempo em segundos, depois minutos, e em seguida horas, pois as variáveis são: *logsec*, *logmin* e *loghour*, respectivamente). O comando "*time()*" é outro importante, pois é ele que retorna o tempo em *Timestamp*[3] (utilizado no RRDtool). A linha 12 contém a variável *Logfile,* esta receberá o nome em que será salvo o arquivo de log, neste caso, por padrão, é salvo um registro diário. E, por fim, as próximas 4 linhas são responsáveis por abrir o arquivo de *log* e adicionar os eventos que o UniscanWAF está monitorando em tempo real, isto é armazenar todas às requisições efetuadas ao servidor web "protegido".

Já na Figura 11, está o código-fonte do arquivo de *log* em que se faz o armazenamento do somatório geral dos tipos de vulnerabilidades detectadas pelo *Firewall* de Aplicação UniscanWAF, permitindo a contabilidade de vulnerabilidades encontradas diariamente com a aplicação rodando. Fazendo com que desta forma se possa ter um princípio de se trabalhar com o gerenciamento de contabilidade, podendo ser feita uma análise quantitativa de vulnerabilidades por exemplo. Contribuindo para o aprimorando da ferramenta.

Nomeada *LogGeral.pm*, esta classe, que é responsável pelo registro dos totais de vulnerabilidades encontradas diariamente, possui seu código-fonte bastante similar ao *Log* anteriormente descrito (*LogFile.pm*). As alterações encontram-se em seu nome "*package Uniscan::LogGeral;*" na linha 1.

Na linha 12 a variável *Logfile,* recebe o nome em que será salvo o arquivo de log, no mesmo sistema implantado no *Log* anterior, porém, alteração no nome do arquivo para o formato: Mês-Dia-GeralUniscanWAF.log (por exemplo, 2-1-GeralUniscanWAF.log). Da mesma maneira, segue sendo salvo um registro diário.

Por fim, da linha 13 até a linha 37 é efetuada a leitura do arquivo e o somatório das vulnerabilidades. Sendo que, primeiro da linha 13 à 22 é feito o teste se o arquivo já é existente, se não cria-o. A próxima etapa das linhas 23 até 34 é feita a leitura, sendo aberto o arquivo para verificar as vulnerabilidades atual já contabilizadas. E a última etapa é a escrita neste arquivo, ou seja, o somatório das vulnerabilidades detectadas, descrito nas linhas 35, 36 e 37.

---

3 *Timestamp* é uma forma de controlar o tempo como uma execução total de segundos. Esta contagem começa na Era Unix em 1 de janeiro de 1970. Portanto, o este formato de tempo é apenas o número de segundos entre uma determinada data e da Época Unix. (TIMESTAMP, 2013).

```
1  package Uniscan::LogGeral;
2
3  use strict;
4
5  sub Log() {
6     my $logdir="/var/log/";
7     my ($self,$s) = @_;
8     my ($logsec,$logmin,$loghour,$logmday,$logmon,$logyear,$logwday,$logyday,$logisdst)=localtime(time);
9     my $time_stamp = time();
10    my $logtimestamp = sprintf("[%4d-%02d-%02d %02d:%02d:%02d]",$logyear+1900,$logmon+1,$logmday,$loghour,
  $logmin,$logsec);
11    $logmon++;
12    my $logfile="$logdir$logmon-$logmday-GeralUniscanWAF.log";
13    my $linha;
14        my $arq_text = "";
15        my $temp;
16        if (!(-e $logfile)) {
17                $arq_text = "LFI:0\nRCE:0\nRFI:0\nSQL:0\nXSS:0";
18                open(R, '>', "$logfile") or die "$logfile: $!";
19                print R "$arq_text";
20                close(R);
21                $arq_text = "";
22        }
23    open(R, '<', "$logfile") or die "$logfile: $!";
24        $linha = 0;
25         my $s_temp = $s.":";
26         while (<R>) {
27           $linha++;
28           if ($_ =~ /$s/) {
29                $temp = substr ( $_, length($s_temp), (length($_)-length($s)) );
30                substr($_,length($s_temp),(length($_)-length($s))) = ($temp+1)."\n";
31           }
32           $arq_text .= $_;
33         }
34        close(R);
35    open(R, '>', "$logfile") or die "$logfile: $!";
36        print R "$arq_text";
37        close(R);
38 }
39
40 1;
```

Figura 11: Código-fonte da classe *LogGeral.pm*
Fonte: do Autor.

Findando esta seção, conclui-se parcialmente, que os *logs* implementados são ferramentas valiosas para análise de eventos e controle de gestão de vulnerabilidades em um determinado servidor *web.* Para melhor compreensão, exemplos dos dois tipos de *logs* criados serão mostrados no próximo capítulo, em Testes e Resultados.

4.2. Geração de Gráficos

Atribui-se ao módulo de inicialização, configurar ações após encontrar uma

vulnerabilidade. Como exemplos de ações tem-se: encerrar a conexão, colocar a máquina de origem em quarentena, gerar um relatório técnico, ou mesmo obter maiores informações com a execução de outras ferramentas que possibilite uma análise e/ou diagnóstico do UniscanWAF, bem como, a localização do arquivo de *log* (FABRO NETO; TURCHETTI, 2012). Estes três últimos exemplos de ações, abrangem justamente o cerne deste estudo.

O desenvolvimento dos gráficos foi, viabilizado pela utilização da ferramenta descrita anteriormente no capítulo 3 (o RDDtool), o qual gera gráficos, a partir de parâmetros e configurações. Com isto, tem-se uma melhor análise e visualização dos acontecimentos da aplicação, facilitando o trabalho do gerente de rede. Foram desenvolvidas cinco classes, sendo uma para cada tipo de vulnerabilidade, com sua própria base de bados do RRDtool. Sendo que, são gerados com estas classes, seis tipos de gráficos, os quais serão destritos a seguir. Cinco semelhantes, porém altera-se o tipo de vulnerabilidade contida no gráfico, e um gráfico geral englobando todos os tipos de vulnerabilidades detectadas pela ferramenta.

O primeiro tipo de gráfico destrito é o das vulnerabilidades do tipo LFI (*Local File Include*). Segue, na Figura 12, o código-fonte responsável pela geração dos gráficos deste tipo de vulnerabilidade. Tal arquivo foi nomeado *graphLFI.pm*.

```
1  package Uniscan::graphLFI;
2
3  use strict;
4  use RRDs;
5  use Uniscan::WAFScan;
6  use Uniscan::graphRCE;
7  use Uniscan::graphRFI;
8  use Uniscan::graphSQL;
9  use Uniscan::graphXSS;
10
11 sub Graph() {
12   my $graphdir = "/var/log/graphs/";
13   my ($self,$s) = @_;
14   my $db = "wimuLFI.rrd";
15   my $timestamp = time();
16   my ($logsec,$logmin,$loghour,$logmday,$logmon,$logyear,$logwday,$logyday,$logisdst)=localtime(time);
17   my $time = sprintf("[%4d-%02d-%02d_%02d:%02d:%02d]",$logyear+1900,$logmon+1,$logmday,$loghour,$logmin,
   $logsec);
18         $logmon++;
19   my $timei = time()-1;
20   my $timef = time()+1;
21   my $timec = $timei-1;
22
23   RRDs::create("$graphdir $db", "--start", $timec, "--step=1", "DS:lfi:GAUGE:1:U:U", "RRA:MAX:1:1:600",
   "RRA:MIN:0.01:2:1");
24   RRDs::update ("$graphdir $db", "$timei:0");
25   RRDs::update ("$graphdir $db", "$timestamp:1");
26   RRDs::update ("$graphdir $db", "$timef:0");
27
28   my $graphic = RRDs::graph("$graphdir $time LFI.png", "--start", $timestamp-600, "--end", $timestamp+600,
   "--step=1", "--title=Time $time Attempted to Web Server Attack\n$s", "--font=TITLE:7:Times", "--zoom=2",
   "DEF:vlfi=$graphdir $db:lfi:MAX", "DEF:vothers=$graphdir $db:lfi:MIN", "AREA:vlfi#FF0000:LFI",
   "LINE0:vothers#5553F3:RCE", "LINE0:vothers#00FF00:RFI", "LINE0:vothers#00FFFF:SQL",
   "LINE0:vothers#FFFF00:XSS");
29
30   my $dbg = "wimug.rrd";
31   my $graphdirg = "/var/log/graphs/";
32   RRDs::update ("$graphdirg$dbg", "$timei:0:0:0:0:0");
33   RRDs::update ("$graphdirg$dbg", "$timestamp:10:0:0:0:0");
34   RRDs::update ("$graphdirg$dbg", "$timef:0:0:0:0:0");
35 }
36
37 1;
```

Figura 12: Código-fonte da classe *graphLFI.pm*.
Fonte: do Autor.

Para interpretação deste código, na linha 1 é inserido o comando “*package Uniscan::graphLFI*” representando o nome da classe. Descrito na linha 3, o “*use strict;*” é responsável pela inclusão de uma biblioteca para debugar erros de código. Já na linha 4 é incluído no código o módulo para que se possa trabalhar com a ferramenta RRDtool na linguagem de programação Perl, o comando correspondente é “*use RRDs;*”. Logo em seguida, nas linhas de 5 à 9 são inseridas as classes à serem utilizadas pelo código-fonte.

Na linha 11, é declarada a sub-rotina “*Graph*” a qual é executada toda a vez que uma vulnerabilidade LFI é detectada, sendo responsável pela geração dos gráficos. Dentro deste sub-rotina, nas linhas 12 e 31, tem-se as variáveis “*graphdir*” e “*graphdirg*” onde são armazenados os diretórios de destino à serem salvos os gráficos, sendo a segunda variável a que guarda os gráficos que contém todas as

vulnerabilidades juntas. Neste caso o diretório correspondente para ambas é "*/var/log/graphs/*".

As variáveis "*self*" e "*s*", na linha 13, servem para guardar o nome da classe e o conteúdo do ataque encontrado na vulnerabilidade LFI, respectivamente. Já as variáveis "*db*" e "*dbg*", encontradas nas linhas 14 e 30, recebem os nomes das bases de dados do RRDtool, sendo que a primeira base de dados é referente aos gráficos somente da vulnerabilidade LFI. E a segunda correspondendo a base de dados geral, onde todos os tipos vulnerabilidades podem ser encontradas no mesmo gráfico.

Descritas nas linhas de 15 até a linha 21, encontram-se as variáveis referentes ao tempo, Onde na linha 15, é criada a variável "*timestamp*", em que a mesma recebe o valor do comando "*time()"*, fazendo com que esta variável resulte no tempo atual em *Timestamp,* já anteriormente explicado. Na linha 16 tem-se a utilização do comando "*localtime(time)*" o qual retorna o tempo no formato, em que as variáveis de atribuição estão criadas (por exemplo, o primeiro dado significa o tempo em segundos, depois minutos, e em seguida horas, pois as variáveis são: *logsec*, *logmin* e *loghour*, respectivamente). E a variável "*time*", na linha 17, recebe o tempo no formato [Ano-Mês-Dia_Hora:Minuto:Segundo], como por exemplo [2012-02-02_12:30:34]. Por fim, nas linhas 19, 20 e 21, tem-se a criação de variáveis utilizadas para atualização e criação dos gráficos.

A partir deste ponto, são criados os comandos exclusivos para a geração da base de dados, expresso na linha 23. Atualização desta base (inserção de dados) nas linhas seguintes 24, 25 e 26. E por fim a criação do gráfico, descrito na linha 28. Tais procedimentos acima são referentes à geração dos gráficos para somente contendo a vulnerabilidade LFI. Para possibilitar-se a execução destes comandos (na sintaxe do RRDtool) utilizando a linguagem de programação Perl deve ser instalado um módulo de integração do RRDtool com o Perl. Após instalado, deve-se habilitar tal funcionalidade no código-fonte para se permitir a execução dos comandos da ferramenta RRDtool. Tal procedimento foi efetuado na linha 4.

A partir da linha 30, são descritos os procedimentos para a geração do gráfico geral, o qual engloba vulnerabilidades, nota-se que este trecho de código está contido nas cinco classes de criação de gráficos existentes. É claro com adaptações

correspondentes para cada tipo de vulnerabilidade. Desta forma, as linhas 30 e 31 contêm o nome da base de dados e, o local onde são guardados estes gráficos, tais variáveis, por se tratar do gráfico geral, são as mesmas para todos os tipos de vulnerabilidades.

Encerrando tal classe temos a atualização da base de dados do gráfico geral com os comandos correspondentes nas linhas 32, 33 e 34. Visto que a é utilizada a mesma base de dados para todas as vulnerabilidades, a mesma é criada em no módulo de inicialização do UniscanWAF, onde é reconhecida por todas as demais vulnerabilidades, sendo assim, esta base é criada neste arquivo principal e somente alimentada (atualizada com a inserção de dados) nas classes correspondentes as vulnerabilidades. Ressalta-se também, que tal gráfico é gerado somente neste módulo de inicialização, contendo já, os dados inseridos pelas classes.

A seguir na Figura 13 é demonstrado o código-fonte do segundo tipo de gráfico, sendo este correspondente à geração dos gráficos de vulnerabilidades do tipo RCE (*Remote Command Execution*). Tal classe foi denominada *graphRCE.pm*.

```
1  package Uniscan::graphRCE;
2
3  use strict;
4  use RRDs;
5  use Uniscan::WAFScan;
6  use Uniscan::graphLFI;
7  use Uniscan::graphRFI;
8  use Uniscan::graphSQL;
9  use Uniscan::graphXSS;
10
11 sub Graph() {
12    my $graphdir = "/var/log/graphs/";
13    my ($self,$s) = @_;
14    my $db = "wimuRCE.rrd";
15    my $timestamp = time();
16    my ($logsec,$logmin,$loghour,$logmday,$logmon,$logyear,$logwday,$logyday,$logisdst)=localtime(time);
17    my $time = sprintf("[%4d-%02d-%02d_%02d:%02d:%02d]",$logyear+1900,$logmon+1,$logmday,$loghour,$logmin,
   $logsec);
18          $logmon++;
19    my $timei = time()-1;
20    my $timef = time()+1;
21    my $timec = $timei-1;
22
23    RRDs::create("$graphdir $db", "--start", $timec, "--step=1", "DS:rce:GAUGE:1:U:U", "RRA:MAX:1:1:600",
   "RRA:MIN:0.01:2:1");
24    RRDs::update ("$graphdir $db", "$timei:0");
25    RRDs::update ("$graphdir $db", "$timestamp:1");
26    RRDs::update ("$graphdir $db", "$timef:0");
27
28    my $graphic = RRDs::graph("$graphdir $time RCE.png", "--start", $timestamp-600, "--end", $timestamp+600,
   "--step=1", "--title=Time $time Attempted to Web Server Attack\n$s", "--font=TITLE:7:Times", "--zoom=2",
   "DEF:vrce=$graphdir $db:rce:MAX", "DEF:vothers=$graphdir $db:rce:MIN", "LINE0:vothers#FF0000:LFI",
   "AREA:vrce#5553F3:RCE", "LINE0:vothers#00FF00:RFI", "LINE0:vothers#00FFFF:SQL", "LINE0:vothers#FFFF00:XSS");
29
30    my $dbg = "wimug.rrd";
31    my $graphdirg = "/var/log/graphs/";
32    RRDs::update ("$graphdirg$dbg", "$timei:0:0:0:0:0");
33    RRDs::update ("$graphdirg$dbg", "$timestamp:0:20:0:0:0");
34    RRDs::update ("$graphdirg$dbg", "$timef:0:0:0:0:0");
35 }
36
37 1;
```

Figura 13: Código-fonte da classe *graphRCE.pm*.
Fonte: do Autor.

Nesta classe, na linha 1 é inserido o comando “*package Uniscan::graphRCE*” representando o nome da classe. Da linha 3 à 5 todas as classes se equivalem quanto a linhas de códigos. Já da linha 6 até a linha 21, o que se tem é que são códigos bastante similares para todas as classes. As mudanças são: **a)** Tem-se a substituição da classe utilizada no comando “*use*”, como por exemplo, a classe da vulnerabilidade LFI deve utilizar todas as demais classes e a vulnerabilidade RCE também deve utilizar todas as classes restantes dentre as cinco e **b)** Na linha 14 altera-se o nome da base de dados, pois são bases individuais para cada tipo de vulnerabilidade, neste caso a base muda para “*wimuRCE.rrd*”.

Novamente a partir da linha 23, são criados os comandos exclusivos para a geração dos gráficos, sendo na linha 23 a criação da base de dados individual. A atualização desta base de dados (inserção de dados) nas linhas seguintes 24, 25 e 26. E por fim a criação do gráfico, descrito na linha 28. Tais procedimentos acima

são referentes à geração dos gráficos somente para a vulnerabilidade do tipo RCE.

A partir da linha 30, são descritos os procedimentos para a geração do gráfico geral, o qual abrange todas as vulnerabilidades, sabe-se que este trecho de código está contido nas cinco classes de criação de gráficos existentes. Contendo as adaptações correspondentes para cada tipo de vulnerabilidade. Desta forma, nas linhas 30 e 31 mantem-se o mesmo nome da base de dados e diretório onde são armazenados estes gráficos. Por fim, temos a atualização da base de dados do gráfico geral com os comandos correspondentes nas linhas 32, 33 e 34.

Logo, na Figura 14 é demonstrado o código-fonte do terceiro tipo de gráfico, sendo este correspondente à geração dos gráficos de vulnerabilidades do tipo RFI (*Remote File Include*). Tal classe foi nomeada *graphRFI.pm*.

```
1  package Uniscan::graphRFI;
2
3  use strict;
4  use RRDs;
5  use Uniscan::WAFScan;
6  use Uniscan::graphLFI;
7  use Uniscan::graphRCE;
8  use Uniscan::graphSQL;
9  use Uniscan::graphXSS;
10
11 sub Graph() {
12    my $graphdir = "/var/log/graphs/";
13    my ($self,$s) = @_;
14    my $db = "wimuRFI.rrd";
15    my $timestamp = time();
16    my ($logsec,$logmin,$loghour,$logmday,$logmon,$logyear,$logwday,$logyday,$logisdst)=localtime(time);
17    my $time = sprintf("[%4d-%02d-%02d_%02d:%02d:%02d]",$logyear+1900,$logmon+1,$logmday,$loghour,$logmin,
   $logsec);
18          $logmon++;
19    my $timei = time()-1;
20    my $timef = time()+1;
21    my $timec = $timei-1;
22
23    RRDs::create("$graphdir $db", "--start", $timec, "--step=1", "DS:rfi:GAUGE:1:U:U", "RRA:MAX:1:1:600",
   "RRA:MIN:0.01:2:1");
24    RRDs::update ("$graphdir $db", "$timei:0");
25    RRDs::update ("$graphdir $db", "$timestamp:1");
26    RRDs::update ("$graphdir $db", "$timef:0");
27
28    my $graphic = RRDs::graph("$graphdir $time RFI.png", "--start", $timestamp-600, "--end", $timestamp+600,
   "--step=1", "--title=Time $time Attempted to Web Server Attack\n$s", "--font=TITLE:7:Times", "--zoom=2",
   "DEF:vrfi=$graphdir $db:rfi:MAX", "DEF:vothers=$graphdir $db:rfi:MIN", "LINE0:vothers#FF0000:LFI",
   "LINE0:vothers#5553F3:RCE", "AREA:vrfi#00FF00:RFI", "LINE0:vothers#00FFFF:SQL", "LINE0:vothers#FFFF00:XSS");
29
30    my $dbg = "wimug.rrd";
31    my $graphdirg = "/var/log/graphs/";
32    RRDs::update ("$graphdirg$dbg", "$timei:0:0:0:0:0");
33    RRDs::update ("$graphdirg$dbg", "$timestamp:0:0:30:0:0");
34    RRDs::update ("$graphdirg$dbg", "$timef:0:0:0:0:0");
35 }
36
37 1;
```

Figura 14: Código-fonte da classe *graphRFI.pm*
Fonte: do Autor.

Na classe para geração dos gráficos de vulnerabilidades do tipo RFI, na linha 1 é inserido o comando “*package Uniscan::graphRFI*” representando o nome da classe. Conforme dito anteriormente, as linhas 3, 4 e 5 são iguais as classes já descritas. E no intervalo da linha 6 à linha 21 as mudanças foram a adequação das classes utilizadas, como já exemplificado e, a alteração do nome da base de dados para “*wimuRFI.rrd*” na linha 14.

A partir da linha 23, são efetuados os comandos necessários para a geração dos gráficos, nesta linha é criada a base de dados individual. A atualização desta base de dados (inserção de dados) nas linhas seguintes 24, 25 e 26. E por fim a criação do gráfico, descrito na linha 28. Tais procedimentos acima são referentes à geração dos gráficos somente para a vulnerabilidade do tipo RCE.

Novamente, os comandos, a partir da linha 30, fazem com que sejam feitos os procedimentos para a geração do gráfico geral, o qual abrange todas as vulnerabilidades, ressalta-se que este trecho de código está contido nas cinco classes de criação de gráficos existentes. Contendo as adequações correspondentes para cada tipo de vulnerabilidade. Desta forma, nas linhas 30 e 31 mantem-se o mesmo nome da base de dados e diretório onde são armazenados estes gráficos. Por fim, temos a atualização da base de dados do gráfico geral com os comandos correspondentes nas linhas 32, 33 e 34.

A seguir nas Figuras 15 e 16 são demonstrados os códigos-fonte do quarto e quinto tipos de gráficos, sendo estes correspondentes à geração dos gráficos de vulnerabilidades do tipo SQL (*Structured Query Language*) e do tipo XSS (*Cross Site Scripting*), respectivamente. Tais classes foram denominadas *graphSQL.pm* e *graphXSS.pm*.

```
1  package Uniscan::graphSQL;
2
3  use strict;
4  use RRDs;
5  use Uniscan::WAFScan;
6  use Uniscan::graphLFI;
7  use Uniscan::graphRCE;
8  use Uniscan::graphRFI;
9  use Uniscan::graphXSS;
10
11 sub Graph() {
12    my $graphdir = "/var/log/graphs/";
13    my ($self,$s) = @_;
14    my $db = "wimuSQL.rrd";
15    my $timestamp = time();
16    my ($logsec,$logmin,$loghour,$logmday,$logmon,$logyear,$logwday,$logyday,$logisdst)=localtime(time);
17    my $time = sprintf("[%4d-%02d-%02d_%02d:%02d:%02d]",$logyear+1900,$logmon+1,$logmday,$loghour,$logmin,
   $logsec);
18          $logmon++;
19    my $timei = time()-1;
20    my $timef = time()+1;
21    my $timec = $timei-1;
22
23    RRDs::create("$graphdir $db", "--start", $timec, "--step=1", "DS:sql:GAUGE:1:U:U", "RRA:MAX:1:1:600",
   "RRA:MIN:0.01:2:1");
24    RRDs::update ("$graphdir $db", "$timei:0");
25    RRDs::update ("$graphdir $db", "$timestamp:1");
26    RRDs::update ("$graphdir $db", "$timef:0");
27
28    my $graphic = RRDs::graph("$graphdir $time SQL.png", "--start", $timestamp-600, "--end", $timestamp+600,
   "--step=1", "--title=Time $time Attempted to Web Server Attack\n$s", "--font=TITLE:6:Times", "--zoom=2",
   "DEF:vsql=$graphdir $db:sql:MAX", "DEF:vothers=$graphdir $db:sql:MIN", "LINE0:vothers#FF0000:LFI",
   "LINE0:vothers#5553F3:RCE", "LINE0:vothers#00FF00:RFI", "AREA:vsql#00FFFF:SQL", "LINE0:vothers#FFFF00:XSS");
29
30    my $dbg = "wimug.rrd";
31    my $graphdirg = "/var/log/graphs/";
32    RRDs::update ("$graphdirg$dbg", "$timei:0:0:0:0:0");
33    RRDs::update ("$graphdirg$dbg", "$timestamp:0:0:0:40:0");
34    RRDs::update ("$graphdirg$dbg", "$timef:0:0:0:0:0");
35 }
36
37 1;
```

Figura 15: Código-fonte da classe *graphSQL.pm*
Fonte: do Autor.

Em que pode-se observar novamente as semelhanças anteriormente descritas e possuindo as mesmas mudanças: **a)** Do nome da classe na linha 1 para "*package Uniscan::graphSQL*", **b**) Das adequações das classes utilizadas, como já exemplificado e, **c)** A alteração do nome da base de dados para "*wimuSQL.rrd*" na linha 14.

Quanto as atualizações no gráfico geral, que contêm todas as vulnerabilidades na mesma base de dados, os procedimentos também seguiram os mesmos padrões já vistos.

```
1  package Uniscan::graphXSS;
2
3  use strict;
4  use RRDs;
5  use Uniscan::WAFScan;
6  use Uniscan::graphLFI;
7  use Uniscan::graphRCE;
8  use Uniscan::graphRFI;
9  use Uniscan::graphSQL;
10
11 sub Graph() {
12    my $graphdir = "/var/log/graphs/";
13    my ($self,$s) = @_;
14    my $db = "wimuXSS.rrd";
15    my $timestamp = time();
16    my ($logsec,$logmin,$loghour,$logmday,$logmon,$logyear,$logwday,$logyday,$logisdst)=localtime(time);
17    my $time = sprintf("[%4d-%02d-%02d_%02d:%02d:%02d]",$logyear+1900,$logmon+1,$logmday,$loghour,$logmin,
   $logsec);
18          $logmon++;
19    my $timei = time()-1;
20    my $timef = time()+1;
21    my $timec = $timei-1;
22
23    RRDs::create("$graphdir $db", "--start", $timec, "--step=1", "DS:xss:GAUGE:1:U:U", "RRA:MAX:1:1:600",
   "RRA:MIN:0.01:2:1");
24    RRDs::update ("$graphdir $db", "$timei:0");
25    RRDs::update ("$graphdir $db", "$timestamp:1");
26    RRDs::update ("$graphdir $db", "$timef:0");
27
28    my $graphic = RRDs::graph("$graphdir $time XSS.png", "--start", $timestamp-600, "--end", $timestamp+600,
   "--step=1", "--title=Time $time Attempted to Web Server Attack\n$s", "--font=TITLE:7:Times", "--zoom=2",
   "DEF:vxss=$graphdir $db:xss:MAX", "DEF:vothers=$graphdir $db:xss:MIN", "LINE0:vothers#5553F3:RCE",
   "LINE0:vothers#00FF00:RFI", "LINE0:vothers#FF0000:LFI", "LINE0:vothers#00FFFF:SQL", "AREA:vxss#FFFF00:XSS");
29
30    my $dbg = "wimug.rrd";
31    my $graphdirg = "/var/log/graphs/";
32    RRDs::update ("$graphdirg$dbg", "$timei:0:0:0:0:0");
33    RRDs::update ("$graphdirg$dbg", "$timestamp:0:0:0:0:50");
34    RRDs::update ("$graphdirg$dbg", "$timef:0:0:0:0:0");
35 }
36
37 1;
```

Figura 16: Código-fonte da classe *graphXSS.pm*
Fonte: do Autor.

Findando as classes implementadas, tem-se a quinta e última, responsável pela geração dos gráficos de vulnerabilidades do tipo XSS. Esta procede com o mesma modelagem utilizada pelas classes anteriores, bem como, as mesmas alterações. Isto é, na linha 1 muda-se o nome da classe para "*package Uniscan::graphXSS*". Nas linhas entre 4 e 10, mantem-se as mesmas adequações das classes utilizadas. E por fim, altera-se o nome da base de dados para "*wimuXSS.rrd*" na linha 14.

Para as atualizações (inserção de dados) no gráfico geral, o qual contêm todas as vulnerabilidades em uma mesma base de dados (*wimug.rrd*), os procedimentos mantiveram-se normalmente.

Na Figura 17, tem-se os comandos efetuados para criação dos gráficos gerais, estes contendo uma base de dados com todos os tipos de vulnerabilidades. A geração destes gráficos, por se tratar de uma base de dados com múltiplas fontes

de dados (*Data-Sources*), necessitou ser implementada em um local onde pudesse haver uma integração entre as diferentes classes criadas. Por este motivo, tal procedimento foi efetuado a partir de adições de códigos-fonte tanto no núcleo do sistema, quanto no módulo de inicialização do UniscanWAF, bem como em cada um dos *plug-ins* para detecção de vulnerabilidades. Como já citados tais trechos de códigos para atualização (inserção de dados) deste gráfico geral.

```
1  package Uniscan::WAFScan;
2
3  use strict;
4  use Moose;
5  use Net::Socket::NonBlock;
6  use URI::Escape;
7  use Uniscan::Factory;
8  use XML::Parser;
9  use XML::SimpleObject;
10 use Data::Dumper;
11 use DBI;
12 use Uniscan::LogFile;
13 use Uniscan::graphLFI;
14 use Uniscan::graphRFI;
15 use Uniscan::graphRCE;
16 use Uniscan::graphXSS;
17 use Uniscan::graphSQL;
18 use RRDs;
19
20 $SIG{INT}=\&graph;
21
22 sub graph {
23   my $dbg = "wimug.rrd";
24   my $graphdirg = "/var/log/graphs/";
25   print "\nGraph Generation on $graphdirg\n";
26   my $timestamp = time();
27   my ($logsec,$logmin,$loghour,$logmday,$logmon,$logyear,$logwday,$logyday,$logisdst)=localtime(time);
28   my $time = sprintf("[%4d-%02d-%02d_%02d:%02d:%02d]",$logyear+1900,$logmon+1,$logmday,$loghour,$logmin,
   $logsec);
29        $logmon++;
30
31   RRDs::graph("$graphdirg$time-geral.png", "--start", $timestamp-150, "--end", $timestamp+150, "--step=1",
   "--title=Attempted to Web Server Attack ~ Time $time", "--zoom=2", "--vertical-label=Vulnerabilities Types",
   "DEF:vulnlfi=$graphdirg$dbg:glfi:MAX", "DEF:vulnrce=$graphdirg$dbg:grce:MAX", "DEF:vulnrfi=$graphdirg
   $dbg:grfi:MAX", "DEF:vulnsql=$graphdirg$dbg:gsql:MAX", "DEF:vulnxss=$graphdirg$dbg:gxss:MAX",
   "AREA:vulnlfi#FF0000:[10]LFI", "AREA:vulnrce#0F00F0:[20]RCE", "AREA:vulnrfi#00FF00:[30]RFI",
   "AREA:vulnsql#00FFFF:[40]SQL", "AREA:vulnxss#FFFF00:[50]XSS");
32
33   exit (0);
34 }
35
36   my $graphdirg = "/var/log/graphs/";
37   if (!(-e $graphdirg)) {
38     my $graphdirg = mkdir("/var/log/graphs",0755) || die "Error in creating directory: $!";
39   }
40   my $timei = time()-2;
41   my $dbg = "wimug.rrd";
42   my $graphdirg = "/var/log/graphs/";
43   my $geraldb = RRDs::create("$graphdirg$dbg", "--start", $timei, "--step=1", "DS:glfi:GAUGE:1:0:U",
   "DS:grce:GAUGE:1:0:U", "DS:grfi:GAUGE:1:0:U", "DS:gsql:GAUGE:1:0:U", "DS:gxss:GAUGE:1:0:U",
   "RRA:MAX:0.99:1:86400");
```

Figura 17: Implementação dos gráficos gerais no módulo de inicialização do UniscanWAF
Fonte: do Autor.

A ilustração da Figura 17, demostra os comandos necessários para se efetuar a criação da base de dados do gráfico geral, bem como, a geração deste gráfico contendo estas informações. Desta forma, nas linhas de 12 à 17 são inseridas as classes novas criadas as quais serão utilizadas pelo módulo de inicialização. Na linha 18, é adicionado o módulo RRDs, responsável por possibilitar a interpretação dos comandos da ferramenta RRDtool.

Na linha 20, é inserido o comando "*SIG{INT}=\&graph*", o qual possibilita se fazer o tratamento da interrupção (CTRL+C) na execução do monitoramento do *Firewall* de Aplicação. Tal comando foi essencial pela necessidade que se tem de definir um início e um fim de um gráfico, pois desta maneira a base geral de dados fica atualizando até o momento da interrupção. Onde é invocada a sub-rotina "*graph*", a qual tem por finalidade fazer a geração de tais gráficos.

Dentro da sub-rotina tem-se então, nas linhas 23 e 24, o nome da base de dados (*wimug.rrd*) e o diretório onde serão salvos os gráficos, respectivamente. Em seguida, na linha 25 é mostrado na tela, no momento da interrupção o local onde são salvos os gráficos.

Logo, no intervalo das linhas 26 à 29 são criadas as variáveis de tempo. Sendo, na linha 26, a variável "*timestamp*", a qual recebe o comando "*time*();" resultando no tempo atual em timestamp. Na linha 27, é efetuado o comando "*localtime(time);*", que retorna o tempo no formato, em que as variáveis de atribuição estão criadas (por exemplo, o primeiro dado significa o tempo em segundos, depois minutos, e em seguida horas, pois as variáveis são: *logsec*, *logmin* e *loghour*, respectivamente). Já na linha 28, a variável "*time*" recebe o tempo no formato [Ano-Mês-Dia_Hora:Minuto:Segundo], como por exemplo [2012-02-02_12:30:34]. Por fim, na linha 31, é efetuado o comando para a geração do gráfico e na linha 33 o comando "*exit (0);*" encerra a execução do UniscanWAF.

Das linhas 36 até 39, é declarado o diretório onde são armazenados a base de dados e seus gráficos, o diretório correspondente é "*/var/log/graphs/*". Onde tem-se a verificação se o mesmo já existe no sistema (por padrão inexistente). Ao verificar sua ausência é feito a criação do mesmo na linha 38. Para tal finalidade é necessário que o UniscanWAF seja executado como super-usuário (para permitir criar o diretório no endereço /var/log/), ao menos pela primeira vez de execução.

Na linha 41 foi criada a variável “*dbg*”, onde é definido o nome da base de dados (*wimug.rrd*). Concluindo, na linha 43, é efetuado o comando para a criação da base geral de dados.

A biblioteca RRDtool aliado à linguagem de programação Perl, mostraram-se ferramentas bastante eficazes para o desenvolvimento dos gráficos, sendo que estes, facilitam a interpretação dos acontecimentos da aplicação, sendo mais fácil visualizar um gráfico do que vasculhar arquivos de logs em busca de informações.

5. TESTES E RESULTADOS

Neste capítulo são mostradas duas seções. Primeiramente, na seção 6.1 serão descritas as configurações de ambiente utilizadas. E, em seguida, na seção 6.2 serão mostrados os resultados obtidos com a proposta deste trabalho. Sintetizando, como resultados pode-se citar a implementação de *logs*, para tal funcionalidade, são adicionadas classes no núcleo do WAF, onde as mesmas são invocadas em cada um dos *plug-ins* de vulnerabilidades presentes na arquitetura do UniscanWAF. Desta forma, é criado um arquivo de *Log* contendo todos os eventos detectados pela ferramenta.

5.1. Ambiente de Testes

Nesta seção serão abordados os aspectos quanto as descrições dos equipamentos utilizados, bem como o ambiente para as coletas das informações para o estudo, onde serão feitas as análises e gerência das vulnerabilidades *web* detectadas pelo *firewall* de aplicação.

A priori, para a obtenção deste estudo, serão utilizadas duas máquinas. Um *Notebook* modelo i36, com processador Intel Pentium Dual-Core, 3GB de memória RAM DDR2, HD 160GB, interface de rede Wireless Broadcom 802.11, sistema operacional Linux Ubuntu 12.04 LTS 32 *bits*. E um microcomputador AMD Phenon II X2, 3.00 GHz, 2 GB RAM, com interface de rede *Wireless* Encore ENLWI-G2 (RTL8185), sistema operacional Linux Ubuntu 10.04 LTS 32 *bits*.

O *Notebook* será utilizado para hospedar as aplicações de monitoramento (UniscanWAF) e geração de gráficos (RRDtool), bem como o servidor *Web* Apache/2.2.22 (*Apache Hypertext Transfer Protocol Server*), enquanto que o *Desktop* será utilizado para solicitar as requisições da aplicação *web* hospedada no *Notebook*, isto é, desta máquina partirão as possíveis vulnerabilidades *web,* à serem detectadas no cenário criado. É importante ressaltar que, para aplicação dos testes

também foi utilizada uma máquina virtual, com sistema operacional Windows XP SP3 32 *bits*, disponível no *Notebook.*

Quanto aos tipos de redes, foram utilizadas a rede cabeada do CTISM (Intranet) da Universidade Federal de Santa Maria, no prédio do curso de Redes de Computadores. Bem como as redes sem fio denominadas CTISM_PUBLICA e CTISM_CORPORATIVA, sendo esta segunda protegida por senha. E também utilizou-se uma rede doméstica residencial.

5.2. Resultados Obtidos

Com as classes desenvolvidas em Perl, bem como as adições de códigos efetuadas na estrutura da ferramenta UniscanWAF, tanto no núcleo da ferramenta, bem como no módulo de inicialização, e também passando pelos *plug-ins* para detecção de vulnerabilidades, puderam-se obter resultados.

Após testes de funcionalidades para verificar se os códigos estavam fazendo o que deveriam. Obteve-se a implementação de dois tipos de registros de *logs* e a geração de seis tipos de gráficos. Os dois tipos de registros de *logs* gerados são ilustrados nas Figuras 18 e 19.

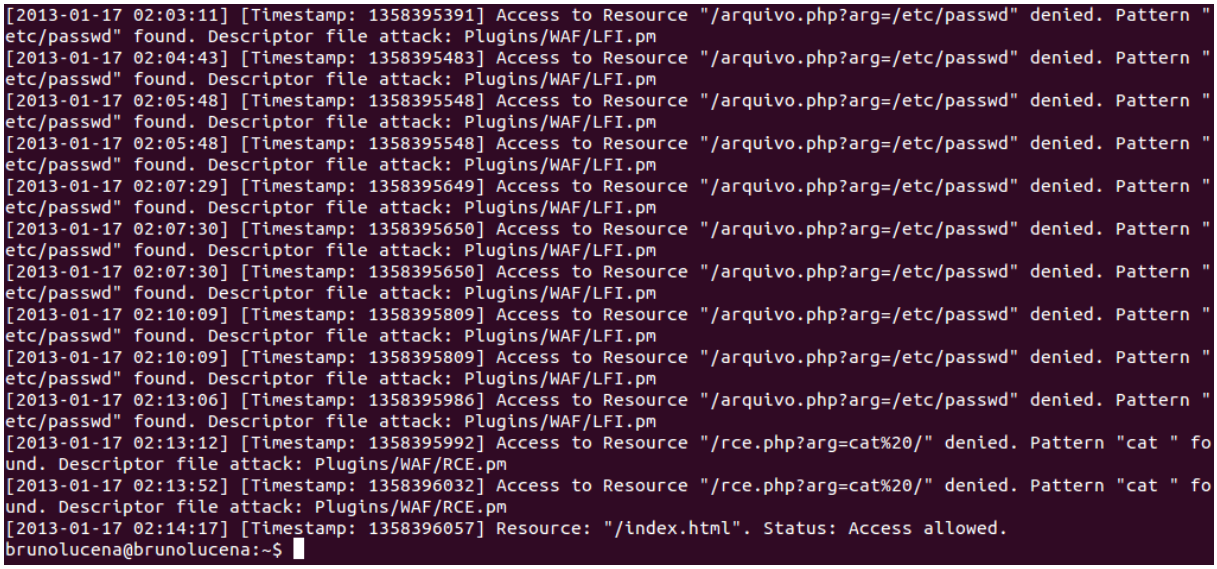


Figura 18: Registro de *Log* implementado com a classe *LogFile.pm*.
Fonte: do Autor.

Sendo este tipo de *log,* ilustrado na Figura 18*,* responsável por registrar todos os eventos e acontecimentos do servidor *web*, isto é, neste arquivo estão guardadas todas as requisições efetuadas à aplicação *web* *“*protegida*”* pelo *Firewall* de Aplicação UniscanWAF. A partir das informações contidas nestes *logs* que são gerados os seis tipos de gráficos existentes, os quais serão mostrados à seguir.

Também são gerados *logs* gerais contendo a quantidade de vulnerabilidades detectadas pela aplicação diariamente, este tipo de *log* como já informado anteriormente, é um somatório quantitativo do total de vulnerabilidades diárias.

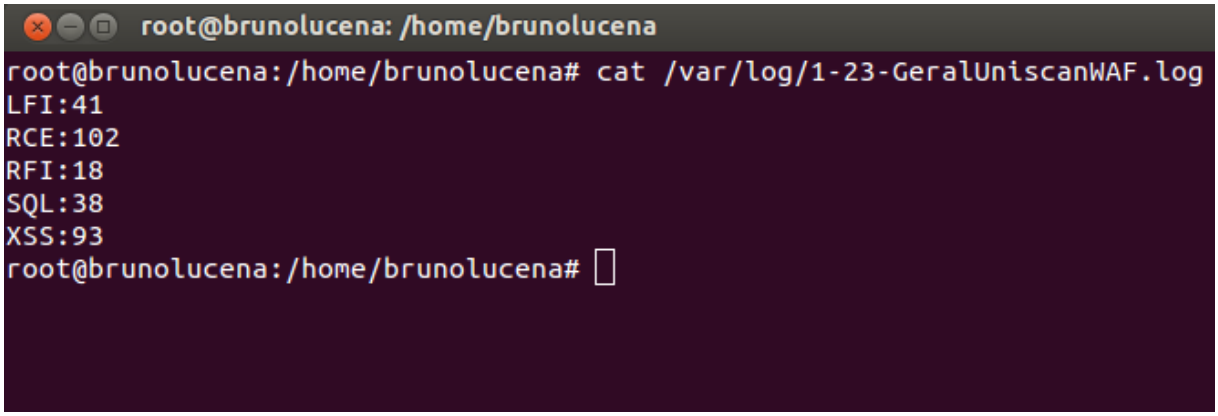


Figura 19: Registro de *Log* implementado com a classe *LogGeral.pm*.
Fonte: do Autor.

Destes *logs,* no entanto, não foram gerados gráficos, pelo fato de a visualização do mesmo apresentar-se de forma simples e agradável analisando diretamente o arquivo de *log*.

Tendo como resultados do *log* ilustrado na Figura 18, obteve-se a geração dos gráficos mostrados nas seguintes ilustrações: a Figura 20 contém o gráfico representando a vulnerabilidade LFI e a Figura 21 mostra o gráfico da vulnerabilidade RCE.

Interpretam-se os gráficos, com a leitura do título, no qual é descrito o tempo exato do registrado do último ataque, o recurso requisitado, e o ataque encontrado. No eixo y, o valor 1.0 representa que ocorreu a tentativa do ataque do tipo de vulnerabilidade referenciado na legenda e o 0.0 não representa a detecção do ataque.

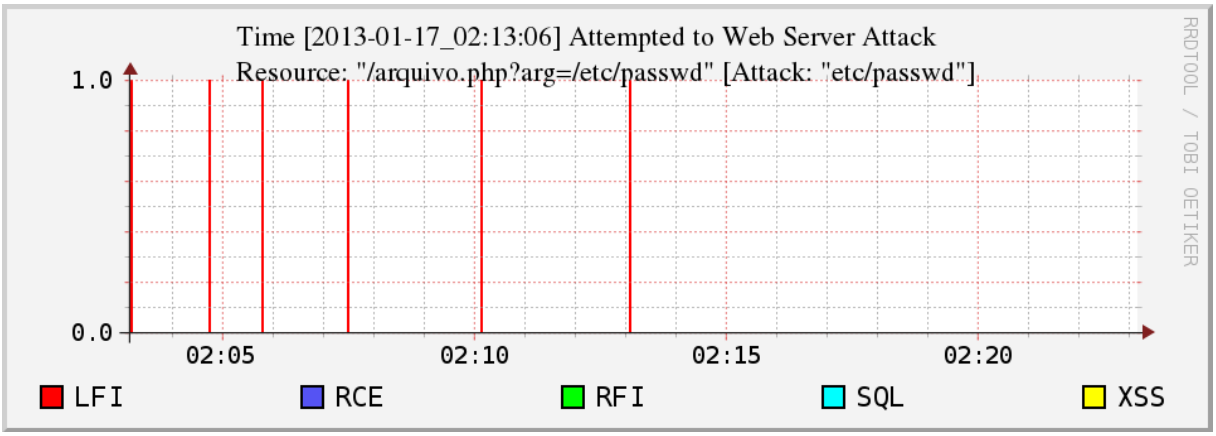


Figura 20: Gráfico da vulnerabilidade LFI gerado com os registros do *log*.
Fonte: do Autor.

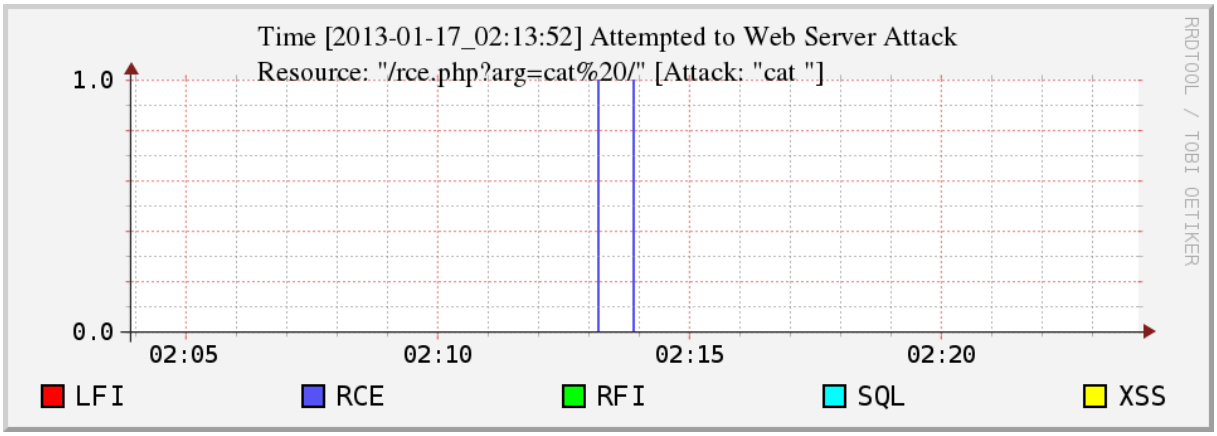


Figura 21: Gráfico da vulnerabilidade RCE gerado com os registros do *log*.
Fonte: do Autor.

A seguir na Figura 22, é mostrado o registro de *log* contendo as vulnerabilidades dos tipos RFI, SQL e XSS. Onde tais gráficos gerados são demonstrados nas Figuras 23, 24 e 25, para as vulnerabilidades dos tipos RFI, SQL e XSS, respectivamente. Tendo suas interpretações equivalentes aos gráficos demonstrados nas figuras anteriores.

```
[2013-02-03 22:18:21] [Timestamp: 1359937101] Access to Resource "/insecured.php?secret_file=/etc/profile?http://
192.168.2.3" denied. Pattern "http://" found. Descriptor file attack: Plugins/WAF/RFI.pm
[2013-02-03 22:18:37] [Timestamp: 1359937117] Access to Resource "/login.php?username=1'%20OR%20'1&password=1'%20
OR%20'1" denied. Pattern "OR " found. Descriptor file attack: Plugins/WAF/SQL.pm
[2013-02-03 22:18:47] [Timestamp: 1359937127] Access to Resource "/index.php?\%3CScRiPt%3Eprompt()%3C/ScRiPt%3E"
denied. Pattern "<ScRiPt>prompt(" found. Descriptor file attack: Plugins/WAF/XSS.pm
[2013-02-03 22:20:56] [Timestamp: 1359937256] Access to Resource "/insecured.php?secret_file=/etc/profile?http://
192.168.2.3" denied. Pattern "http://" found. Descriptor file attack: Plugins/WAF/RFI.pm
[2013-02-03 22:21:01] [Timestamp: 1359937261] Access to Resource "/login.php?username=1'%20OR%20'1&password=1'%20
OR%20'1" denied. Pattern "OR " found. Descriptor file attack: Plugins/WAF/SQL.pm
[2013-02-03 22:21:05] [Timestamp: 1359937265] Access to Resource "/index.php?\%3CScRiPt%3Eprompt()%3C/ScRiPt%3E"
denied. Pattern "<ScRiPt>prompt(" found. Descriptor file attack: Plugins/WAF/XSS.pm
[2013-02-03 22:21:11] [Timestamp: 1359937271] Access to Resource "/insecured.php?secret_file=/etc/profile?http://
192.168.2.3" denied. Pattern "http://" found. Descriptor file attack: Plugins/WAF/RFI.pm
[2013-02-03 22:21:29] [Timestamp: 1359937289] Access to Resource "/login.php?username=1'%20OR%20'1&password=1'%20
OR%20'1" denied. Pattern "OR " found. Descriptor file attack: Plugins/WAF/SQL.pm
[2013-02-03 22:21:48] [Timestamp: 1359937308] Access to Resource "/index.php?\%3CScRiPt%3Eprompt()%3C/ScRiPt%3E"
denied. Pattern "<ScRiPt>prompt(" found. Descriptor file attack: Plugins/WAF/XSS.pm
[2013-02-03 22:21:55] [Timestamp: 1359937315] Access to Resource "/login.php?username=1'%20OR%20'1&password=1'%20
OR%20'1" denied. Pattern "OR " found. Descriptor file attack: Plugins/WAF/SQL.pm
[2013-02-03 22:21:58] [Timestamp: 1359937318] Access to Resource "/index.php?\%3CScRiPt%3Eprompt()%3C/ScRiPt%3E"
denied. Pattern "<ScRiPt>prompt(" found. Descriptor file attack: Plugins/WAF/XSS.pm
[2013-02-03 22:22:17] [Timestamp: 1359937337] Access to Resource "/index.php?\%3CScRiPt%3Eprompt()%3C/ScRiPt%3E"
denied. Pattern "<ScRiPt>prompt(" found. Descriptor file attack: Plugins/WAF/XSS.pm
brunolucena@brunolucena:~$
```

Figura 22: *Log* contendo as vulnerabilidades dos tipos RFI, SQL e XSS.
Fonte: do Autor.

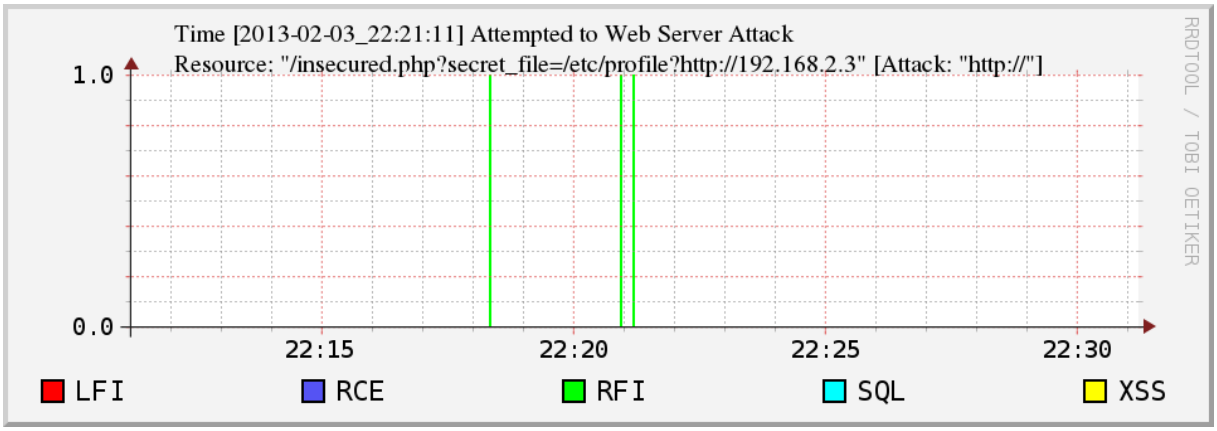


Figura 23: Gráfico da vulnerabilidade RFI gerado com os registros do *log*.
Fonte: do Autor.

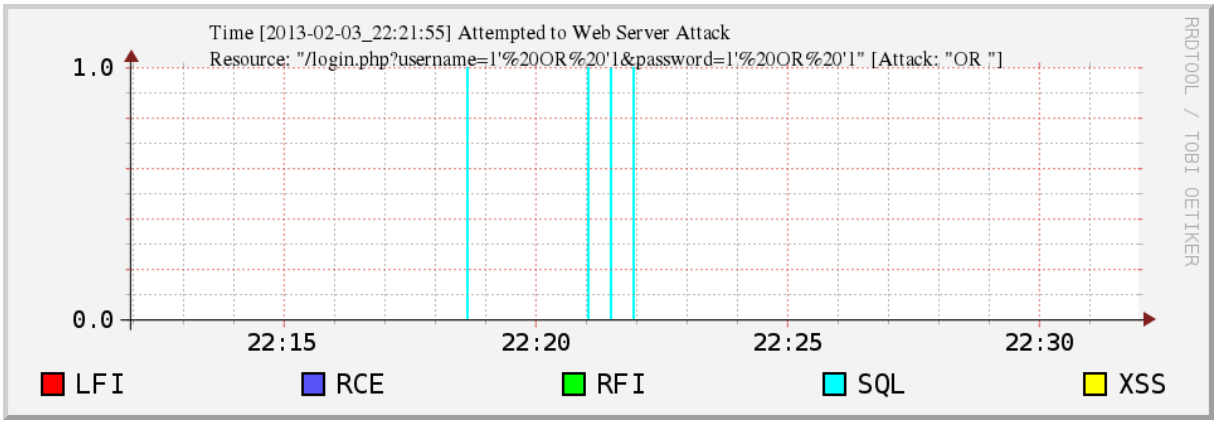


Figura 24: Gráfico da vulnerabilidade SQL gerado com os registros do *log*.
Fonte: do Autor.

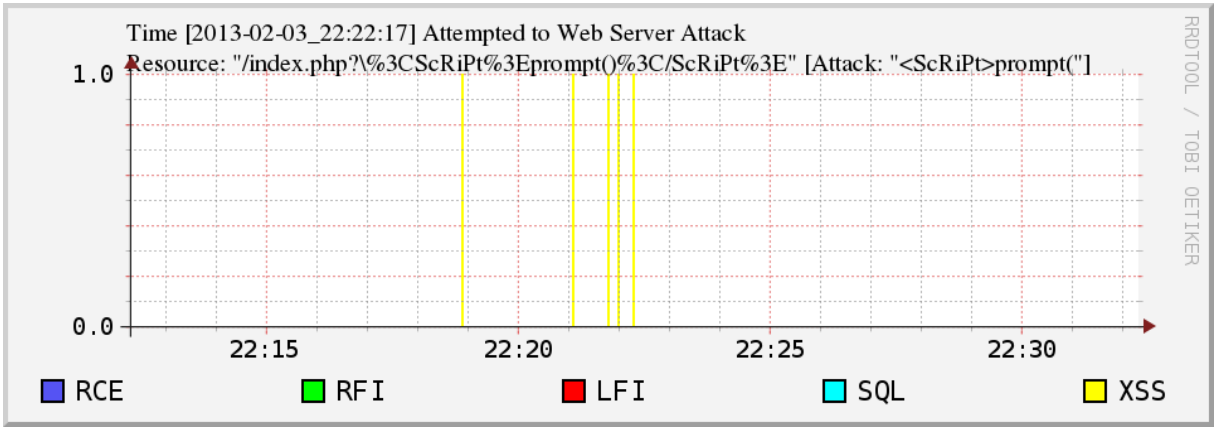


Figura 25: Gráfico da vulnerabilidade XSS gerado com os registros do *log*.
Fonte: do Autor.

Por fim, como exemplo, para a geração do gráfico geral contendo todas as vulnerabilidades foi utilizado o seguinte *log*, mostrado a seguir na Figura 26. E em continuidade seu gráfico gerado na Figura 27.

```
[2013-02-04 12:52:54] [Timestamp: 1359989574] Access to Resource "/rce.php?cat%20/" denied. Pattern "cat " found
. Descriptor file attack: Plugins/WAF/RCE.pm
[2013-02-04 12:52:59] [Timestamp: 1359989579] Access to Resource "/login.php?username='1%20OR%201'&password='1%2
0OR%201'" denied. Pattern "OR " found. Descriptor file attack: Plugins/WAF/SQL.pm
[2013-02-04 12:53:05] [Timestamp: 1359989585] Access to Resource "/index.php?%3CScRiPt%3Eprompt(%22%22)%3C/ScRiP
t%3E" denied. Pattern "<ScRiPt>prompt(" found. Descriptor file attack: Plugins/WAF/XSS.pm
[2013-02-04 12:53:08] [Timestamp: 1359989588] Access to Resource "/insecured.php?secret_file=http://192.168.2.3"
 denied. Pattern "http://" found. Descriptor file attack: Plugins/WAF/RFI.pm
[2013-02-04 12:53:11] [Timestamp: 1359989591] Access to Resource "/insecured.php?secret_file=/etc/profile?/etc/p
asswd" denied. Pattern "etc/passwd" found. Descriptor file attack: Plugins/WAF/LFI.pm
[2013-02-04 12:53:20] [Timestamp: 1359989600] Access to Resource "/index.php?%3CScRiPt%3Eprompt(%22%22)%3C/ScRiP
t%3E" denied. Pattern "<ScRiPt>prompt(" found. Descriptor file attack: Plugins/WAF/XSS.pm
[2013-02-04 12:53:27] [Timestamp: 1359989607] Access to Resource "/rce.php?cat%20/" denied. Pattern "cat " found
. Descriptor file attack: Plugins/WAF/RCE.pm
[2013-02-04 12:53:29] [Timestamp: 1359989609] Access to Resource "/insecured.php?secret_file=http://192.168.2.3"
 denied. Pattern "http://" found. Descriptor file attack: Plugins/WAF/RFI.pm
[2013-02-04 12:53:33] [Timestamp: 1359989613] Access to Resource "/login.php?username='1%20OR%201'&password='1%2
0OR%201'" denied. Pattern "OR " found. Descriptor file attack: Plugins/WAF/SQL.pm
[2013-02-04 12:53:37] [Timestamp: 1359989617] Access to Resource "/insecured.php?secret_file=/etc/profile?/etc/p
asswd" denied. Pattern "etc/passwd" found. Descriptor file attack: Plugins/WAF/LFI.pm
[2013-02-04 12:53:45] [Timestamp: 1359989625] Access to Resource "/index.php?%3CScRiPt%3Eprompt(%22%22)%3C/ScRiP
t%3E" denied. Pattern "<ScRiPt>prompt(" found. Descriptor file attack: Plugins/WAF/XSS.pm
[2013-02-04 12:53:52] [Timestamp: 1359989632] Access to Resource "/index.php?%3CScRiPt%3Eprompt(%22%22)%3C/ScRiP
t%3E" denied. Pattern "<ScRiPt>prompt(" found. Descriptor file attack: Plugins/WAF/XSS.pm
[2013-02-04 12:53:55] [Timestamp: 1359989635] Access to Resource "/rce.php?cat%20/" denied. Pattern "cat " found
. Descriptor file attack: Plugins/WAF/RCE.pm
[2013-02-04 12:54:02] [Timestamp: 1359989642] Access to Resource "/login.php?username='1%20OR%201'&password='1%2
0OR%201'" denied. Pattern "OR " found. Descriptor file attack: Plugins/WAF/SQL.pm
[2013-02-04 12:54:07] [Timestamp: 1359989647] Access to Resource "/insecured.php?secret_file=http://192.168.2.3"
 denied. Pattern "http://" found. Descriptor file attack: Plugins/WAF/RFI.pm
[2013-02-04 12:54:10] [Timestamp: 1359989650] Access to Resource "/insecured.php?secret_file=/etc/profile?/etc/p
asswd" denied. Pattern "etc/passwd" found. Descriptor file attack: Plugins/WAF/LFI.pm
[2013-02-04 12:54:25] [Timestamp: 1359989656] Access to Resource "/insecured.php?secret_file=http://192.168.2.3"
 denied. Pattern "http://" found. Descriptor file attack: Plugins/WAF/RFI.pm
brunolucena@brunolucena:~$
```

Figura 26: Exemplo de *Log* para geração do gráfico contendo todas as vulnerabilidades.
Fonte: do Autor.

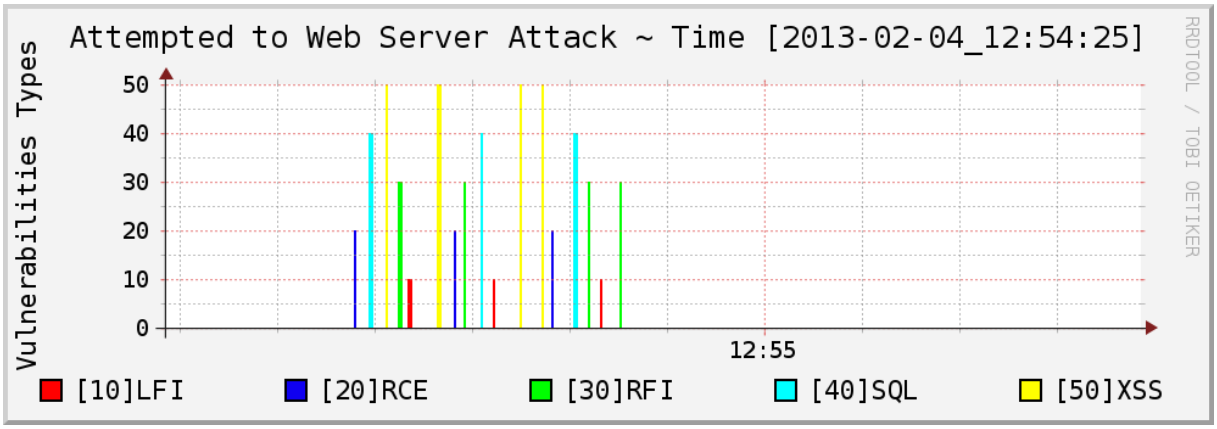


Figura 27: Gráfico contendo todas as vulnerabilidades gerado com o *Log*.
Fonte: do Autor.

A interpretação de tal gráfico dar-se-á, primeiramente pela leitura do título, contendo o momento exato do último ocorrido. Bem com, a análise da legenda, a qual além da diferenciação das cores, faz a atribuição de valores diferenciados para cada um dos tipos de vulnerabilidades, melhorando a visualização do administrador

de rede. E, não menos importantes, as análises dos eixos x e y, onde mostram o gráfico em função do tempo e os valores atribuídos a cada vulnerabilidade (10 para LFI; 20 para RCE; 30 para RFI; 40 para SQL e 50 para XSS).

Com este tipo de gráfico pode-se saber, por exemplo, se a aplicação *web*, está recebendo um grande número de tentativas de ataques a qualquer um dos cinco tipos de vulnerabilidades que o *Firewall* de Aplicação UniscanWAF detecta, sem a necessidade de fazer a análise de múltiplos gráficos.

6. TRABALHOS FUTUROS

6.1. WIMU: Web Interface Management UniscanWAF

A idéia de se implementar uma interface para gerenciar a saída UniscanWAF, envolvendo os *logs* implementados e os gráficos gerados, é algo que gera uma contribuição para a ferramenta UniscanWAF. Com esta interface será possível gerar estatísticas, das detecções encontradas com o UniscanWAF. Em vista disto, com a gerência desdes dados, poderão ser descobertos novos tipos de vulnerabilidades a partir de uma análise comportamental anômala das requisições recebidas no servidor *web* com o firewall de aplicação.

Visto que obteve-se apenas a conclusão parcial da interface de gerenciamento do *firewall* de aplicação UniscanWAF, a WIMU será proposta como um trabalho futuro, para possíveis publicações, juntamente com todos os aprimoramentos possíveis para contribuir com a ferramenta UniscanWAF.

A Figura 24 mostra uma demonstração do layout de como será a interface da ferramenta de gerenciamento de *logs* WIMU.

Figura 28: Interface da ferramenta de gerenciamento de *logs* (WIMU).
Fonte: do Autor.

7. CONSIDERAÇÕES FINAIS

Definir o número de informações que uma aplicação *Web* pode tratar diariamente é uma tarefa complexa, pois deve-se para isto tomar nota de todos os eventos que acontecem. Visto que, normalmente, um servidor *web* gera um grande volume de dados, esta tarefa torna-se uma função com um grau de dificuldade bastante elevado. Conforme cresce o número de acessos aumenta-se o número de informações recebidas pela aplicação *Web*, tendo a necessidade de haver um tratamento adequado destes dados para possíveis ações ativas ou, melhor ainda, pró-ativas.

Este trabalho visou compreender uma parte sobre o estudo desta área de gerenciamento de informações, utilizando para isto, um fluxo de informações geradas pela saída de um *firewall* de aplicação, o UniscanWAF.

Foi feito a implementação de um sistema de *logs*, onde são registrados todos os eventos captados pela ferramenta, no caso as requisições ao servidor *web.* Bem como também, a geração de um registro quantitativo do total de vulnerabilidades detectadas com a ferramenta, a partir de tais eventos pode-se traçar uma análise comportamental diária do número de vulnerabilidades mais requisitadas ao servidor *web,* podendo-se aplicar um gerenciamento de contabilidade, por exemplo.

Para auxiliar o administrador de rede, que neste caso é o responsável pela manipulação das ferramentas para garantir segurança ao servidor *web*, foram gerados gráficos a partir das informações encontradas nos *logs*, facilitando a visualização das informações, promovendo uma interpretação simples das informações geradas inicialmente pela aplicação. Em vista disto, com o gerenciamento desdes dados, poderão ser descobertos novos tipos de vulnerabilidades a partir de uma análise de comportamento das requisições recebidas no servidor *web*.

Visto que não foi possível concluir a interface de gerenciamento do *firewall* de aplicação UniscanWAF, a WIMU será desenvolvida futuramente, gerando possíveis publicações. Fazendo assim, com que a ferramenta UniscanWAF ganhe novos recursos e funcionalidades, contribuindo com a evolução do UniscanWAF. Um

aspecto interessante para aprimoramento seria, aplicando-se a metodologia de detecção por comportamento, isto é, a partir de instruções detectadas por meio do comportamento do sistema (estatístico por exemplo) auxiliando na descoberta de novas vulnerabilidades.

8. REFERÊNCIAS BIBLIOGRÁFICAS

ACUNETIX. **Website Security - Acunetix Web Security Scanner**. Disponível em: <http://www.acunetix.com>. Acessado em: 07/11/2012.

AUDITCONSOLE. **A Web-Console for Managing ModSecurity Events**. Disponível em: <http://jwall.org/web/audit/console/index.jsp>. Acessado em: 16/12/2012.

BELL, Jerome. **Log Management**. Disponível em: <http://www.syslog.org/>. Acessado em: 03/11/2012. Stelesys. USA: Hiram, 2009.

CERT.br, NBSO. **Práticas de Seguranca para Administradores de Redes Internet**. Disponível em: <http://www.cert.br/docs/seg-adm-redes/>. Acessado em: 05/11/2012. Versão 1.2. NIC BR Security Office, 2003.

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

CIRT. **Nikto2 - CIRT.net**. Disponível em: <http://cirt.net/nikto2/>. Acessado em: 07/11/2012.

CLEMM, A. **Network Management Fundamentals – A Guide to Understanding How Network Management Tecnology Really Works**. CISCO Press, 2006.

DANTAS, Mario. **Tecnologias de Redes de Comunicação e Computadores**. 1. ed. Rio de Janeiro: Axel Books do Brasil Editora Ltda, 2002.

FABRO NETO, A.; TURCHETTI, R.; TROIS, C.; PRIESNITZ, W. F. **Avaliação Qualitativa e Quantitativa entre Ferramentas para Detecção de Vulnerabilidades em Código Fonte para Aplicações Web.** 7ª Conferência Ibérica de Sistemas e Tecnologias de Informação, Madrid, 2012.

FABRO NETO, A.; TURCHETTI, R.; PRIESNITZ, W. F.; TROIS, C.; RIZZETTI, T.; ROCHA, D; KREUTZ, D. **Proposta e Implementação de um Firewall para Aplicações Web Denominado UniscanWAF**. 10ª Escola Regional de Redes de Computadores - UFPEL, 2012.

FABRO NETO, A.; TURCHETTI, R.; PRIESNITZ, W. F.; TROIS, C.; RIZZETTI, T.; LUCENA, B. **Implementação de um Firewall para detecção de vulnerabilidades na camada de aplicação.** 27ª Jornada Acadêmica Integrada - UFSM, 2012.

FREITAS, C. A.; MONTEIRO, J. W. A. **Análise de protocolos na Área de Gerência de Redes (SNMP/RMON)**. Universidade Federal de Goiás – UFG, 2004.

FRYE, R.; LEVI, D.; ROUTHIER, S.; WIJNEN, B. **[RFC-3584] - Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework**. Disponível em: <http://www.ietf.org/rfc/rfc3584.txt>. Acessado em: 10/11/2012. United States of America: Virginia, 2003.

GERHARDS, R. **[RFC-5424] – The Syslog Protocol**. Adiscon GmbH. Network Working Group. March 2009. Disponível em: <http://www.ietf.org/rfc/rfc5424.txt>. Acessado em: 04/11/2012. Germany: Grossrinderfeld, 2009.

HAKING. **Haking Extra – IT Security Magazine**. Hakin9 Media Sp. z o.o. SK , 02-682 Warszawa, ul. Bokserska 1 . Disponível em: <http://www.hakin9.org>. Acessado em: 17/11/2012.

ISO. **International Organization for Standardization - ISO**. Disponível em: <http://www.iso.org>. Acessado em: 07/01/2012.

KUROSE, James F.; ROSS, Keith W. **Redes de Computadores e a Internet: Uma abordagem Top-down**. 5. ed. São Paulo: Pearson Prentice Hall, 2010.

LOGGLY. **Log Management Service in the Cloud - Loggly**. Disponível em: <http://www.loggly.com/>. Acessado em: 24/11/2012.

LOGSTASH. **Open Source Log Management - Logstash**. Disponível em: <http://www.logstash.net/>. Acessado em: 13/11/2012.

MAURO, Douglas R.; SCHMIDT, Kevin J. **SNMP Essential - Help for System and Network Administrators**. 2. ed. O'reilly Book. United States of America: 2005.

MODSECURITY. **ModSecurity: Open Source Web Application Firewall**. Disponível em: <http://www.modsecurity.org/>. Acessado em: 09/11/2012.

NAKAMURA, E.; GEUS, P. **Segurança de Redes em ambientes cooperativos**. 1. ed. São Paulo: Novatec Editora, 2007.

NETO, Pedro de Alcântara. **Rede de gerência para telecomunicações**. Disponível em: <http://poli.br/~pan/Apostila%20-%20REDES%20-%20pdf/009%20-%20Cap%EDtulo%2009%20-%20Rede%20de%20gerencia%20para%20as%20telecomunica%E7%F5es.pdf>. Acessado em: 12/12/2012. Versão: Fevereiro, 2009.

NETSTAT. **Network Statistics - NetStat**. Disponível em: <http://www.netstat.net/>. Acessado em: 03/01/2013.

OWASP. **Open Web Application Security Project – OWASP**. Disponível em: <https://www.owasp.org/>. Acessado em: 06/11/2012.

OETIKER, T. **Round Robin Database Tool. RRDtool – Logging & Graphing.** Disponível em: <http://oss.oetiker.ch/rrdtool/>. Acessado em: 01/10/2012. Olten Switzerland, 2009.

PERL. **The Perl Programming Language**. Disponível em: <http://www.perl.org/>. Acessado em: 23/12/2012.

PINHEIRO, José Maurício dos Santos. **Gerenciamento de Redes de Computadores**. Versão: 2.0 - Agosto de 2002. Disponível em: <http://www.allnetcom.com.br/upload/GerenciamentodeRedes.pdf>. Acessado em: 09/11/2012. v.2.0. Agosto, 2002.

ROCHA, Douglas Poerschke. **Uniscan: Um scanner de vulnerabilidades para sistemas Web**. Trabalho de Conclusão de Curso. Ciência da Computação da UNIPAMPA. Alegrete, 2012.

SHIREY, R. W. **[RFC-2828]** - **Internet Security Glossary**. GTE / BBN Technologies. Network Working Group. May 2000. Disponível em: <http://www.ietf.org/rfc/rfc2828.txt>. Acessado em: 10/10/2012. USA: Arlington, 2000.

STALLINGS, William. **Criptografia e segurança de redes – Princípios e práticas**. 4. ed. São Paulo: Pearson Prentice Hall, 2008.

STALLINGS, William. **SNMP, SNMPv2, SNMPv3 and RMON 1 and 2**. Networking & Telecommunications. 3. ed. Hardcover, Addison-Wesley, 1999.

TIMESTAMP. **Unix Time Stamp . Com** .Disponível em: <http://www.unixtimestamp.com/index.php>. Acessado em: 14/01/2013.

TANENBAUM, Andrew S. **Redes de Computadores**. 4. ed. Campus, 2003.

WAFEC. **Web Application Firewall Evaluation Criteria.** Disponível em: <http://www.webappsec.org/projects/wafec/v1/wasc-wafec-v1.0.pdf>. Acessado em: 10/10/2012. Web Application Security Consortium, 2006.

WAPITI. **Wapiti – Web application security auditor**. Disponível em: <http://wapiti.sourceforge.net/>. Acessado em: 08/11/2012.