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

Sadan Eduardo Moura Figueira

DESENVOLVIMENTO DE UMA FERRAMENTA MODULAR PARA ANÁLISE DE *LOGS* DE APLICAÇÕES DE SEGURANÇA DE REDES

Santa Maria/RS
2017

CTISM/UFSM,BR MOURA FIGUEIRA, sadan eduardo Tecnólogo 2017

Sadan Eduardo Moura Figueira

DESENVOLVIMENTO DE UMA FERRAMENTA MODULAR PARA ANÁLISE DE *LOGS* DE APLICAÇÕES DE SEGURANÇA DE REDES

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

Orientador: Prof. Me. Tiago Antônio Rizzetti

Ficha gerada com os dados fornecidos pelo autor

Figueira, Sadan Eduardo Moura
/ Sadan Eduardo Moura Figueira.-2017.
59 p.; 30cm

Orientador: Tiago Antônio Rizzetti
Coorientadora: Maria Silva
Trabalho de conclusão de curso – Universidade Federal de Santa Maria, Colégio Técnico Industrial de Santa Maria, Curso Superior de Tecnologia em Redes de Computadores, RS, 2017.

1. Análise de logs 2. segurança de rede 3. monitoramento I. Rizzetti, Tiago Antônio II. Figueira, Sadan Eduardo Moura

Este trabalho está licenciado pela Creative Commons Attribution-ShareAlike 4.0
This work is licensed under the Creative Commons Attribution-ShareAlike 4.0

Sadan Eduardo Moura Figueira

DESENVOLVIMENTO DE FERRAMENTA MODULAR PARA ANÁLISE DE *LOGS* DE APLICAÇÕES DE SEGURANÇA DE REDES

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

Aprovado em 12 de julho de 2017:

_____

Prof. Me. Tiago Antônio Rizzetti **(UFSM)**
(Orientador)

_____

Bolívar Menezes da Silva, Tecg**. (UFSM)**

_____

Alexandre Silva Rodrigues, Tecg**. (UFSM)**

Santa Maria/RS
2017

DEDICATÓRIA

Gostaria de dedicar a minha família e a todos que contribuíram de alguma forma até esse momento.

AGRADECIMENTOS

Agradeço a todos que me auxiliaram até o momento.

“Domine seu medo ou vire refém dele.”
(Autor desconhecido.)

RESUMO

DESENVOLVIMENTO DE FERRAMENTA MODULAR PARA CONTROLE DE APLICAÇÕES DE SEGURANÇA DE REDES

AUTOR: Sadan Eduardo Moura Figueira
ORIENTADOR: Prof. Me. Tiago Antônio Rizzetti

Mesmo em redes de pequeno porte o processamento do excessivo número de *logs* gerados pelas diversas ferramentas de segurança de rede podem dificultar o trabalho do administrador da rede, a ponto do mesmo ignorar esses arquivos e somente recorrer a eles no caso do acontecimento de alguma falha de segurança. Mediante ao surgimento de ameaças a segurança aos sistemas de informação surgiu a necessidade da listagem e catalogação das mesmas, de forma que diversas empresas e grupos além de coletar também começaram a compartilhar essas informações. Dessa forma atualmente existem muitos bancos de dados que permitem a consulta a essas informações de segurança, as quais são compartilhadas através de *APIs*. A falta de uma ferramenta que permita realizar análise dos *logs* de diversas ferramentas em conjunto com essas diversas *APIs* motivou este trabalho e a criação da ferramenta LogParser, através da qual, será possível buscar ameaças encontradas e compartilhadas por outras organizações nos arquivos de *log* gerados por aplicações de rede e armazenados em um servidor de *logs*, permitindo assim um melhor tratamento das informações contidas nesses arquivos assim como uma melhor contenção de ameaças.

Palavras-chave: análise de *logs*, segurança de redes, monitoramento de ameaças.

ABSTRACT

DEVELOPMENT OF A MODULAR TOOL FOR ANALYSIS OF NETWORK SECURITY APPLICATIONS LOGS.

AUTHOR: Sadan Eduardo Moura Figueira
ADVISOR: Prof. Me. Tiago Antônio Rizzetti

Even in small networks processing the excessive number of logs files generated by the various network security tools can be a challenge for the network administrator, making him ignore these files and only resort to them in the event of a security breach. Due to the emergence of security threats to information systems, the need for listing and cataloging them has arisen, with that in mind several companies and groups besides collecting have also started to share this information. The lack of a tool that allows to analyze the logs of several tools together with these several APIs motivated this work and The creation of the LogParser tool, through which it will be possible to search for threats found and shared by other organizations in the log files generated by network applications and stored in a log server, thus allowing a better treatment of the information contained in these files as well as a Better containment of threats.

Keywords: logs analizys, network security, threat monitoring.

ÍNDICE DE FIGURAS:

LISTA DE TABELAS:

LISTA DE ABREVIATURAS E SIGLAS:

HTTP – HyperText Transfer Protocol

SSH – Secure Socket Shell

SCP – Secure copy protocol

API – Application program interface

PHP – Personal home page

REGEX – Regular expression

DNS – Domain Name System

ERP – Enterprise resource planning

SGBD – Sistema de gerenciamento de banco de dados

IPS – Intrusion Prevention System

IDS – Intrusion Detection System

IBM – International Business Machines

IP – Internet Protocol

XML – eXtensible Markup Language

JSON – JavaScript Object Notation

URL – Uniform Resource Locator

RAM – Random Access Memory

GB – GigaBit

SQL – Structured Query Language

SUMÁRIO:

1 INTRODUÇÃO

Atualmente, através de sistemas informatizados, compartilhamos e expomos diversas informações, para aumentar a praticidade das ações do nosso dia a dia, acessamos nossas contas do banco pela internet, realizamos pagamentos, trocando arquivos e mensagens. Tamanha quantidade de informações se tornam um atrativo para pessoas mal-intencionadas que buscam, de forma ilegal, se apossar dessas informações para usos ilícitos.

Para combater essas e outras ameaças a segurança dos usuários de um sistema de telecomunicações diversas ferramentas e tecnologias foram e são desenvolvidas, tanto com o intuito de analisar ameaças quanto de as bloqueá-las, entre elas podemos citar *IDS* (I*ntrusion Detection Systems*), servidores de *proxy*, *firewalls*, sistemas de criptografia entre outras, assim como diversos grupos catalogam informações relativas a essas ameaças e as fornecem para consulta do público. Entretanto por diversos motivos uma ameaça pode passar por uma dessas ferramentas de forma que a mesma não a considere uma ameaça e sim uma ação legítima de algum usuário, porém ações legítimas ou não todas podem serem logadas em arquivos de *log* gerados por essas ferramentas.

No campo da computação, arquivos de *log* podem armazenar informações para diversos usos, como para encontrar erros, tanto falhas na execução de softwares quanto falhas de segurança dentro de uma rede. Nesse último caso, comumente aplicações tanto de segurança quanto as que funcionam provendo serviços de rede mantêm um padrão quanto as informações guardadas nesses arquivos, contendo geralmente endereços *IPs* referentes as conexões, faixas de horário entre outras que permitem uma localização de informações utilizando diversas fontes referentes as máquinas que realizaram a conexão.

Segundo Stallings (2014), um conhecido sistema de segurança implementado em *firewall* são as chamadas listas de bloqueio, conhecidas também pelo termo *blacklist*. As mesmas funcionam de forma que endereços *IPs* previamente conhecidos como danosos tenham seu acesso bloqueado a rede. Esses *IPs* podem ser encontrados em bancos de dados de ameaças, que podem ser alimentados por usuários ou ate mesmo empresas. Entretanto esses bancos não contém as mesmas informações, assim como não mantêm um padrão de retorno da mesma, por uma série de diferentes fatores. Então como se manter a par de diversas ameaças?

Diversas fontes armazenam informações sobre endereços *IPs* em toda a internet, através de uma rápida pesquisa podemos encontrar informações sobre a quem pertence um determinado endereço, entretanto estas informações na maioria dos casos não são suficientes

para identificar uma fonte de ameaças. Dessa forma surgiram bancos de dados dedicados a coletar, armazenar e distribuir, utilizando-se de *APIs* para consultas, essas informações, dando assim uma noção referente a ameaça apresentada pelo mesmo.

Com isso nos deparamos com o problema de como fazer o processamento quanto a possíveis ameaças encontradas nos arquivos de *logs* gerados por diversas aplicações de rede, que possuem diferentes sintaxes quanto a disposição de informações, buscando ao mesmo tempo dados referentes aos endereços *IPs* em diferentes bancos de dados e após exibir ao usuário informações extraídas proveniente de diferentes formatos de retorno de *APIs*.

Mediante a tal desafio é proposto o desenvolvimento da ferramenta LogParser, através da qual administradores de rede poderão analisar o conteúdo de seus arquivos de *log* quanto a ameaças relatadas por outros administradores. Através da separação da ferramenta em módulos, será possível processar arquivos de *log* de praticamente qualquer aplicação de segurança, assim como será possível realizar consulta em diversas *APIs*.

1.1 OBJETIVO GERAL

A presente monografia tem como objetivo desenvolver uma aplicação capaz de realizar a análise de arquivos de *logs* para a extração de informações baseadas em padrões supridos pelo usuário, após pesquisar a ocorrência dos mesmos em bancos de dados de ameaças que disponibilizam consultas sobre essas informações. Uma vez obtidas, essas informações serão exibidas ao usuário que poderá tomar uma melhor decisão em relação a segurança de rede.

1.2 OBJETIVOS ESPECÍFICOS

→ Desenvolvimento de uma interface *web* de controle para a aplicação que auxilie o administrador a usar a ferramenta com diversas aplicações através da criação de módulos.

→Desenvolver um sistema capaz de analisar os arquivos de *log* de várias aplicações contanto que estejam em formato de texto plano ou em um banco de dados MySQL.

→Desenvolver um sistema capaz de se utilizar de variadas fontes na busca por informações relativas a ameaças encontradas nos arquivos de *log*, contanto que essas fontes disponibilizem acesso através de alguma *API*.

1.3 JUSTIFICATIVA

Mesmo em um curtos espaço de tempos arquivos de *logs* de aplicações se tornam de difícil leitura para humanos devido a sua grande quantidade de informações. Entretanto, no caso de aplicações de rede, tanto de segurança, gerenciamento e de uso geral de uma organização essas informações podem ser de extrema importância na busca por ameaças que por ventura passam desapercebidas pelas aplicações de segurança da rede.

Bancos de dados de ameaças surgiram para catalogar informações que podem indicar vulnerabilidades em sistemas informatizados, os mesmos são mantidos tanto por sua própria comunidade assim como por organizações que também estão interessadas na segurança geral da internet e seus usuários. Muitos desses bancos ainda dispõem *APIs* as quais é possível consultar grandes quantidades de informações.

Nesse trabalho foi desenvolvida uma ferramenta capaz de fazer a pesquisa de diversos tipos de informações em arquivos de *logs* de diversas aplicações e após pesquisar se essas informações são provenientes de alguma ameaça detectada e catalogada previamente em algum banco de dados de ameaças.

1.4 ESTRUTURA DO TRABALHO

A descrição detalhada do desenvolvimento deste projeto encontra-se apresentada neste documento na forma de capítulos. O primeiro mostrando o tema e os objetivos do mesmo, o segundo apresentando uma revisão bibliográfica das tecnologias utilizadas no desenvolvimento da ferramenta. O terceiro apresenta uma detalhada descrição da proposta. O Quarto capítulo apresenta todos os passos que foram seguidos durante o desenvolvimento da ferramenta. No quinto capítulo são realizados testes para comprovar o correto funcionamento da ferramenta, enquanto no sexto capítulo é feita a análise dos resultados provenientes dos mesmos. Por último, no sétimo capítulo são mostradas as conclusões e possíveis trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Para a realização do trabalho proposto será necessário a integração de diversas ferramentas e tecnologias da área de segurança, entre elas temos:

2.1 *LOGS*;

Segundo Kent (2006) Arquivos de *logs* são registros históricos de acontecimentos. No campo da computação eles existem tanto como uma ferramenta de depuração de erros como de segurança, através deles podemos analisar o comportamento de um sistema ou aplicação, buscando erros e gargalos assim como registrando, possivelmente, todas as ações tomadas tanto por usuários ou mesmo pelo funcionamento de alguma ferramenta, podendo gerar um mapa completo do seu comportamento, através deles violações de segurança podem ser detectadas logo após ocorrerem.

Para garantir sua usabilidade em caso de falhas, visando também uma melhor segurança, é comum ser feita uma espécie de pré-processamento, geralmente ou transformando essas entradas do *log* em um banco de dados ou mesmo movendo para um servidor de *logs*, evitando assim, perca de informações, Kent (2006).

Segundo Vandman **(**2001) mesmo com diferentes aplicações gerando diferentes sintaxes em seus arquivos de *log*, *os mesmos* também tentam manter um padrão de informações contidas, como por exemplo, no campo de segurança, endereços de destino, e origem de cada entrada, tais padrões serão de extrema importância no desenvolvimento desse trabalho pois permitirão uma fácil analise através do uso de expressões regulares.

2.2 *REGEX* OU EXPRESSÕES REGULARES;

Segundo Nguyen (2013) Expressões regulares, conhecidas como *regex*, são padrões de strings que costumam se repetir dentro de um arquivo contendo somente caracteres. Através do uso dessa ferramenta podemos localizar esses padrões dentro de um texto, como, por exemplo, uma linha de um arquivo de *log* ou um campo dentro de uma linha em um grande arquivo.

Essas expressões regulares podem ser aplicadas através de diferentes formas por diferentes linguagens de programação. Nesse trabalho serão aplicadas através da linguagem de programação *Python*, que possui, por padrão, a biblioteca RE capaz de fazer análise em

strings unicode e 8-bits. Ao utilizar esta biblioteca podemos, por exemplo, buscar uma sub-*string* dentro de uma linha de uma forma bastante simplificada através do uso de funções.

Embora seja uma ferramenta muito útil no processamento de texto ela apresenta alguns problemas que deverão ser devidamente trabalhados para o sucesso deste trabalho, como, por exemplo, seu alto custo computacional e a complexidade envolvida na geração de padrões de busca. Por exemplo, se desejamos uma sub-*string* dentro de uma *string* utilizando *python*, basta utilizar a função RE.*search*() que mostrará se a *string* contém a sub-*string*. Entretanto o padrão de pesquisa é diferente caso seja necessário procurar, por exemplo, um endereço *IP* que fica após o campo DST caso a linha indique acesso externo, em um arquivo SYSlog.

2.3 *PYTHON*;

Segundo Mckinney (2012) *Python* é uma linguagem de programação interpretada, ou seja, que não é compilada, de alto nível que teve sua primeira aparição em 1991, visada como uma linguagem de propósito genérico e amplamente usada no ramo científico. A Sua principal característica é a fácil legibilidade do código, devido principalmente a sua forma obrigatória de estruturação e indentação. A linguagem também funciona em uma grande gama de sistemas operacionais e dispositivos ao mesmo tempo que suporta diversos paradigmas de programação diferentes.

Os principais motivos da escolha da linguagem no desenvolvimento da ferramenta foram seu grande número de bibliotecas, com um repositório centralizado, que permite uma fácil instalação de dependências necessárias a ferramenta. Além da eficaz integração com sistemas operacionais Linux e a preferência do autor.

A fácil instalação de bibliotecas externas permite uma fácil reutilização de código, por exemplo, podemos usar a biblioteca *Flask* para a criação de um pequeno servidor web, o mesmo será usado para a implementação da interface web e também a biblioteca *Requests* que será usada para fazer consultas as diversas *APIs* externas e também uma leitura inicial do resultado

2.4 *RESTFULL API*:

Segundo Woods (2011) uma interface de programação de aplicativos ou *API* tem como suas funções tanto compartilhar funções quanto informações sem a necessidade de grande exposição da estrutura interna de sua prestadora. Sendo assim, a mesma funciona como uma espécie de *backend* para diversas aplicações, como exemplo, uma organização pode desenvolver um *software* e disponibilizar seu uso através de uma *API*, a partir desse momento uma outra organização pode aproveitar essa como parte integrante de seu *software*.

Segundo Richardson (2013) as *restfull APIs* descrevem um meio de compartilhar informação de forma organizada, a qual, por exemplo, retorna um *json* após uma requisição. Como um exemplo dessas *APIs* podemos ter *domainIP, project honey pot*, *Moocher.io* e *ibm x-force xchange.* Essas *APIs* proveem uma consulta simplificada em relação a diversos tipos de dados conhecidos como ameaças de segurança.

2.5 *IBM X-FORCE EXCHANGE*:

Segundo Franklin (2015), desde os anos 90 a IBM vem realizando a coleta de informações na área de segurança, detalhando desde vulnerabilidades até reputação de endereços *IPs*, *DNS*, *URLs* e informações sobre *malwares*. No ano de 2015 ela tornou esses dados disponíveis ao público através de consultas no portal *ibm x-force exchang*, ao mesmo tempo ela disponibilizou uma *API* gratuita capaz de fazer um número determinado de consultas.

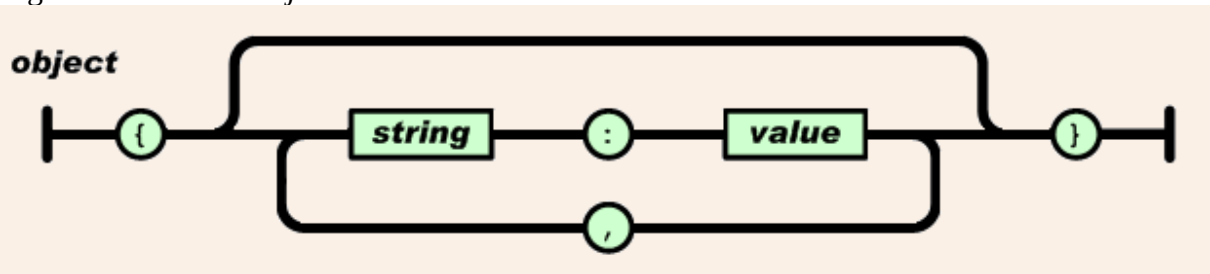
Segundo Franklin (2015) a mesma disponibiliza um grau de risco de 1, o qual não apresenta ameaça, até 10, o qual apresenta o maior nível de risco. Essa pontuação é obtida através do processamento de informações realizados pela IBM, por exemplo, se um *IP* começar a enviar *spam* a pontuação do mesmo aumentará, caso ele pare com essa distribuição com o passar do tempo sua reputação diminuirá. Outras ameaças a qual ele procura são *botnets*, tanto vítimas quanto servidores de comando; *malwares*, catalogando *IPs* conhecidos por serem um vetor de propagação de *malwares*; *scanners* de *IP*, indicando *IPs* que usam ferramentas para procurar vulnerabilidades em sistemas; *proxys*, que são serviços que oferecem proteção dos dados do usuário.

Com tamanha quantidade de informações providas por essa *API*, seu processamento seria difícil, caso sua sintaxe fosse desorganizada. Entre as opções para essa organização temos formatos conhecidos como *XML* e *JSON*.

2.6 JSON:

Segundo Richardson (2013) o formato *JSON* ou Notação de Objetos *JavaScript* foi desenvolvido com o objetivo de ser uma forma rápida e eficiente de troca informações além de ser independente da linguagem *javaScript*, diferente do que o nome sugere. Ela é de fácil leitura e escrita tanto para máquinas quanto para seres humanos, isso devido a sua organização que consiste basicamente em uma *string* formada por pares separados por dois pontos no qual o primeiro, que é um valor fixo, representa o nome do dado e o segundo o conteúdo, esses pares também podem ser agrupados, para isso bastando separá-los por vírgula, permitindo dessa forma grandes agrupamentos de dados. Podemos uma representação desse padrão na figura 2.1:

Figura 2.1: sintaxe do *json*



Fonte: http://www.json.org (acessado em 04 de maio de 2017)

Ao decorrer do desenvolvimento da ferramenta ele se mostrará de grande utilidade, pois, além da grande maioria das *APIs* usarem o mesmo para retornar dados ele também será utilizado para guardar os módulos que a ferramenta utilizará para se comunicar com as diversas ferramentas de segurança de rede.

2.7 XML

Segundo Ray (2001) XML é tanto um protocolo quanto uma linguagem de marcação extensiva desenvolvida para guardar informação através de *tags*. Através do XML podemos descrever padrões de informações de forma a nos facilitar diversas tarefas no campo do processamento de texto, como por exemplo, extração de dados de documentos devidamente formatados.

2.8 MYSQL

Em locais onde é necessário armazenar grandes quantidades de informações é comum o uso de bancos de dados, através dos quais podemos fazer um gerenciamento dessas informações de forma eficiente e rápida. Um dos modelo de banco de dados mais comuns são conhecidos como *SQL*, tal sigla pode tanto significar S*tructured Query Language* quanto fazer uma alusão ao nome de uma de suas versões anteriores, SEQUEL, (BEAULIEU, 2009), existindo também modelos *NO-SQL*.

Segundo Beaulieu (2009), um banco de dados podemos fazer diversas ações, entre elas criar, apagar e editar informações, porém para isso precisamos de um sistema que gerencie essas tarefas, para isso são usados os SGBDs ou Sistemas Gerenciadores de Bancos de Dados, onde como exemplos temos *postgres*, *mongoDB, slq-server*, *mariaDB, Mysql* entre outros. Os SGBDs tem como principal função criar uma camada entre a aplicação cliente e o banco de dados, diminuindo assim a complexidade da sua implementação e embora existam vários softwares diferentes a sintaxe de todos é muito próxima.

2.9 SISTEMAS DE SEGURANÇA DE REDES:

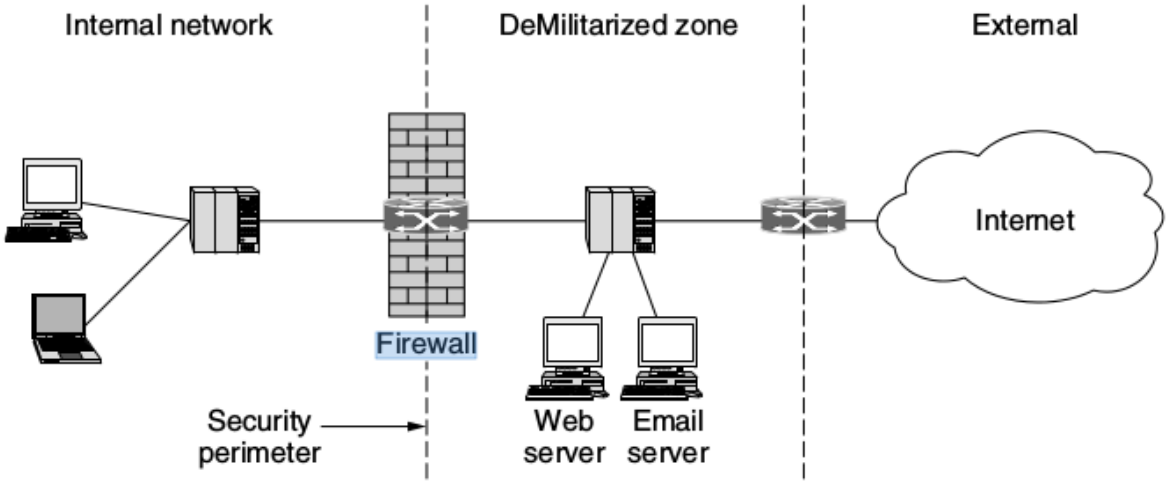
Segundo Tanenbaum e Wetherall (2011) organizações utilizam redes de computadores para o compartilhamento de diversas informações críticas ao seu funcionamento. Percebe então que elas se tornam alvos tentadores a ataques mal-intencionados, os quais podem causar uma série de problemas. Para evitar que tais problemas se tornem comuns ao longo dos anos foram desenvolvidas diversas ferramentas, tanto para bloqueio quanto análise de riscos, entre elas podemos citar *firewalls*, *IDS's* e sistemas de *proxy*.

2.9.1 *FIREWALLS*:

A implementação de um *firewall* aumenta a segurança da rede, ele, que é uma combinação de hardware e software, é implementado entre a rede interna e a internet, assim, servindo como um filtro de pacotes, impedindo a passagem de determinados tipos de pacotes ou de pacotes de determinadas origens. Um exemplo desse tipo de aplicação é o *netfilter* em conjunto com o *iptables*, que já vem implementado por padrão em diversos sistemas Linux (KUROSE, 2016), e por essa razão será usado nesse trabalho.

Segundo Stallings (2014) *firewalls* podem ser implementados tanto protegendo a rede interna da conexão com a internet como também pode ser implementa em redes internas, nas quais é possível fazer uma filtragem mais rigorosa dos dados assim como separar redes internas umas das outras Na figura 2.2 podemos ver uma forma comum de sua implementação.

Figura 2.2: cenário onde *firewall* é implementado.



Fonte: (Stallings, 2014)

2.9.2 IDS:

Uma importante ferramenta de segurança de rede é chamada de I*ntrusion Detection System* ou *S*istema de Detecção de Intrusos. Esse sistema realiza uma análise profunda do conteúdo de um pacote, verificando os dados transportados pelo mesmo, e no caso de encontrar alguma ameaça proíbe esses pacotes de entrarem em uma rede.

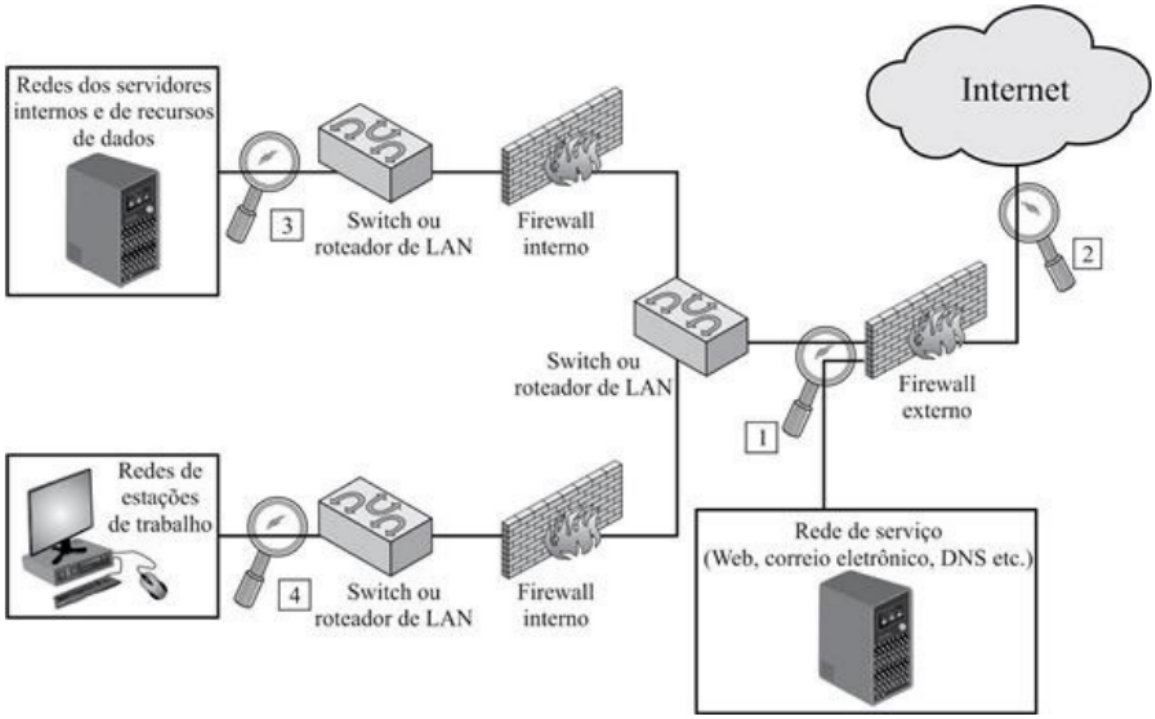
Eles são basicamente classificados em 2 tipos, o primeiro realiza a análise dos pacotes baseados em assinatura. O mesmo contém um banco dados contendo assinaturas de diversos tipos de ataques. O segundo é baseado em comportamento anômalo, o mesmo procura comportamentos de pacotes diferentes do normal.

Um sistema de *IDS* comumente usado é a aplicação *SNORT*. O sistema possui uma vasta comunidade de usuários e implementações para diversos sistemas operacionais, seus bancos de dados de assinaturas são constantemente atualizados. Geralmente após poucas

horas do surgimento do ataque algum filtro da assinatura já foi desenvolvido e compartilhado com a comunidade (KUROSE, 2016).

Os sensores de um sistema de *IDS* pode ser implementado em diversos locais. Na figura 2.3 podemos o ver implementado em 1, entre o *firewall* e a rede externa realizando a detecção do que passa pelo *firewall* e em 2 sem nenhuma proteção para detectar e documentar ataques destinados a rede. Em 3 ele é implementado perto dos serviços utilizados pelos usuários a fim de detectar violações de comportamento dos mesmos e em 4 visa detectar ataques a dados de usuários.

Figura 2.3: disposição de sistemas *IDS* em uma rede.



Fonte: (Stallings, 2014)

2.9.3 *PROXY*:

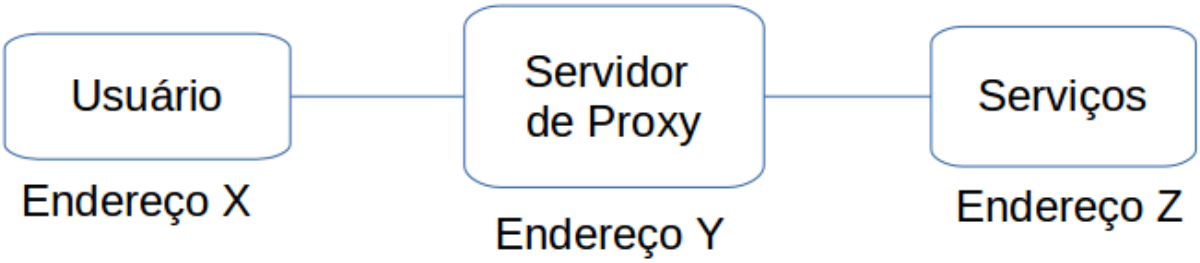
Comumente servidores de *proxy* atual como um intermediador entre um cliente e um serviço, ao fazer uma requisição o pacote oriundo do cliente passa primeiro pelo *proxy* que faz a comunicação com o servidor e após retorna para o cliente. Os principais motivos para a

implementação desse sistema são uma redução do uso de banda e uma melhoria da qualidade da internet do usuário (SAINI, 2011).

Entretanto aplicações de *proxy* também podem servir como uma forma de prover maior controle e segurança a rede, isso devido ao fato do mesmo fazer um processamento de todos os pacotes que tentam entrar ou sair da rede, também podendo realizar uma filtragem do mesmo.

Na figura 2.4 é exemplificada o uso de um sistema de *proxy*, o usuário com endereço X deseja acessar os serviços que estão hospedados no endereço Z, entretanto o servidor de *proxy*, que está no endereço Y, intermediará a comunicação entre os dois. Dessa forma garante-se um maior controle sobre o tráfego da rede.

Figura 2.4: funcionamento de um *proxy*.



Fonte: Arquivo pessoal.

Squid é o sistema de web *proxying* mais popularmente usado, implementado em diversos sistemas operacionais. O mesmo prove uma melhora significativa para a performance da rede através da realização de *caching* de páginas web. O mesmo ainda funciona como um *firewall* onde através de filtros, como, por exemplo, de endereços *IPs*, é feito o bloqueio de comunicação de forma ambi direcional (WESSESL,2009).

2.10 CRIPTOGRAFIA ASSIMÉTRICA

Entretanto as medidas de segurança descritas em 2.8.1, 2.8.2 e 2.8.3 podem não ser suficientes ou pode ocorrer o caso de tais informações passaram por trechos de redes desprotegidas. Se esses dados passarem em uma forma que o mesmo possa ser lido ou editável todo o sistema de segurança se tornaria inútil, por exemplo, se os *logs* de acesso dos

usuários forem capturados enquanto são movidos para um servidor de *logs*, o atacante pode ter um mapa total das ações dos usuários. Para contornar esse problema podemos usar diversas tecnologias de criptografia, entre elas, a criptografia assimétrica, a qual tem se mostrado ao longo dos anos muito robusta e escalável (TANENBAUM,2011).

Em 1976 foi proposto o método de chave pública, também conhecido como Criptografia Assimétrica, seu principal intuito foi resolver o problema de distribuição das chaves de segurança, tendo em vista que caso um intruso tenha acesso a uma dessas chaves todo o sistema estaria vulnerável. O sistema proposto visava a implementação de duas chaves uma sendo pública que podia ser enviada para qualquer cliente e é usada para criptografar os dados e uma chave privada capaz de descriptografar os dados (TANENBAUM,2011).

2.10.1 RSA:

O algoritmo *RSA* surgiu em 1978 como um dos primeiros algoritmos a implementar Criptografia assimétrica de forma eficiente e robusta, se mantendo por quase 4 décadas sem ser quebrado, grande parte de sua robustez vem de sua obrigação de usar uma chave mínima de 1024 bits, porém o mesmo torna o funcionamento geral do algoritmo lento.

Seu funcionamento se dá através do uso de números primos, através da multiplicação de dois valores numéricos extensos, de no mínimo 1024 bits, são criadas duas chaves, para realizar a quebra dessa criptografia basta fatorar um desses valores, entretanto como o número é muito grande o processo se torna inviável com a tecnologia atual, e caso o processo de fatoração se torne mais eficaz o número de bits pode ser aumentado (TANENBAUM, 2011).

2.11 SERVIDOR SSH E SCP

Ao trabalharmos com administração de tanto de servidores quanto de sistemas baseados em Linux percebemos a necessidade de realizar ações de forma remota, para isso surgiram diversas ferramentas, porém ao surgir a necessidade de conexões seguras foi criado o protocolo *SSH*, o mesmo permitindo uma conexão remota segura mesmo entre redes inseguras. O protocolo oferece uma criptografia entre suas comunicações e o mesmo duas versões, a primeira sendo ssh1, a mais simples e a segunda ssh2 que possui mais formas de criptografia (ELLEN, 2009). Diversas aplicações implementam o protocolo SSH para conexões remotas, entre eles temos *OpenSSH*, *DropBear* e *Apache MINA*.

Ao surgir a necessidade de envio de arquivos de forma segura entre servidores Linux, ainda podemos nos aproveitar dos protocolos de segurança provenientes do *SSH*, para isso podemos usar o protocolo *SCP*, S*ecure Copy Protocol,* que se utiliza da mesma forma de criptografia do *SSH* e assim como o mesmo, utiliza a porta 22 para suas conexões.

2.12 SISTEMA DE *ERP – ODOO* 8

É comum em empresas o uso de um sistema integrado aos seus diversos setores, garantido assim, mais agilidade aos seus processos. Como exemplo desse tipo de aplicação temos o *Odoo*, um software *opensource* capaz de se adaptar a diversas empresas de diversos ramos devido ao uso de módulos. Ela vem se tornando cada vez mais comum no Brasil devido a sua comunidade *opensource* que continua ativamente adequando a mesma ao cenário brasileiro.

Devido as suas funcionalidades de compartilhamento de dados e arquivos, seu sistema de múltiplos bancos de dados e usuários ela foi escolhida para mostrar como a logParser pode realizar o processamento de aplicações diversas. Nos testes foi usado sua versão 8, pois esta é a última versão a ser adaptada ao uso brasileiro.

2.13 TRABALHOS RELACIONADOS

Existem na literatura vários trabalhos tratando do gerenciamento e processamento de arquivos de *log*, tais trabalhos visam auxiliar os administradores e desenvolvedores a extrair o maior número possível de informações relevantes de seus sistemas e redes. Recentemente portais de compartilhamento de informações de segurança começaram a surgir, entretanto faltam na literatura trabalhos que tratem sobre seu uso.

Como é mostrado em Tweed e Rouse (2016) o compartilhamento de informações de segurança pode se tornar uma peça chave no esquema de segurança de uma rede. Através desse compartilhamento após a primeira ocorrência do ataque ser detectada será possível transmitir essas informações aos demais usuários que poderão bloqueá-las antes das mesmas ocorrerem, e como tais informações podem ser extraídas de sistemas de *honeypots*, é possível, que não haja uma primeira vítima. Vale ressaltar que é comum aos bancos de dados de ameaças implementarem interfaces web onde é possível realizar consultas sobre *IPs* e fazer busca de informações relevantes, porém com a existência de diversos bancos de dados onde a

informação é exposta de diversas maneiras se faz necessário a criação de uma central única para o agregamento e processamento de tais informações.

Em Aeri e Tukadiya, (2015) é feita uma comparação entre diversas ferramentas de monitoramento de arquivos de *log* sobre suas funcionalidades e características. A partir dela podemos observar que as ferramentas de análise citadas não fazem nenhuma espécie de consulta em bancos de dados de ameaças. É possível ver que tal comportamento é visto mesmo em ferramentas não citadas no trabalho. Em seu trabalho foram comparadas cinco ferramentas de análise e gerenciamento de logs em quesitos como sistemas de alerta; compatibilidade com aplicações; uso de filtros; pesquisas; criação e exibição de gráficos entre outros quesitos. Após levando em consideração esses critérios foi escolhida a aplicação KIWIlog.

Uma ferramenta voltada a análise de *logs* é desenvolvida em Rastogi e Akash (2016). Com ela é possível extrair diversas informações de arquivos de *log*, como por exemplo o número de ocorrências de uma *string*, e as exibir em forma de gráficos para os usuários, isso através de uma interface WEB. Ainda com tal ferramenta é possível a geração de padrões de expressões regulares usando exemplos de log. Entretanto tal ferramenta não foi projetada para lidar especificamente com *logs* de ferramentas de segurança de rede e bancos de dados de ameaças, mesmo sendo possível fazer a extração de endereços *IPs* de um arquivo de *log* ela não realizará uma pesquisa em relação aos mesmos, e mesmo as informações extraídas por ela tem maior utilidade na parte de geração de estatísticas. Ao final do trabalho a aplicação desenvolvida foi capaz de gerar expressões regulares para fazer o procura de informações nos arquivos de *log* e as exibir através da interface *web*.

Em Ghafir e Prenosil (2015), no qual também é exposta tanto a importância quanto a simplicidade de sistemas de lista de exclusão, é desenvolvido um sistema de capaz de detectar ameaças a rede através de *blacklists* que são atualizadas através do processamento de informações vindas de diferentes fontes e, no caso de encontrar algum *IP* que esteja na *blacklist* acessando a rede, é gerado um aviso ao administrador. Embora tenha uma função similar à da ferramenta desenvolvida nesse trabalho, auxiliar o administrador a encontrar ameaças dentro de rede, seu método de funcionamento é diferente.

As duas principais diferenças são as fontes o os formatos das informações, o sistema proposto neste trabalho pode exibir toda uma gama de informações vindas de diferentes bancos de dados online, não se resumindo somente a *IPs* que devem ser bloqueados, como exemplo temos em *ibm x-force exchange*, onde podemos obter o grau de ameaça que um *IP* representa, o motivo dele possuir tal grau de ameaça, quais outros *IPs* ele pode afetar,

informações sobre quem registrou o endereço e todo um histórico detalhado sobre o mesmo, mesmo assim, a *ibm x-force exchange* não é a única fonte de ameaças de onde a aplicação pode fazer pesquisas e a ferramenta aqui proposta permitirá a pesquisa em diversos bancos.

A segunda grande diferença se dá devido as aplicações suportadas, enquanto em Ghafir e Prenosil (2015**)** é usada a aplicação BRO para fazer uma análise em tempo real do tráfego da rede, fazendo uma busca em sua *blacklist* e então gerar os avisos, a ferramenta proposta neste trabalho pode trabalhar em conjunto com qualquer aplicação que gere arquivos de *log* contendo endereços *IPs* de origem provenientes de fora da rede, aumentando assim ainda mais sua área de atuação. Com as informações mostradas nesse segmento fica evidente que o trabalho aqui proposto se diferencia das atuais tecnologias de processamento de *logs* devido a sua grande adaptabilidade em relação a API*s* e ferramentas de segurança de rede, sendo assim, após implementada dentro de uma, será de grande ajuda ao administrador na busca de ameaças que passam desapercebidas pelas demais aplicações de segurança.

Ao final do trabalho foram elaborados testes da aplicação, a fim de verificar o correto funcionamento da aplicação. Uma vez que a aplicação foi alimentada com os endereços e começou a fazer o processamento do tráfego de rede a mesma conseguiu alertar os administradores de rede das possíveis ameaças.

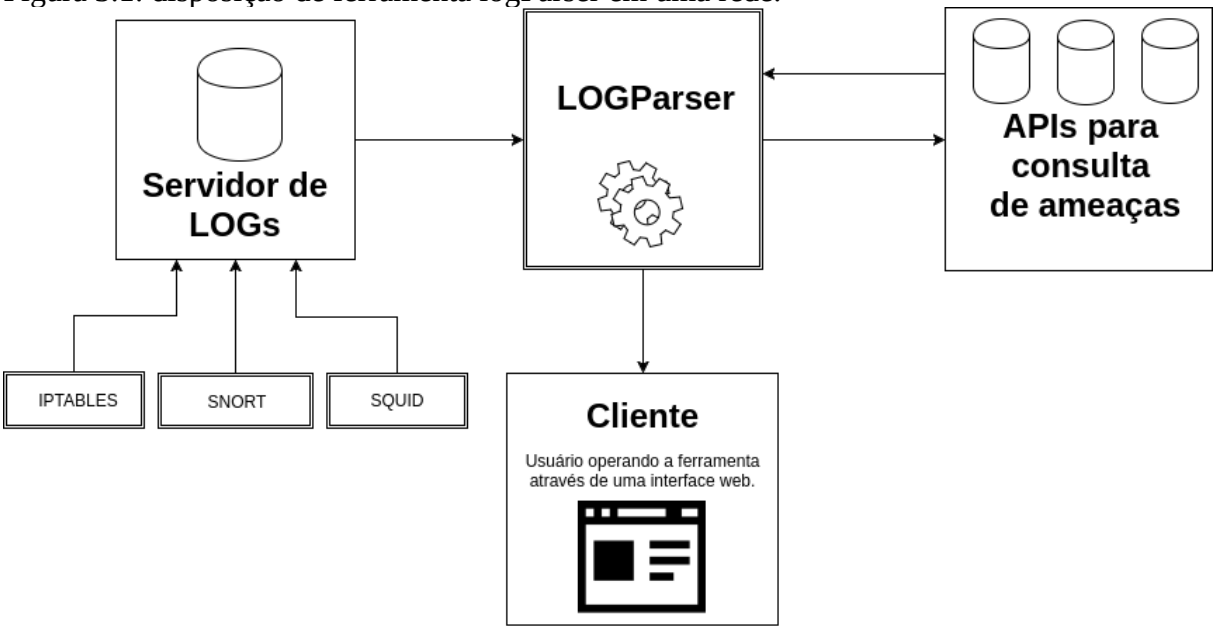
3 DESCRIÇÃO DA PROPOSTA

Através da busca e análise feita pela ferramenta será possível verificar quais são as principais ameaças encontradas nos *logs*, podendo após, por exemplo, facilitar ao administrador da rede a realização de bloqueios de ameaças ou tomar outras ações. A Ferramenta terá um sistema de módulos, que uma vez inseridos na aplicação vão permitir a leitura de *logs* de diversas aplicações, contanto que o *log* contenha informações que possam ser usadas como parâmetros de pesquisa em bancos de dados de ameaças ele poderá ser usado em conjunto com a ferramenta.

Outro ponto chave da ferramenta será a possibilidade de realizar consultas a *APIs* externas que contenham bancos de dados sobre ameaças, pois através da utilização de diversos bancos será maior a possibilidade de encontrar informações pertinentes a segurança da rede.

Na figura 3.1 podemos observar os pontos chaves em relação ao funcionamento da ferramenta:

Figura 3.1: disposição de ferramenta logParser em uma rede.



Fonte: arquivo pessoal

Servidores de *log* armazenam os arquivos gerados pelas ferramentas assim como podem realizar o seu processamento. Seu principal intuito é separar a aplicação de seus *logs*, tendo em vista que uma falha pode resultar na perda dos mesmos.

Em outro dispositivo, ou até mesmo no próprio servidor de *logs* será hospedada a aplicação de análise, ela será responsável por realizar o processamento desses arquivos de uma forma segura, lidando com diversos formatos de arquivos de *log* e suas diferentes sintaxes e formatos. Uma vez que a aplicação consiga ler esses arquivos ela precisará pesquisar informações em diversas fontes que novamente terão diversos formatos e sintaxes e por último ela precisará mostrar, através de uma interface web no qual o servidor será contida nela mesma por motivos de portabilidade, dados diversos e escolhidos pelos seus usuários.

Com essas funcionalidades, os principais objetivos da ferramenta são auxiliar o administrador de rede com a análise dos arquivos de *log*, que muitas vezes são usados somente após os problemas serem encontrados, como também, evitar que possíveis ameaças, que as outras ferramentas não conseguiram detectar, pois, ainda não apresentaram um comportamento suspeito, tenham acesso à rede.

4 IMPLEMENTAÇÃO DO SISTEMA PROPOSTO

Para o desenvolvimento dessa ferramenta foi utilizada a linguagem de programação *python*, a mesma foi escolhida por sua extensa documentação, boa portabilidade entre sistemas e seu código de fácil leitura e entendimento, em conjunto com a IDE *visual studio code*.

4.1 *HOST*

A aplicação deve ser utilizada em algum computador que tenha acesso aos arquivos de *log*. O acesso aos recursos poderá se dar de duas maneiras, uma sendo local, instalando-a junto com o arquivo de *log*, assim como remoto, onde a aplicação é instalada em um *host* e realiza o acesso a um servidor de *logs* através de uma chave pública e através de *SSH* em conjunto com *SCP* no caso de arquivos em texto plano. A aplicação também pode realizar consultas em bancos de dados que utilizem a tecnologia *mysql*.

4.2 INTERFACES *WEB*

A primeira parte desenvolvida foi a interface web, através da qual os módulos de *APIs* e comunicação com outras ferramentas são adicionados de forma que não seja necessário a edição do código fonte da própria aplicação. Para isso foi usado a biblioteca *flask* que permite a criação de um pequeno servidor *http*, que gerencia requisições e métodos do protocolo, através da mesma foi feito o gerenciamento da aplicação, como chamadas a métodos de consulta e de processamento.

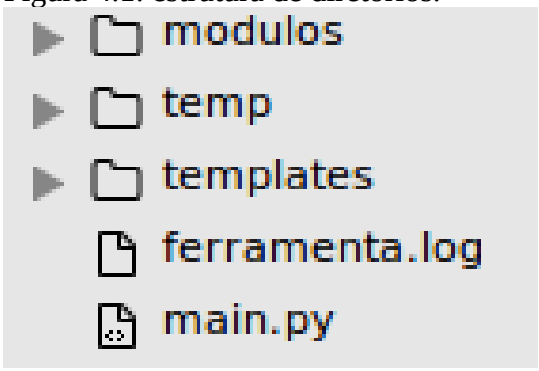
Em conjunto com essa ferramenta temos também *jinja2*, que ficou responsável pela renderização de *templates* em *HTML* para a exibição de dados através de *python*. Com ela foram elaborados os *templates* das telas que serão exibidas ao usuário.

4.3 INSTALAÇÃO E EXECUÇÃO

Para realizar o uso da aplicação logParser é necessário instalar o mesmo em um *host linux* que possa se comunicar com as *APIs* que serão usadas, para isso bastando uma conexão com a internet na maioria dos casos e consiga acessar os arquivos de log. Por último se faz necessário a instalação das bibliotecas *flask* e *PymySQL*, presente em todas as versões do

python através do gerenciador de pacotes *pip* ou *pip3*. Uma vez a ferramenta presente no sistema temos a seguinte estrutura de arquivos, como mostrada na figura 4.1:

Figura 4.1: estrutura de diretórios.



fonte: arquivo pessoal.

na qual o *python* deve ter permissão de escrita nas pastas “modulos” e “temp” assim como o arquivo “ferramenta.log”. Uma vez que as permissões estejam corretas a aplicação é iniciada com o comando:

```
python main.py
```

Vale observar que a aplicação foi projetada para funcionar com qualquer versão do *python*, no desenvolvimento não foram usadas funcionalidades que não estivessem presentes em ambas as versões. Após iniciado pode-se ver tanto no terminal quanto no arquivo “ferramenta.log” o que está ocorrendo na aplicação e podemos acessá-la no endereço *IP* de seu *host* na porta 5000 como é visto na figura 4.2:

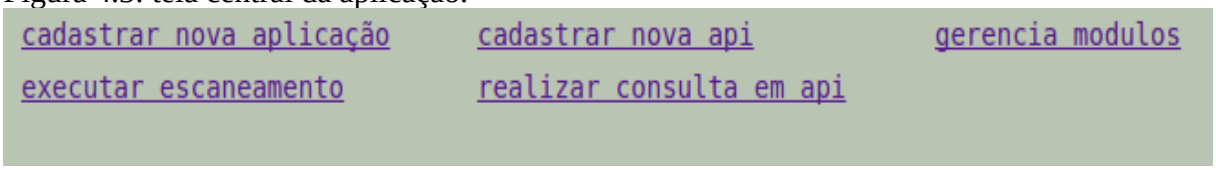
Figura 4.2: terminal após aplicação ser iniciada.

```
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 143-694-610
```

fonte: arquivo pessoal.

A aplicação é dividida basicamente em seis telas. A partir da tela inicial é possível acessar as demais, e é possível retornar a tela principal a partir de qualquer uma das telas como visto na figura 4.3:

Figura 4.3: tela central da aplicação.



Fonte: arquivo pessoal

4.4 CRIAÇÃO DE MÓDULOS:

Os módulos de comunicação com ferramentas funcionam de forma genérica, permitindo assim grande portabilidade ao logParser, para isso, foi previsto dois cenários comuns de armazenamento de arquivos de *logs*, o primeiro sendo em texto plano no qual cada linha do arquivo é uma entrada diferente. No segundo caso as informações do *log* estejam salvas em uma tabela de um banco de dados do formato *SQL*.

No primeiro caso, Via interface web os dados relevantes aos módulos são passados, entre eles temos:

→ Nome do módulo: uma *string* que guarda o identificador único do módulo.

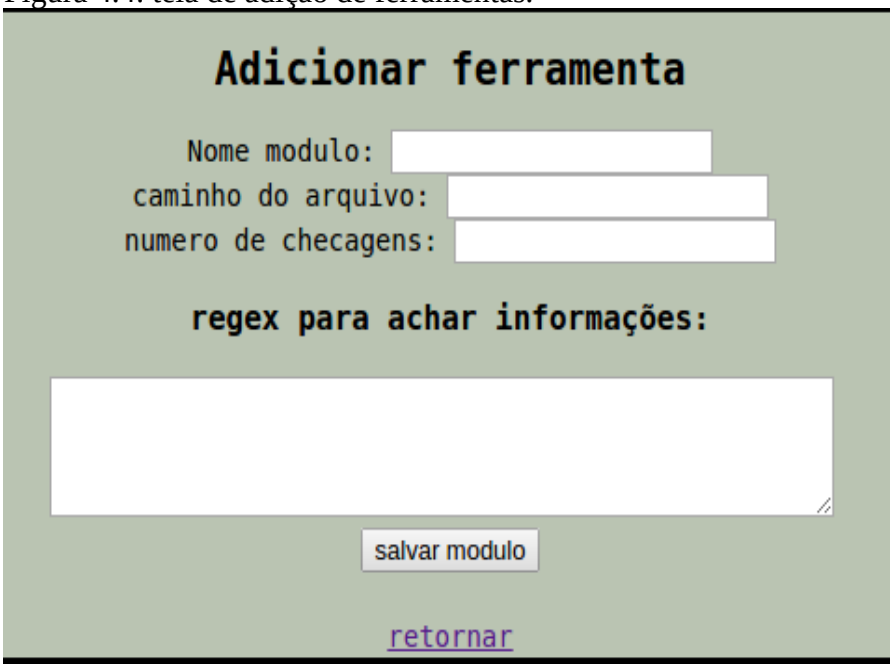
→ Local do arquivo: uma *string* que guarda a localização do arquivo dentro do servidor de *logs* remoto ou local, no caso da aplicação encontrar algum endereço *IP* tratará o mesmo como remoto e tentara fazer uma conexão para procurar o arquivo de *log*.

→ Número de checagens: em um período de 24 horas, quantas vezes o programa deve analisar os arquivos de log.

→ Regex para achar informações: expressão regular que extrairá informações do log.

O mesmo pode ser visto na figura 4.4:

Figura 4.4: tela de adição de ferramentas.



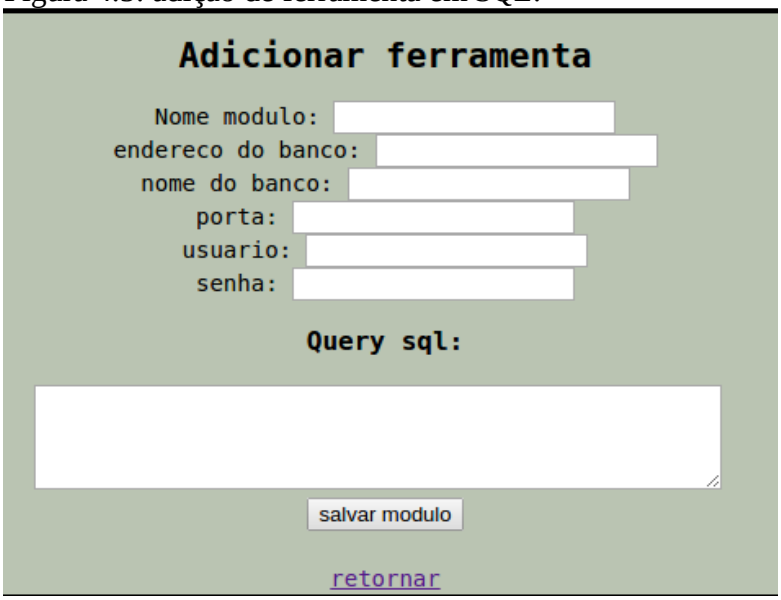
Fonte: arquivo pessoal.

No segundo caso, será extraído informações de um banco de dados *SQL*, para tal temos os seguintes dados:

→ nome do módulo: uma *string* que guarda o identificador único do módulo.
→ endereço de banco: endereço onde o banco de dados *SQL* está guardado.
→ nome do banco: nome do banco de dados.
→ porta: em qual o banco de dados está acessível.
→ usuário: usuário com permissão de leitura.
→ senha: senha usada na autenticação.
→ *query sql: query* utilizada para extrair informações do banco de dados.

Após preenchido o formulário, como na tela 4.5, a aplicação o salvará em forma de *json* dentro de um arquivo e o guardará na pasta de módulos.

Figura 4.5: adição de ferramenta em *SQL*.



Fonte: arquivo pessoal.

4.5 CADASTRO DE *APIS*

De forma similar aos módulos, o cadastramento de *APIs* é feito via interface web. Através desse cadastro será possível fazer pesquisas sobre endereços *IPs* extraídos dos *logs*. Através de um método *POST*, proveniente do HTTP, com diversos parâmetros. Após o retorno é analisado e informações úteis retornam ao usuário. Esse retorno geralmente sé dá através de *json* ou *xml*, como a sintaxe desse retorno pode variar de diversas formas fica a cargo do usuário adicionar informações de quais campos que ele deseja ser notificado.

Por padrão os seguintes campos serão necessários para o cadastro:

→ nome da *API:* uma S*tring* que guarda identificação única da *APIs*.

→ URL de consulta: uma *String* que guarda a *URL* que receberá o método *POST*.

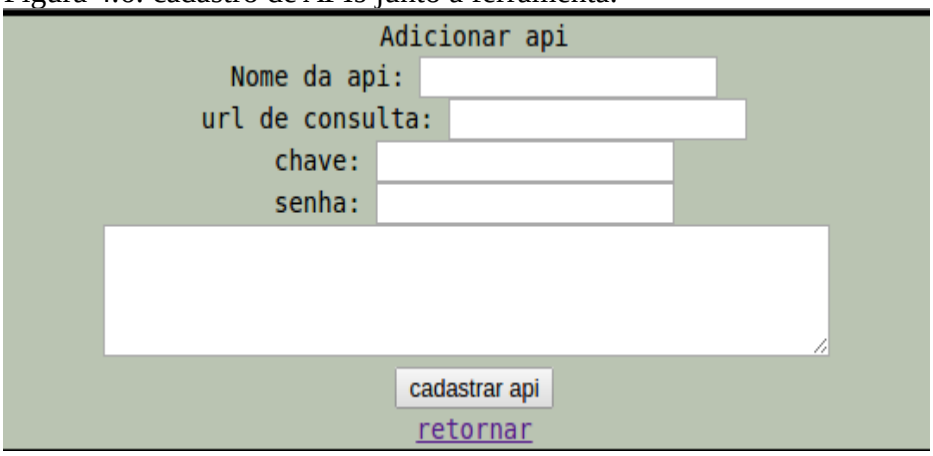
→ Chave: uma *String* contendo a chave ou login usado por algumas *APIs* para controlar as requisições feitas pelos usuários, caso esteja em branco será ignorada.

→ Senha: senha que será usada para autenticação, a mesma pode funcionar sem a chave e caso em branco será ignorada.

→ Campo de texto: *String* que guarda as expressões de consulta *json* ou *xml* que processarão o retorno da *API* e mostrará informações escolhidas pelo usuário, Caso esteja em branco ou contenha erros é exibido todo o retorno.

A respectiva tela pode ser vista na figura 4.6:

Figura 4.6: cadastro de *APIs* junto a ferramenta.



Fonte: arquivo pessoal

Conforme melhorias foram feitas durante o desenvolvimento da ferramenta foi possível remover a necessidade de identificar manualmente se uma S*tring* possui conteúdo em *JSON* ou *XML*, sendo assim, se for necessário enviar uma requisição a *URL* de consulta utilizando *JSON* basta colocar o campo entre parênteses e caso seja em *XML* colocá-lo entre sinal me menor e maior (< e > respectivamente) de modo a simular uma *tag* XML. *D*essa forma, a aplicação consegue reconhecer a sintaxe da *string*.

Nesta parte vários erros podem ocorrer a aplicação, sendo o mais grave deles uma falhas no login as APIs. Por exemplo, ao usar a biblioteca *requests* do *python*, tendo uma *API* X, na qual o usuário se autentica com uma requisição contendo:

```
{
  “user”:”usuario” ,
  “pass”:”senha” ,
  “data”:”dados a serem pesquisados”
}
```

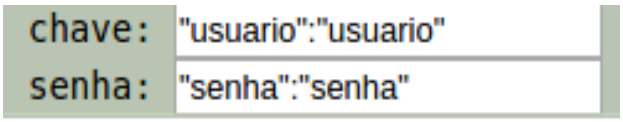
Se comporta diferente de uma *API* Y onde teríamos s seguinte formato:

```
{
  “key”:”hash” ,
  “data”:”dados a serem pesquisados”
}
```

Para amenizar esse problema foram implementadas duas funcionalidades, a primeira é redirecionar toda a informação de *debug* do console do *python* para o usuário, assim, o mesmo poderá encontrar erros e caso possível editar diretamente o arquivo de módulo que esta salvo

na pasta “*modulos*”. A segunda funcionalidade foi permitir a reescrita dos parâmetros de envio. Como na interface web temos os campos “chave” e “senha”, que são enviados nas requisições com os *keywords* “*key*” e “*pass*”, porém caso necessário pode-se colocar entre parênteses um novo *keyword* que substituirá o antigo, como no exemplo mostrado na figura 4.7:

Figura 4.7: alterações nos padrões da ferramenta.



Fonte: arquivo pessoal

Após os campos preenchidos a informação é transformada em um *JSON* e salva em um arquivo de texto plano especifico com o mesmo nome do campo “Nome da api” acrescido de uma extensão jso.

4.6 EXCLUSÃO E MODIFICAÇÃO DE *APIS* E MÓDULOS:

O principal motivo da locação permanente dos módulos e APIs em arquivos de texto foi para um melhor gerenciamento dos mesmos, por exemplo, para apagar um módulo pode-se tanto apagá-lo via interface web quanto direto na pasta onde ele está, pois, como esses módulos só são acessados quando usados e a aplicação não guarda nenhuma referência a eles exceto quando estão sendo usados e assim erros de execução são evitados.

Através da interface web tanto módulos quanto *APIs* são listados, quando o usuário selecionar algum dos módulos serão apresentadas as seguintes opções sobre o mesmo:

→ apagar: apaga o arquivo de sua respectiva pasta.

→ editar: abre o arquivo na interface web para edição.

Essas opções, assim como os nomes dos módulos e *API*s podem ser vistos na figura 4.8:

Figura 4.8: gerenciamento de módulos e APIs.

Fonte: arquivo pessoal

4.7 REALIZAÇÃO DE CONSULTAS:

Após a inserção de *APIs* e módulos de comunicação na tela principal da aplicação é encontrado a opção para fazer as consultas. Através de ação do usuário a ferramenta realiza a execução de todos os arquivos de módulos junto a ferramenta, primeiramente procurando se existe um arquivo da última execução de um módulo, se esse arquivo existir a aplicação lerá a última linha, onde fica salvo o último tamanho do arquivo, ou seja, o tamanho que o arquivo tinha antes da sua última execução. Graças a essa informação é possível pegar os últimos bytes do arquivo de forma menos custosa computacionalmente graças a integração *do python* em conjunto com a aplicação *Linux tail*.

Com a expressão regular especificada é feito a extração de informações do arquivo de *log*. Após os mesmos são então processados, onde é feita a remoção de duplicatas baseado na comparação de strings e então salvos em uma lista, junto com o tamanho atual do arquivo de *log*, tal lista substituirá a antiga lista que após será guardada em um arquivo de texto plano.

Após teremos a opção de realizar consultas, para isso temos a opção “realizar consulta em api” na tela inicial. Ao selecionarmos essa opção temos todas as listas que foram processadas, e ao lado delas temos a opção de selecionar em qual *API* queremos realizar a busca, tal metodologia foi usada para permitir uma melhor separação e controle das consultas e permitir o uso de uma *API* junto com vários módulos.

Ao selecionarmos uma *API*, é feita uma leitura linha por linha do arquivo proveniente do processamento do *log*, ao mesmo tempo é feita uma consulta de cada linha junto a *API*

selecionada, na qual as partes selecionadas de seu retorno são salvas em um arquivo temporário na pasta “temp”, por último essa informação é exibida ao usuário.

4.8 ANÁLISE E EXIBIÇÃO DOS DADOS:

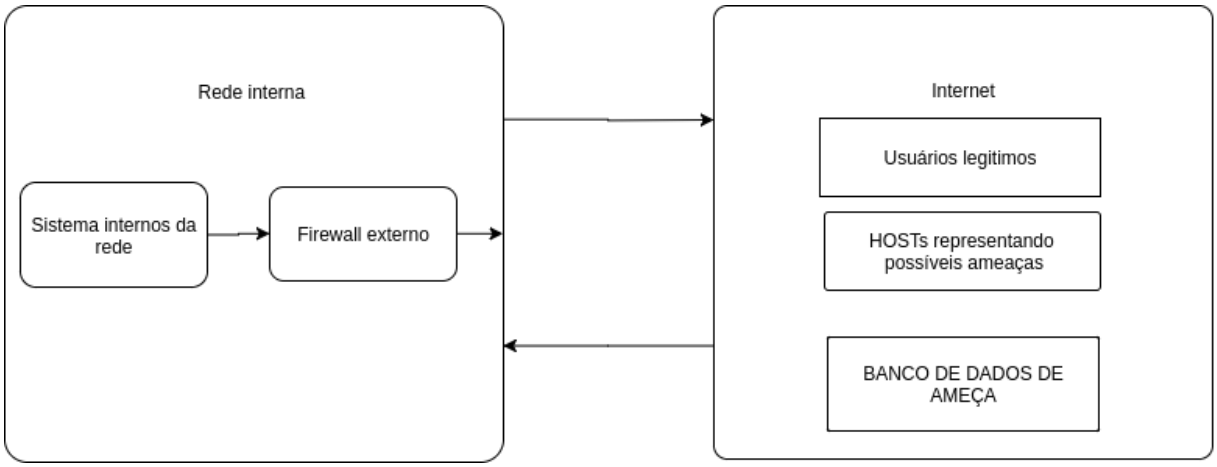
Após as consultas os bancos de dados de ameaças serem realizadas as informações devem ser exibidas ao usuário, de forma que, somente informações escolhidas previamente como úteis sejam exibidas. Para isso ao criar um módulo de pesquisa deve ser inserido informações sobre quais campos devem ser exibidos, tanto no formato de *JSON* ou de *XML*, sendo assim, as informações desnecessárias são descartadas ao serem exibidas.

Muitos fatores tornam tal modo de operação necessário, sendo o principal a diferença nas informações obtidas entre bancos de dados, podemos tomar como exemplo duas API*s* distintas, “A” e “B”, enquanto a “A” fornecem informações sobre reputação de endereços *IPs* em relação a ataques com informações guardadas e transmitidas em formato de *XML* a “B” oferece informações sobre endereços conhecidos pelo envio de spam, transmitindo informações em *JSON*. Outro ponto importante é a disposição interna dessas informações, onde mesmo com duas *APIs* distintas, mas que porém transmitem dados utilizando a mesma tecnologia (*JSON* ou X*ML*), terão campos completamente diferentes.

5 TESTES

Para comprovar o funcionamento da ferramenta foi criado dois cenários, o primeiro com dados provenientes da comunicação entre duas redes simuladas, com uma delas simulando a internet e a outra uma rede corporativa como exemplificado na figura 5.1:

Figura 5.1: cenário de testes.



Fonte: arquivo pessoal.

Onde a rede “rede interna” simula uma rede corporativa, com um servidor de aplicação de *ERP* acessível a rede “internet”. A rede conta com um “firewall externo”, o mesmo com uma implementação das aplicações *squid*, *iptables*, *snort*, e a aplicação logParser.

A rede “internet” seriam compostas de máquinas que fazem acesso aos serviços internos da rede, ela se divide entre três tipos de *host*; usuários legítimos seriam os que usariam recursos da “rede interna” de forma prevista, como por exemplo, acessando um servidor de aplicação; “hosts representando possíveis ameaças” seriam *host* que tentariam fazer acessos indevidos a rede, como por exemplo, um escaneamento de vulnerabilidades; o “banco de dados de ameaça” guardaria informações sobre a reputação dos *host* contidos na rede “internet” e simula uma *API*, permitindo consultas vindas da “rede interna”.

Tal metodologia de testes foi pensada levando em consideração que a grande maioria de bancos de dados de vulnerabilidades implementam autenticação de usuário, de forma a limitar o número de requisições feitas diariamente. Sendo assim, mesmo durante o processo de desenvolvimento foi necessário a criação de uma *API* local devido as execuções de testes e correções de erros.

5.1 CENÁRIO DE TESTES COM *API* SIMULADA

O primeiro passo dado na realização dos testes da ferramenta foi checar seu completo funcionamento, para isso, foi escolhido simular todo o ambiente localmente, onde cada um dos pontos de rede é composta por uma máquina virtual rodando o sistema operacional Ubuntu S*erver* 14.04.05, possuindo 1 *GB* de memória *RAM* e memória de 8 *GB*, no cenário também é representada duas redes distintas em duas faixas diferentes de *IP*, onde a faixa 192.168.0.0 representa a rede interna e a faixa 172.20.0.0 representa a rede internet.

Para a realização dos testes foi desenvolvida uma *restfull API* que tem como *backend* um *script* em *PHP* que está alocado em um servidor apache2 no endereço *IP* 172.20.2.51, a mesma aceita três parâmetros em uma requisição *http*, usuário e senha, que são os usuários cadastrados junto a *API* e um endereço *IP* que deverá ser pesquisado no banco de dados simulado. No caso de não existir usuário ou a senha estiver incorreta a requisição é descartada.

No banco de dados é guardado o endereço *IP* junto com informação de reputação de zero a cinco, sendo zero livre de risco e representando um usuário legítimo e cinco de alto risco e uma entrada do nome do provedor ao qual o endereço *IP* está registrado. No cenário caso o grau de risco seja maior que zero o *host* é considerado ilegítimo. Os *IPs* guardados estão dispostos em uma tabela como a tabela 5.1:

Tabela 5.1: estrutura interna dos dados contidos na *API* simulada.

ID	ENDEREÇO	GRAU DE RISCO	INFO
0	172.20.2.3	0	Provedor um
1	172.20.2.4	1	Provedor um
2	172.20.2.5	3	Provedor dois
3	172.20.2.8	5	Provedor três

Com esse formato de dados, caso uma requisição seja feita corretamente utilizando como parâmetro de busca o *IP* 172.20.2.2 ela retornaria a seguinte informação:

```
{
        "ip": "172.20.2.2",  "grau": "2",  "info": "\"provedor x\""
}
```

Fazendo a conexão entre as duas redes, está o “firewall externo”, o mesmo é composto pelo *iptables*, o servidor de *proxy Squid*, e o *IDS snort*, os mesmos estão configurados para somente registrar os pacotes que passam, registrando os arquivos de *log* localmente. Nesse mesmo servidor está instalada a aplicação *log* parser.

Na parte interior da rede, em um *host* diferente, está instalado a aplicação de *ERP Odoo* 8 que gera arquivos de *log* com endereços *IPs* de todos que fazem acesso ao mesmo. Essa aplicação salva os arquivos de *log* localmente, e através dela será testada a capacidade da ferramenta trabalhar com arquivos remotos.

5.2 TESTES COM API SIMULADA

A aplicação foi projetada para funcionar tanto com *python* 2 quanto *python* 3, nos testes foi-se usado *python* 2. Após, no terminal usado para iniciar a aplicação são exibidas informações sobre o funcionamento da aplicação, como pode ser observado na figura 5.2, entre elas, quais páginas estão sendo acessadas, quais requisições estão sendo feitas e a quais *APIs*, ela possui também a capacidade de exibir erros ocorrentes em geral.

Figura 5.2: ferramenta iniciada.

```
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 139-770-157
127.0.0.1 - - [20/Jun/2017 20:14:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2017 20:14:23] "GET /templates/stilo.css HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2017 20:14:23] "GET /favicon.ico HTTP/1.1" 404 -
```

Fonte: arquivo pessoal.

Com a aplicação em estado de operação, foram seguidos os seguintes passos; criação de módulos para extrair endereços *IPs* dos *logs* provenientes das aplicações locais e remotas, extração desses endereços, consulta contra a *API* e exibição de dados.

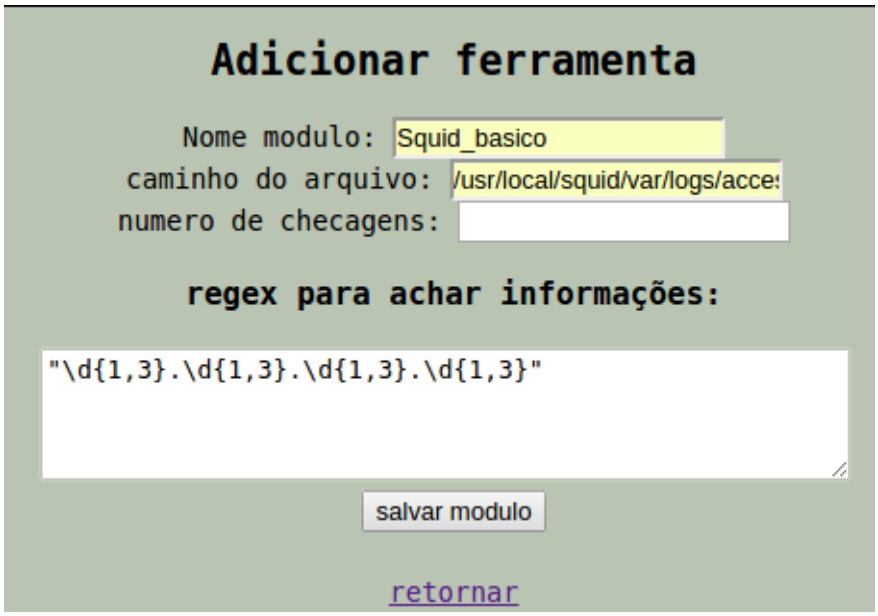
5.3 ARQUIVOS LOCAIS EM TEXTO PLANO

No primeiro teste foi processado um arquivo de *log* que se encontrava localmente no servidor, em formato de texto plano, proveniente da aplicação *Squid*, em busca de endereços *IP*. Ao acessar a interface web da aplicação foi escolhido “adicionar nova aplicação”, na tela seguinte “log em texto plano” e após foi adicionado as informações necessárias para o funcionamento do módulo, onde os campos que ficaram em branco serão ignorados no momento em que a informação é salva em arquivos permanentes. Vale ressaltar que a expressão regular contido nesse módulo funciona de forma geral para a extração de possíveis endereços *IPs*, capturando tanto *IPs* validos quanto inválidos, contanto que sigam o padrão seguinte onde X representa um número de 0 a 9:

XXX.XXX.XXX.XXX

Tal padrão foi concebido a partir da obervação do arquivo de *log*, pois mesmo ele aceitando valores de *IPs* incorretos, não foram encontradas ocorrências que não representassem um endereço *IP* correto no arquivo de log. Embora seja possível criar uma expressão regular que faça uma melhor filtragem deve ser levando em consideração o custo computacional. Na figura 5.3 é possível o preenchimento do formulário de adição de módulo:

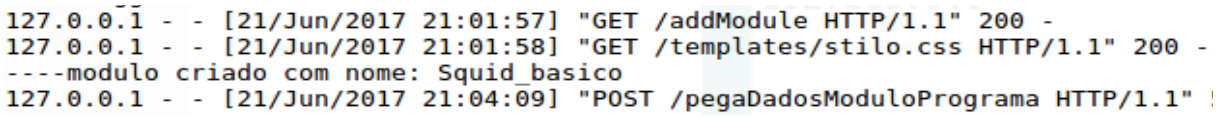
Figura 5.3: ferramenta sendo adicionada.



Fonte: arquivo pessoal.

Na figura 5.4 é possível ver na figura que o módulo foi adicionado corretamente:

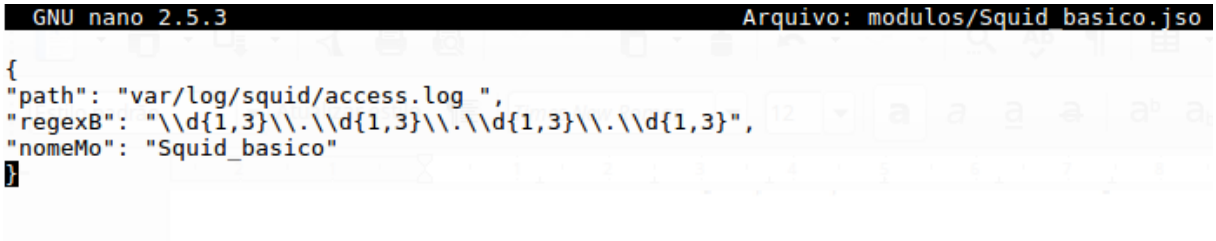
Figura 5.4: confirmação da criação do módulo.



Fonte: arquivo pessoal.

Além disso se percebe na pasta de módulos a criação do arquivo contendo o módulo como visto na figura 5.5:

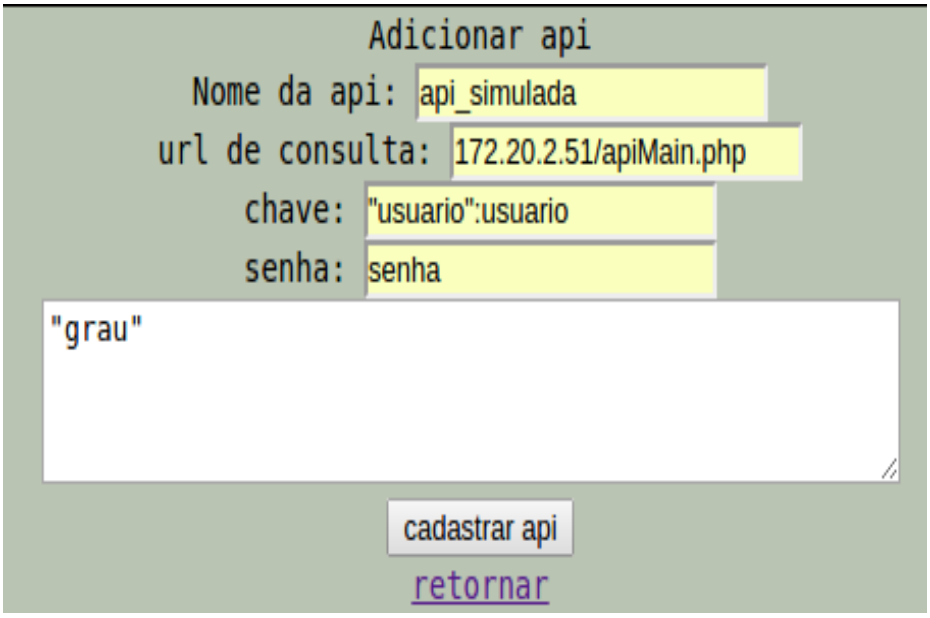
Figura 5.5: interior do arquivo de módulo.



Fonte: arquivo pessoal.

Após é realizado o cadastro da *API* simulada, como visto na figura 5.6, a mesma estava hospedada no endereço 172.20.2.51 na página “apiMain.php”. Nesse teste foi buscado informações referentes ao campo “grau” do *JSON* retornado pela *API*, caso esse campo fosse deixado em branco seria exibido todo o *JSON* retornado pela *API*, assim como, podem ser exibidos diferentes dados caso os campos sejam separados por vírgula. Uma imagem mostrando a criação da *API* pode ser vista na figura 5.6:

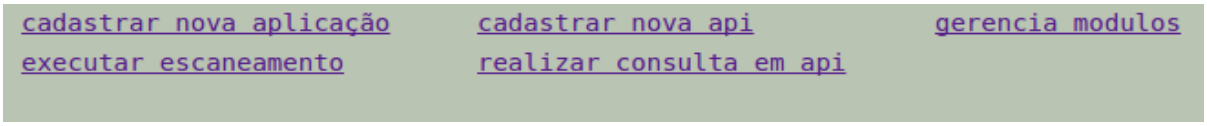
Figura 5.6: adicionando uma *API* via interface *web*.



Fonte: arquivo pessoal.

Ao clicar no link “executar escaneamento” todos os arquivos de módulo são executados como exemplificado nas figuras 5.7 e 5.8:

Figura 5.7: tela central da interface web.



Fonte: arquivo pessoal.

Figura 5.8: dados obtidos sendo salvos em arquivo temporário.

```
inicio do modulo:Squid_basico.jso
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid_basico.temp
dado obtido: adicionado ao:Squid basico.temp
```

Fonte: arquivo pessoal.

Em um primeiro momento é possível ver no terminal toda vez que uma informação é obtida. Ao verificar o arquivo criado na pasta “temp” com o nome de “Squid_basico.temp” podemos ver os *IPs* guardados, assim como na última linha o tamanho total do arquivo, em *bytes*, que foi extraído a partir de uma função em *python*. Tal arquivo não contém informações repetidas, devido a uma remoção previa baseada na comparação de strings. O conteúdo do mesmo pode ser visto na figura 5.9:

Figura 5.9: arquivo temporário mantido entre execuções.

Fonte: arquivo pessoal.

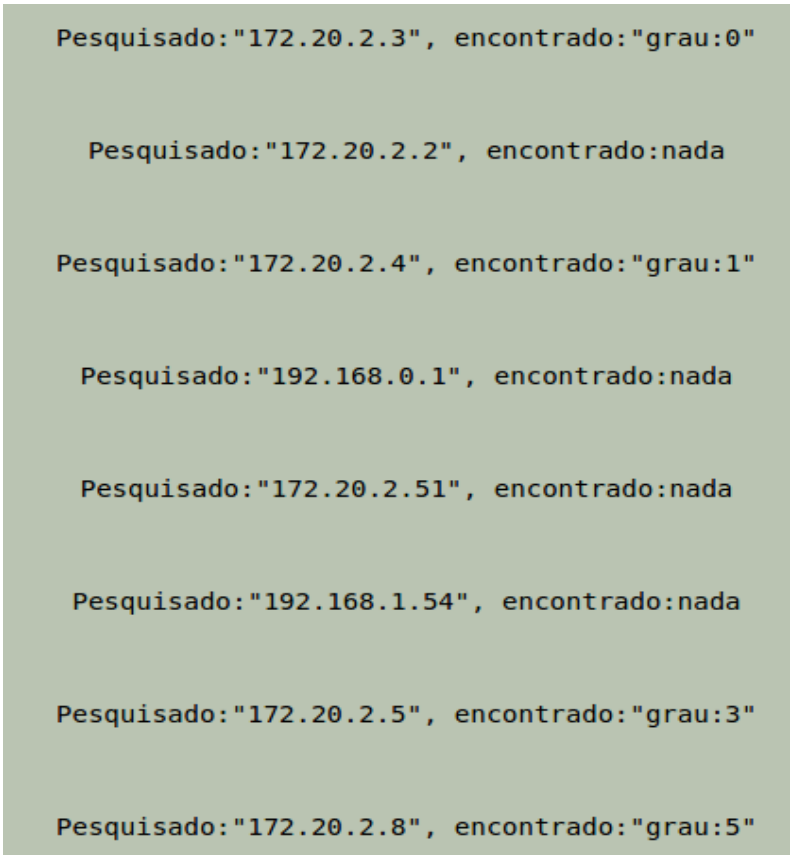
O segundo passo, como mostrado na figura 5.10 é a realização da consulta junto a *API*, para isso deve ser selecionada a saída do processamento do módulo e após seleciona-se em qual *API* deve-se pesquisar os mesmos:

Figura 5.10: busca de lista em *API*.

Fonte: arquivo pessoal.

Por último a informação precisa ser exibida ao usuário, para isso, de uma forma mais completa são exibidas todas as informações encontradas durante a pesquisa, como mostra a figura 5.11.

Figura 5.11: resultado obtido no teste com *APIs* simuladas.



Fonte: arquivo pessoal.

5.4 ARQUIVOS DE *LOG* EM TEXTO PLANO EM SERVIDOR REMOTO

No próximo teste os arquivos de *log* se encontravam em outro servidor, no caso, o mesmo estava no endereço 192.168.1.54, e nele estava rodando a aplicação de *ERP Odoo* 8. O arquivo de *log* do *Odoo* 8 é simples, porém nos dá informações sobre endereços *IPs* que estão fazendo acesso a aplicação.

De forma similar ao teste anterior foi criado um módulo, porém, dessa vez foram necessários alguns passos no servidor para a geração de um certificado de forma a permitir um login no servidor remoto e a realização da cópia dos arquivos de *log* através do comando *scp* sem a utilização de senhas.

No primeiro passo, foi gerado um certificado com o comando *ssh-keygem* como visto na figura 5.12:

Figura 5.12: geração de chave.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): /etc/certificado
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /etc/certificado.
Your public key has been saved in /etc/certificado.pub.
The key fingerprint is:
SHA256:7zUeq3ZExjVQgtGp7Kud+Unu7jeaRrWNCdBVaTLo7Kw root@ironcastle
The key's randomart image is:
+---[RSA 2048]----+
|           .=+=o.o|
|           o.++oo |
|          .o+ .+. |
|           oo= .  |
|         S.oo o = |
|          ..oo + .|
|           o+=    |
|          EoBo*o  |
|         .o*O%o . |
+----[SHA256]-----+
```

Fonte: arquivo pessoal.

Após, com o comando abaixo, o mesmo deve ser copiado para o servidor de onde serão retiradas os *logs* das aplicações:

```
ssh-copy-id usuario@host
```

A aplicação foi implementa dessa forma levando em consideração que, diferentemente dos arquivos gerados para guardar informações sobre *APIs*, que podem conter informações sobre senhas em texto plano, principalmente para facilitar seu compartilhamento, senha e logins de usuários de servidores são informações que devem sem protegidas, com esse método é possível manter a facilidade de compartilhar módulos, uma vez que esse módulo não se diferencia de um módulo para *logs* locais

Uma vez que a aplicação esteja conectada ao servidor ela realizara os mesmos procedimentos que realizaria caso fosse um arquivo local, porém, para evitar a necessidade da instalação de um cliente no servidor o processo de busca, comparação e cópia do arquivo serão feitos com comandos nativos do Linux. O primeiro comando é:

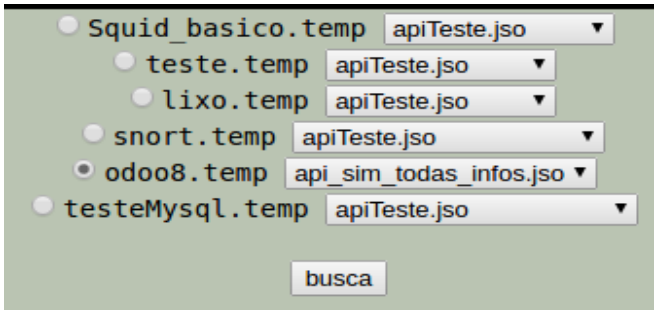
```
stat -c %s nome_do_arquivo
```

O mesmo retorna o tamanho em bytes. Caso necessário a aplicação pega os últimos bytes do arquivo com o comando abaixo, e após os envia para o local onde logParser está instalado:

```
tail -c $tamanho $nome_do_arquivo
```

Após o processamento novamente temos a opção de realizar as consultas selecionando a *API* desejada, abaixo foi selecionada o arquivo “odoo8.temp” para ser usada como pesquisa na “api_sim_todas_info.jso”:

Figura 5.13: busca de lista em *API*.



Fonte: arquivo pessoal.

E por último podemos ver na figura 5.14 um trecho dos resultados, dessa vez todas as informações da *API* foram recolhidas:

Figura 5.14: resultados obtidos.

```
Pesquisado:"172.20.2.3", encontrado:"grau:0","info:Provedor um"

Pesquisado:"172.20.2.4", encontrado:"grau:1","info:Provedor um"

Pesquisado:"172.20.2.5", encontrado:"grau:3","info:Provedor dois"
```

Fonte: arquivo pessoal.

5.5 TESTES BUSCANDO INFORMAÇÕES EM BANCOS DE DADOS

Para contemplar um cenário onde os *logs* são guardados em bancos de dados que utilizem de sql, a parte da aplicação responsável pela adição de módulos foi dividida em duas, ficando uma parte responsável pelo processamento de arquivos em formato de texto e a outra para consultas em bancos de dados.

O banco de dados foi criado utilizando MySQL e foi hospedado localmente sobre o nome de “bancoteste” na tabela “logteste”, o mesmo possuía somente informações de endereços *IPs* que realizaram acessos ao servidor, sendo assim, haviam informações repetidas dentro do mesmo. Como visto na figura 5.15 o módulo é adicionado a aplicação:

Figura 5.15: adição de ferramenta.

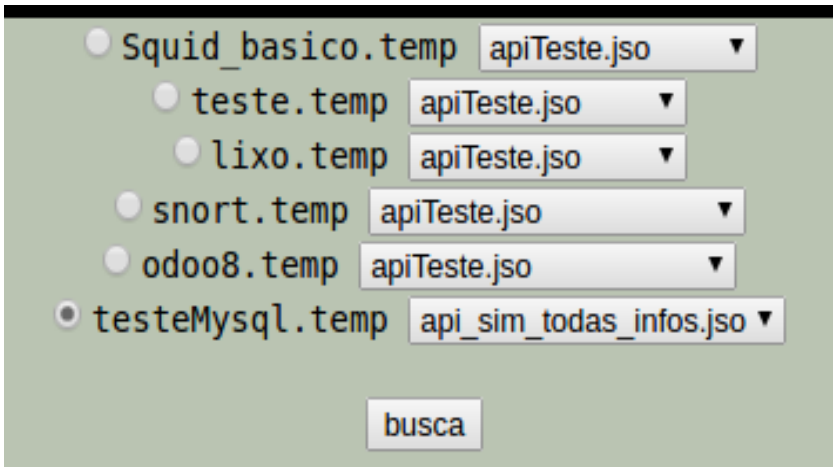
Fonte: arquivo pessoal.

A aplicação usa como conector ao banco de dados a biblioteca PyMySQL e embora ela aceite *queries* mais complexas ela só pode fazer o processamento de um valor por retorno, nesse caso *queries* como a abaixo não seriam processadas:

“SELECT ip, data FROM logteste”

Após executar o processamento do módulo for feita uma pesquisa na *API* simulada, selecionando a lista “testeMysql.temp” como exemplificado na figura 5.16:

Figura 5.16: busca de informações em API.



Fonte: arquivo pessoal.

Por último podemos ver os resultados na figura 5.17:

Figura 5.17: resultados da consulta.

```
 Pesquisado:"172.20.2.3", encontrado:"grau:0","info:Provedor um"

 Pesquisado:"172.20.2.4", encontrado:"grau:1","info:Provedor um"

Pesquisado:"172.20.2.5", encontrado:"grau:3","info:Provedor dois"

Pesquisado:"172.20.2.8", encontrado:"grau:5","info:Provedor três"
```

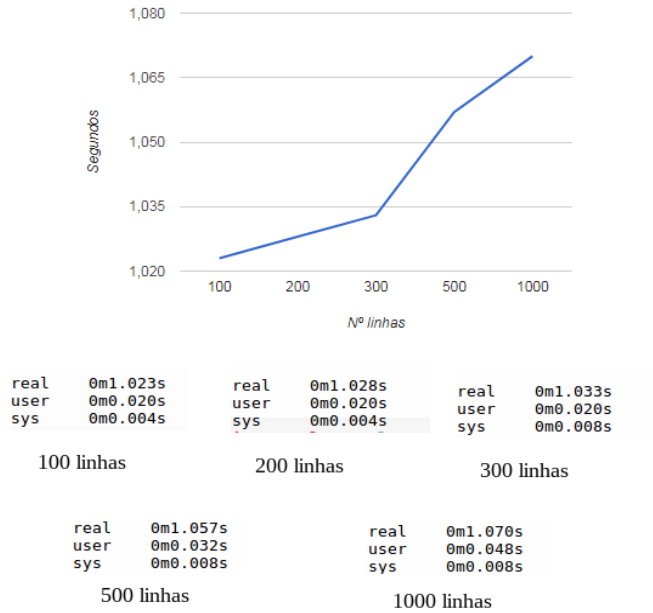
Fonte: arquivo pessoal.

6 ANÁLISE DOS RESULTADOS

Pode se observar no teste proposto em 5.2.1 e 5.2.2 uma deficiência importante relativa ao cenário de testes onde se é buscado informações de endereço *IP*, é o fato de que a expressão regular usada para a extração do arquivo de *log* não fez uma distinção sobre endereços internos e externos da rede. Entretanto percebe-se que a aplicação não falhou tanto na obtenção desses dados quanto em sua pesquisa, retornando ao usuário somente informações relevantes.

Para a obtenção de dados mais precisos, nos testes 5.2.1 e 5.2.2, seria necessário a criação de uma expressão regular mais complexa, por exemplo, pegando todos os *IPs*, menos os que começam com o prefixo de rede interna. Porém antes da elaboração de tal expressão seria necessário levar em consideração o tempo gasto com uma pesquisa. No teste representado na figura 6.1 foi medido o tempo de execução da parte da aplicação responsável por aplicar a expressão regular em uma linha do arquivo, usando a função *time proveniente* do *python:*

Figura 6.1: Medição de tempo.



Fonte: arquivo pessoal.

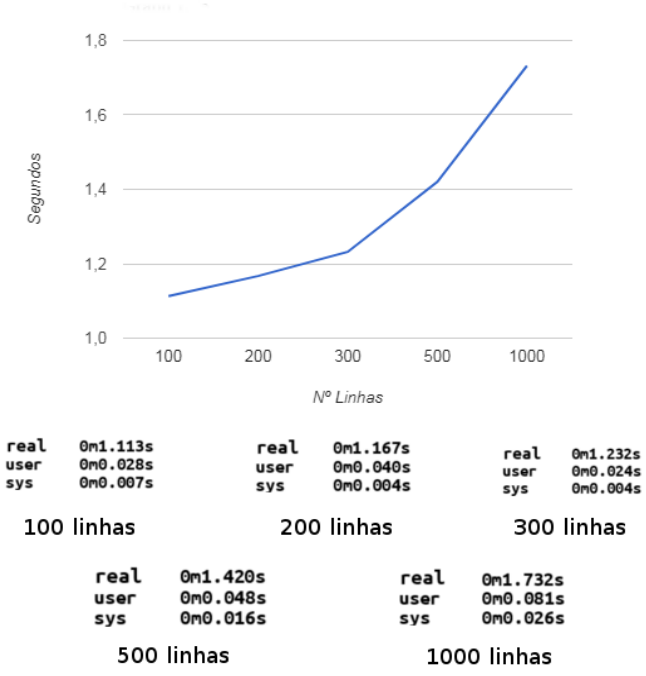
Na figura 6.1 percebe-se que o tempo não aumenta significativamente, isso se dá devido à simplicidade da expressão regular usada. Com a mesma procurando somente por padrões numéricos que se assemelham a endereços IPs valores incorretos podem passar desapercebidos, entretanto uma expressão regular capaz de fazer essa checagem seria mais custosa em questão de tempo.

Ao utilizar uma expressão regular mais complexa, seria possível realizar uma melhor filtragem dos arquivos de log. Como exemplo temos uma expressão regular capaz de detectar endereços *IPs* válidos e descartar endereços inválidos:

```
^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$
```

Embora o aumento de tempo não parece significante, visto que o tempo total de execução aumentou em poucos milissegundos, como mostrado na figura 6.2, temos que ter em vista que os testes foram feitas com poucas linhas de log. Ao escalarmos o número de linhas para um valor mais próximo do real, dos arquivos de *log* encontrados em servidores, teremos uma diferença ainda maior.

Figura 6.2: Tempo de pesquisa.

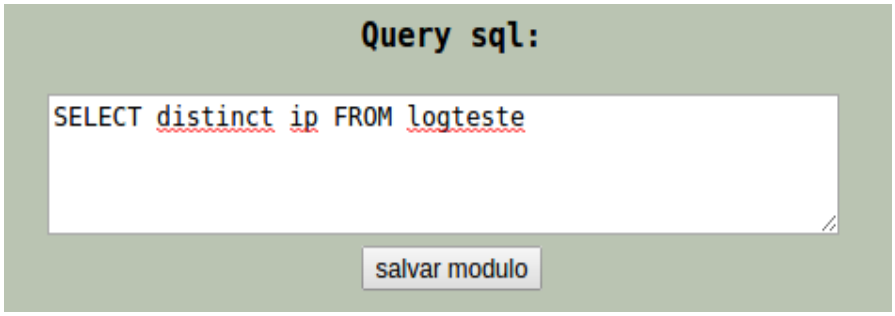


Fonte: Arquivo pessoal.

Por último no teste 5.2.3 foram encontrados dois erros graves, o primeiro é a falta de conectores com diferentes bancos de dados, como por padrão foi implementado somente a conexão com bancos de dados MySQL a aplicação não vai conseguir se comunicar nativamente com bancos de dados guardados em *sql-server*, *mysql* ou *mongoDB*, por exemplo, sem fazer alterações significativas no código fonte da aplicação. Porém, a aplicação funciona corretamente na extração de dados.

O segundo erro é a falta do processamento de campos contendo a data da entrada do *log*, fazendo com que a aplicação tenha que fazer uma releitura de todas as informações do banco. Entretanto tal tarefa é atribuída ao SGBD, cabendo a aplicação somente um pós-processamento em busca de duplicatas embora esse mesmo passo possa ser feito através de *queries SQL* como a abaixo, que remove duplicatas no momento da consulta:

Figura 6.3: *query SQL* melhorada.



Fonte: arquivo pessoal.

7 CONCLUSÕES E TRABALHOS FUTUROS

De um ponto de vista geral a aplicação cumpre com o que foi proposto; a pesquisa de informações sobre acontecimentos registrados em arquivos de *log*, a mesma funcionou de forma satisfatória, porém, para manter sua adaptabilidade aos *logs* de diversas ferramentas algumas funcionalidades que afetam principalmente sua performance teve de ser sacrificadas.

Durante o desenvolvimento desse trabalho foram encontrados diversos desafios e embora a ferramenta consiga fazer praticamente todas as funções que propôs, percebe-se que é possível haver melhorias, mas que porém, fogem do escopo desse trabalho.

Um dos pontos mais marcantes desse trabalho se encontra na parte de pesquisa de informações, atualmente é feito um processamento completo do arquivo de *log* e somente depois se é feita a pesquisa. Percebe-se então que seria muito mais proveitoso fazer um processamento, assim como consultas em APIs, em tempo real. Porém encontramos uma série de desafios, sendo o primeiro o limite de requisições diárias implementadas pela maioria das APIs. Uma vez que esse problema seja resolvido chegamos a outro ponto, o tempo decorrido na realização dessas requisições, onde com grandes quantidades de dados irá ser percebível o surgimento de gargalos.

Um importante ponto, em vista de tornar a aplicação proveitosa a um administrador de rede, seria permitir a aplicação entender quais são os retornos das APIs, atualmente a aplicação só exibe ao usuário as informações, porém caso ela soubesse valores aceitáveis ou não para um campo, ações poderiam ser tomadas por ela automaticamente. Por exemplo, como executado nos testes obtemos o grau de risco que um *IP* representa variando de zero a cinco, seria possível que caso fosse encontrado algum *IP* com grau de risco superior a quatro o endereço fosse bloqueado pelo *firewall*.

REFERENCIAL BIBLIOGRÁFICO

KENT K, Murugiah S. **Guide to Computer Security Log Management.** Disponível em:<http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf> Acesso em: 23 maio 2017.

VALDMAN J. **Log File Analysis.** Disponível em:<https://www.kiv.zcu.cz/site/documents/verejne/vyzkum/publikace/technicke-zpravy/2001/tr-2001-04.pdf> Acesso em: 19 maio 2017.

AERI A, T. **A comparative study of network based system log management tools.** Disponível em:<http://ieeexplore.ieee.org/document/7218075/> Acesso em: 16 maio 2017

McKINNEY W. P**ython for Data Analysis** ,1.ed, Ed.O'Reilly Media, Inc, Wes McKinney 2012

RICHARDSON L, RUBY S, AMUNDSEN M. **RESTful web APIs, services for a changing world ,**1.ed, Ed.O'Reilly Media, Inc, 2012

FRANKLIN D. **A Gentle Introduction to the X-Force Exchange API, 2015.** Disponível em:<https://securityintelligence.com/a-gentle-introduction-to-the-x-force-exchange-api/> Acesso em: 20 maio 2017.

JSON.ORG,**Introducing JSON, 2015.** Disponível em:<http://www.json.org/> Acesso em: 24 mar. 2017.

BEAULIEU A. **Learning SQL, second edition,**2.ed.,Ed.O'Reilly Media, Inc, 2009

JAMES F. KUROSE, K. W. R. **Computer Network A Top-Dow Approach.** 6th.ed. USA: NOVATEC Co., Inc., 2016.

TANENBAUM, A. S. **Computer networks.** 4. Ed. Pearson, 2011

BROWN L, STALLINGS W. **Computer Security: Principles and Practice**. Ed. Elsevier Editora Ltda. 2. edição, 2014

SAINI K. **Squid Proxy Server 3.1: Beginner's Guide**. Ed. PACKT publising, 2011

WESSELS D. **Squid: The Definitive Guide,** 1.ed.,Ed.O'Reilly Media, Inc, 2009

ELLEN S. FIGGINS S. LOVE R. **LINUX in a nutshell a desktop quick reference,** 6.ed, Ed.O'Reilly Media, Inc, 2009

SPIKE E. TWEED A. ROUSE L. **Strategic Cyber Threat Intelligence Sharing: A Case Study of IDS Logs,** Disponível em:<http://ieeexplore.ieee.org/document/7568578/> Acesso em: 25 mar 2017.

TUKADIYA S. AERI A. **A comparative study of network based system log management tools**, Disponível em:<http://ieeexplore.ieee.org/document/7218075/> Acesso em: 12 mar 2017.

GHAFIR I. PRENOSIL V. **Blacklist-based malicious IP traffic detection,** Disponível em:<http://ieeexplore.ieee.org/document/7342657/> Acesso em: 1 mar 2017.

MOLLIN, R. A. **Public-key Cryptography: Theory and Practice**. 1ª. ed. [S.l.]: Chapman & Hall/CRC, v. I, 2002.

RAY Erik. **Learning XML,** 1.ed.,Ed.O'Reilly Media, Inc, 2001