UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

WILLIAM RENNAN DE CASTRO VIDAL

# Estudo de viabilidade para detecção de intrusão usando técnicas de *Big Data* para a análise de *logs*

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Alexandre da Silva Carissimi

Porto Alegre
2015

*“A sorte favorece a mente bem preparada.”*

— LOUIS PASTEUR

## AGRADECIMENTOS

Agradeço a Deus por me dar capacidade e força para concluir todas as dificuldades que colocou em minha vida, pois elas me ensinaram a vencer com qualidade e sabedoria.

À minha mãe, Jussara T. Castro Vidal, por todo o carinho e acreditar em mim desde o início. Sem você a conclusão desta etapa em minha vida não seria possível, muito obrigado.

À minha irmã, Claudia Vidal, e minha sobrinha, Roberta Vidal Zarpelon, por todo apoio e compreensão nos momentos de estresse. Vocês contribuíram muito para eu alcançar esse objetivo, muito obrigado.

Ao meu irmão, André Vidal, pelo suporte em momentos cruciais da minha vida, muito obrigado.

À minha namorada, Vanessa Mieres, pela compreensão e paciência em entender que nem sempre eu poderia estar ao seu lado, por me aguentar e me incentivar a finalizar mais esta etapa em minha vida, muito obrigado.

Ao meu orientador, Alexandre da Silva Carissimi, por todas revisões, sugestões e principalmente, pelo total apoio desde o início da ideia desta monografia. Sem os seus conselhos, a realização deste trabalho seria muito mais difícil, muito obrigado.

Agradeço aos meus amigos que de um jeito ou de outro foram essenciais por contribuir para a minha formação pessoal, profissional e acadêmica, em particular: Thiago Oliveira, William Gomes, Pablo Soares, Luiz Gustavo Gomes, Arthur Santos, Rafael Krause Cenci, Airto Pereira, Rodrigo Dobler e Lucas Neubert, obrigado.

Agradeço aos meus colegas do PoP-RS, pelo apoio e ajuda na construção de ideias para este trabalho. Em especial ao Lucas Arbiza pelas dicas e críticas construtivas, obrigado.

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| BASH | Bourne Again Shell |
| BSON | Binary Javascript Option Notation |
| CERT-BR | Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil |
| CPU | Central Processing Unit |
| DBMS | Data Base Management System |
| DMZ | Demilitarized Zone |
| DOS | Denial of Service |
| DP | Desvio Padrão |
| ELK | ElasticSearch, Logstash, Kibana |
| ES | ElasticSearch |
| GB | Gigabyte |
| Gbps | Gigabit per second |
| GHz | Gigahertz |
| GREP | Global Regular Expression Print |
| HDFS | Hadoop Distributed File System |
| HIDS | Host-based Intrusion Detection System |
| HTTP | Hypertext Tranfer Protocol |
| IDS | Intrusion Detection System |
| IP | Internet Protocol |
| IPS | Intrusion Prevention System |
| JSON | Javascript Option Notation |
| MB | Megabyte |
| NIDS | Network Intrusion Detection System |
| NOC | Network Operations Center |

| | |
|---|---|
| NoSQL | Not Only Structured Query Language |
| NSA | National Security Agency |
| PID | Process identifier |
| PoP-RS | Ponto de Presença da RNP no Rio Grande do Sul |
| RAID | Redundant Array of Independent Disks |
| RAM | Random Access Memory |
| RDBMS | Relational Data Base Management System |
| REST | Representational State Transfer |
| RNP | Rede Nacional de Ensino e Pesquisa |
| TB | Terabyte |
| TCP | Transmission Control Protocol |
| TI | Tecnologia da Informação |
| UDF | User-Defined Functions |
| UDP | User Datagram Protocol |
| XML | eXtensible Markup Language |

## LISTA DE FIGURAS

## LISTA DE TABELAS

**SUMÁRIO**

## RESUMO

Este trabalho apresenta um estudo de viabilidade para implementação de um IDS através do uso de tecnologias *Big Data* com o objetivo de analisar *logs*. Para isso é realizado um estudo das ferramentas de *Big Data*, bem como uma apresentação de seus principais conceitos, visando explorar um dos propósitos da tecnologia: *Analytics*. A ideia fundamental é efetuar pesquisas em *logs* de dispositivos de rede, como *switches*, roteadores e servidores, buscando por padrões específicos como forma básica de detectar intrusões. Para atingir esse objetivo, este trabalho propõe uma arquitetura de sistema (*software*) que emprega ferramentas em *software* livre para realizar a aquisição de dados, seu armazenamento, análise dos *logs* e visualização. Por fim, é feito uma avaliação da solução proposta usando como dados de entrada os *logs* coletados em uma infraestrutura real: o PoP-RS. Essa avaliação é realizada comparando-se o tempo de execução de busca por padrões em *logs* utilizando ferramentas tradicionais (*grep*) contra ferramentas *Big Data* (*Pig*).



**Palavras-chave:** IDS, Big Data, log, análise, software livre.

**Feasibility study for intrusion detection using Big Data techniques for analyzing logs**

## ABSTRACT

This work presents a feasibility study for implementation of an IDS through the use of Big Data technologies in order to analyze logs. For this, a study of Big Data tools it is conducted, as well as a presentation of its main concepts, aiming to explore one of the purposes of the technology: Analytics. The fundamental idea is to perform researches on logs of network devices, such as switches, routers and servers, searching for specific patterns as a basic way to detect intrusions. To achieve this goal, this work proposes a system architecture (software) using free software tools to perform data acquisition, its storage, log analysis and visualization. To conclude, an evaluation of the proposed solution is done using as the input data logs collected in a real infrastructure: PoP-RS. This evaluation is performed by comparing the search runtime for patterns in logs using traditional tools (grep) against Big Data (Pig) tools.

**Keywords:** IDS, Big Data, logs, analytics, free software.

# 1 INTRODUÇÃO

Com o crescente aumento e difusão do acesso a rede mundial de computadores, a Internet deixou de ser um lugar confiável. Usuários mal-intencionados corrompem os sistemas por meio de invasões, roubo informação (segredos de estado ou propriedades intelectuais), ataques de negação de serviço (*Denial of Service*, DOS) e invasão de privacidade, por exemplo, causando danos a outros indivíduos. Os motivos desses usuários mal-intencionados variam desde cobiçar ganhos financeiros até apenas pela satisfação própria das suas necessidades de autoafirmação e reconhecimento perante grupos de indivíduos com intenções semelhantes (SCHNEIER; VIEIRA, 2001).

Então, a segurança ao acesso a informação em um sistema de rede tornou-se muito importante, pois tenta proteger desde sistemas bancários contra prejuízos financeiros (ADACHI, 2004) a sistemas em setores manufatureiros contra espionagem industrial (LANGNER, 2011). Entretanto, o melhor e mais bem projetado sistema de segurança, para um ambiente de rede, inevitavelmente falhará (STALLINGS; VIEIRA, 2008). Assim, um Sistema de Detecção de Intrusão (*Intrusion Detection System*, IDS), que notifique quando aconteceu a invasão, é de suma importância. Um IDS auxilia na investigação pós-intrusão possibilitando traçar o caminho e os passos tomados pelo invasor, bem como identificar quais dados, ou recursos, foram comprometidos e as falhas que permitiram a ocorrência da invasão.

Um IDS pode ser projetado levando em consideração a sua localização na rede, forma de detecção, comportamento pós-detecção e sua frequência de uso (CAMPELLO; WEBER, 2001). Na concepção de Stallings e Vieira (2008), adotada neste trabalho, existem duas formas de detecção por comportamento: baseada por estatística de anomalia e baseada em regras. Na detecção realizada por estatística de anomalia é realizada a coleta de dados dos usuários legítimos a fim de traçar um perfil. Se no futuro algum comportamento for significativamente diferente do perfil, a intrusão é presumida. Já na detecção baseada em regras é configurado um conjunto de diretrizes para sentenciar se um comportamento pertence a um intruso.

Tipicamente, a detecção pode ser feita através da análise dos registros de atividades dos sistemas, popularmente chamado de *logs*. No *log* é encontrado o histórico de eventos realizados nos sistemas computacionais, e na infraestrutura de rede, que podem ser coletados diretamente em um hospedeiro e/ou na rede. Normalmente, a análise final é feita por um administrador de rede que verifica se aconteceu ou não uma intrusão. Existem softwares que auxiliam o trabalho de detectar intrusões, onde se destacam, dentre outros, o Snort (ROESCH et al., 1999) e OSSEC (BRAY; CID; HAY, 2008).

Contudo, em sistemas de rede de grande porte, há dois problemas associados: a não estruturação dos *logs* e sua quantidade. Primeiramente, os *logs* tendem a ser desestruturados, pois cada equipamento, ou aplicação, pode utilizar um formato de *log* próprio. Isso acaba dificultando uma análise por inspeção visual tornando impraticável uma análise humana (MARIANI; PASTORE, 2008). Além disso, os *logs* tendem a se tornarem grandes devido a quantidade de equipamentos e serviços na rede. Há soluções que gerenciam os *logs* ajudando a detecção, como o Splunk (SPLUNK, 2015) e o RSA-Pivotal (RSA-PIVOTAL, 2014), por exemplo, entretanto tratam-se de soluções comerciais e proprietárias, com um alto custo de investimento, não existindo muitas opções de soluções do mesmo nível em *software* livre para análise de *logs*. Levando em conta o crescente aumento no tamanho dos *logs*, soluções que utilizam técnicas de *Big Data* (LYNCH, 2008) para analisar a massiva quantidade de dados gerada pelos sistemas de rede são estudadas neste trabalho.

## 1.1 Objetivos do Trabalho

Tendo em vista a dificuldade de um Sistema de Detecção de Intrusão para analisar uma massiva quantidade de *logs*, o objetivo deste trabalho é propor a implementação de um IDS através de técnicas de *Big Data* utilizando ferramentas em *software* livre. Para isso será proposta e analisada a viabilidade em se desenvolver uma arquitetura para IDS baseada exclusivamente em ferramentas de *software* livre. Dessa forma, além do uso de *software* livre para concepção do sistema proposto, outro diferencial dessa solução será analisar um grande volume de dados (*logs*) de diferentes dispositivos de rede.

Além disso, o desempenho na análise de dados feita com a arquitetura proposta será avaliado a partir de *logs* de autenticação reais do Ponto de Presença da Rede Nacional de Ensino e Pesquisa no Rio Grande do Sul (PoP-RS) (POP-RS/RNP, 2015). Essa solução tem o propósito de auxiliar na visualização dos *logs* facilitando a análise por inspeção visual através da correlação de informação presente nos *logs*.

## 1.2 Organização do texto

Este trabalho é organizado em 6 capítulos, além desta introdução. O capítulo 2 apresenta os principais conceitos da área de segurança e de *Big Data*, fundamentais para o entendimento da proposta. No capítulo 3, é fornecida a especificação e a arquitetura proposta do IDS e quais os módulos necessários para sua construção. No capítulo 4 é discutido os detalhes de implementação da arquitetura proposta. O capítulo 5 apresenta uma avaliação do estudo de viabilidade do IDS utilizando técnicas de *Big Data*. Por fim, no capítulo 6 são ressaltados os principais desafios e contribuições deste trabalho, assim como trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

As intrusões já fazem parte do cotidiano, sejam elas em sistemas de rede, ou em espionagem mundial pela NSA (*National Security Agency*) (CLEMENT, 2014). Por esse motivo, são necessários formas para detectar a intrusão. Com verificações nos registros de auditoria (*logs*) é possível descobrir esses problemas, contudo, em uma grande escala, onde os dados não são estruturados e o volume gerado é grande, a detecção de intrusão por análise manual é impraticável (MARIANI; PASTORE, 2008). Assim, em virtude desses aspectos, características de *Big Data* podem ser exploradas para compor uma solução possível para a análise de *logs* (BEYER; LANEY, 2012).

Neste capítulo serão apresentados os princípios básicos de Segurança da Informação, Sistema de Detecção e Prevenção de Intrusão e *Big Data*. Esses fundamentos são importantes para o entendimento da proposta do trabalho.

## 2.1 Segurança da informação

Conforme Sêmola et al. (2003), Beal (2005) e Campos (2014), segurança da informação é o processo que visa proteger a informação dos perigos que podem violar a sua integridade, disponibilidade e confidencialidade. Esses três princípios formam a base essencial da segurança da informação.

Integridade refere-se ao quão íntegros os dados estão, ou seja, se a informação está em sua forma original. Visa proteger os dados contra alterações indevidas, intencionais e acidentais. Portanto, pretende deixar a informação confiável (CAMPOS, 2014). Já a disponibilidade requer que a informação esteja disponível a qualquer momento garantindo que os dados de um sistema são acessíveis, sob demanda, aos usuários ou processos autorizados (BEAL, 2005). Por fim, a confidencialidade diz respeito a proteção das informações de acordo com nível de sigilo de seu conteúdo. Objetiva limitar o acesso e uso apenas aos usuários para quem elas são destinadas (SÊMOLA et al., 2003).

As principais técnicas que tentam assegurar os princípios de integridade, disponibilidade e confidencialidade são implementadas através de *firewalls*, *softwares* IPS (*Intrusion Prevention System*) e IDS (*Intrusion Detection System*), *Honeypots* e zonas desmilitarizadas (*demilitarized zone*, DMZ). Um *firewall* consiste em uma barreira através da qual o tráfego, originado e destinado à infraestrutura de rede, precisa passar (STALLINGS; VIEIRA, 2008). É o *firewall* quem define o que pode ser permitido e o que deve ser proibido, ou seja, qual tráfego tem autorização

para passar em cada direção.

Um IDS, basicamente, representa um sistema usado para detectar sinais de atividade maliciosa em uma rede ou em um computador individual (GOODRICH; TAMASSIA, 2013). Já um IPS, tipicamente, é um sistema reativo que atua em conjunto com dispositivos de rede, como *firewalls*, para reduzir e prevenir as atividades maliciosas (GOODRICH; TAMASSIA, 2013). Uma outra ferramenta também usada para detecção de intrusão é um *honeypot*. Os *honeypots* são sistemas de armadilha, projetados para atrair um atacante em potencial para longe dos sistemas críticos. Além disso, são planejados para coletar informações sobre a atividade do atacante, encorajando-o a permanecer no sistema por tempo suficiente para deixar evidências que possam levar a sua identificação e/ou sua localização, visto que qualquer tipo de conexão ao *honeypot* pode ser seguramente identificada como intrusão (STALLINGS; VIEIRA, 2008). Além do mais, um IDS pode ser atualizado com as assinaturas recentes (padrões) de ataques com base na maneira com a qual foram iniciadas as conexões no *honeypot*.

Bishop (2005) define DMZ como uma parte da infraestrutura que separa a rede interna da rede externa. Essa técnica é utilizada para segregar um ambiente confiável (rede privada) de um ambiente não confiável, tipicamente, a Internet. Enfim, a intenção de todas essas técnicas é minimizar os prejuízos que os potenciais problemas possam causar quando um ou mais princípios da segurança da informação não são respeitados.

### 2.1.1 Ameaças e vulnerabilidades

Os problemas em uma infraestrutura de rede que tais técnicas visam combater são associados aos termos de vulnerabilidade e ameaça. A IETF (2000) descreve vulnerabilidade como uma falha, ou brecha, nos sistemas, ou no seu *design*/implementação, possibilitando que usuários mal-intencionados façam uso dessa para violar a política de segurança. Uma política de segurança estabelece os níveis de integridade, disponibilidade e confidencialidade, ou seja, define o que é permitido e o que é proibido em uma infraestrutura de rede.

Uma vulnerabilidade é um ponto de entrada no sistema computacional. Ela pode ser classificada de acordo com o seu tipo (RUSSELL; GANGEMI, 1991), sendo eles:

- Vulnerabilidade física: é relacionado à segurança física (prédio, edifício, sala, etc.) da instalação de TI. Nesse caso, fechaduras, guardas e até dispositivos biométricos são importantes para registrar e prevenir o acesso às instalações.
- Vulnerabilidade natural: relacionada a eventos naturais como terremoto, furacão, tsunami

e até mesmo indisponibilidade de energia elétrica que possam afetar os equipamentos e as informações. Em países sujeitos a ocorrências desses eventos, as instalações devem ser construídas para suportá-las, investindo em prevenção contra tais vulnerabilidades.

- Vulnerabilidade de *hardware* e *software*: originada no desenvolvimento do *software* ou incompatibilidade de *hardware* com *software*. Tipicamente, pode ser ocasionada quando o *software* foi instalado, mas não pré-configurado de maneira adequada.
- Vulnerabilidade de comunicação: permite que qualquer um possa entrar na rede. Ocorre, por exemplo, quando *switches* e pontos de rede cabeada, sem fio e portas TCP/UDP de máquinas estão abertas a qualquer indivíduo. Assim, a comunicação entre dispositivos deve ser segura de modo que as informações transmitidas alcancem o destino desejado sem sofrer nenhuma intervenção alheia. Uma forma de garantir a transmissão de informações se dá por meio de criptografia.
- Vulnerabilidade humana: referente a detectar, por exemplo, se o administrador da rede é suscetível a ataques de engenharia social. As brechas de segurança mais criticadas, e de maior incidência, vêm de falhas humanas (WINKLER; DEALY, 1995). Possuir um acesso restrito às informações, e qualificação adequada dos profissionais que atuam diretamente com ela, é de suma importância para uma organização manter seus dados em sigilo.

A exploração de uma vulnerabilidade, que potencialmente pode resultar em danos, é denominado ameaça. Os danos são causados por ataques. Um ataque é o acesso, não autorizado, ao sistema e é classificado em ativo ou passivo (IETF, 2000). Um ataque ativo é aquele que tenta alterar os recursos do sistema ou afetar o seu funcionamento padrão. Já um ataque passivo diz respeito a tentativa de "aprender", ou utilizar informações do sistema, sem afetar os seus recursos.

### 2.1.2 Detecção e Prevenção de intrusão

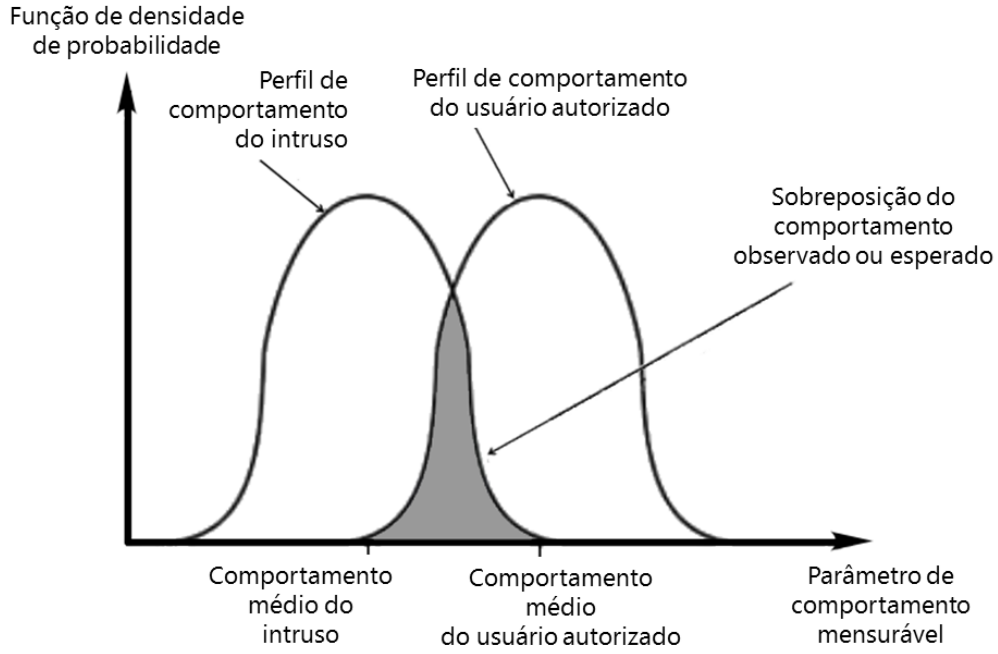
Qualquer equipamento pertencente a uma infraestrutura de rede é vulnerável a ataques (RUSSELL; GANGEMI, 1991). Como mencionado, as técnicas de *firewall* e DMZ formam a primeira frente de defesa em um sistema de rede. Porém, o melhor e mais bem projetado sistema de prevenção de intrusão inevitavelmente falhará (STALLINGS; VIEIRA, 2008). Dessa forma, sistemas de detecção de intrusão (IDS) e sistemas de prevenção de intrusão (IPS) tornam-se relevantes.

Segundo Stallings e Vieira (2008), a relevância da detecção e prevenção de detecção é motiva por três considerações: (i) em uma intrusão detectada com rapidez, o intruso poderá ser identificado e expulso do sistema antes de realizar qualquer dano; (ii) um sistema de detecção de intrusão eficaz serve como um elemento desencorajador, agindo para impedir intrusões; e, (iii) a detecção de intrusão permite a coleta de informações sobre as técnicas de intrusão, o que pode ser utilizado para fortalecer a estrutura de prevenção de intrusão.

A suposição de que o comportamento de um intruso é diferente de um usuário legítimo, de forma que possa ser quantificada, é a base da detecção de intrusão. Seguramente, não é fácil a distinção clara e exata entre um ataque de um intruso e o uso normal dos recursos por um usuário autorizado (STALLINGS; VIEIRA, 2008).

Aliado a isso, a sobreposição de comportamentos leva ao que se denomina falso negativo e falso positivo. Segundo CERT-BR (2015), um falso positivo é o termo que define uma situação na qual o IDS informa que aconteceu uma atividade maliciosa quando na verdade trata-se de uma atividade legítima, ou seja, usuários autorizados identificados como intrusos. Em contrapartida, um falso negativo é quando a atividade maliciosa passa imperceptível pelo IDS, ou seja, intrusos não são identificados como intrusos. A Figura 2.1 ilustra o problema que o projetista do IDS enfrenta na tentativa de eliminar os falsos positivos e falsos negativos da sua detecção.

Figura 2.1 – Perfis de comportamento de intrusos e usuários autorizados




(Fonte: Adaptado de Stallings e Vieira (2008))

Assim sendo, existe uma linha tênue do que é efetivamente uma intrusão. Na tentativa de enfrentar o problema de falsos positivos e falsos negativos foram propostas formas para detecção como a detecção estatística de anomalia e a detecção baseada em regras (PORRAS, 1993).

A técnica de detecção estatística de anomalia implica em coletar dados relacionados ao comportamento dos usuários verdadeiros por um período de tempo. Em seguida, são aplicados testes estatísticos para determinar, com um alto nível de confiabilidade, se esse comportamento do usuário é legítimo ou não. Há duas categorias de detecção estatística de anomalia: a detecção de limiar e a detecção baseada em perfil.

Com a técnica de detecção de limiar, a detecção é presumida quando a contagem de ocorrências de um tipo específico de evento ao longo de um intervalo de tempo ultrapassar um limiar preestabelecido. Essa análise de limiar, no entanto, é ineficaz já que é necessário definir tanto o limiar, quanto uma janela de tempo para contagem de ocorrência, e isso tende a gerar muitos falsos positivos e negativos, visto que existe uma variabilidade entre os usuários. Entretanto, detectores simples como esse, em conjunto de técnicas mais sofisticadas, podem ser úteis.

Já a detecção baseada em perfil foca na caracterização das atividades passadas dos usuários, ou grupos relacionados de usuários, detectando a intrusão através de desvios significativos no comportamento. O desvio em apenas um único parâmetro pode não ser suficiente para sinalizar uma intrusão, já que um perfil pode ser formado por um conjunto de parâmetros. O critério empregado nessa técnica é a análise dos registros de auditoria (*logs*) que originam a entrada para a função de detecção de intrusão de duas formas. Primeira, precisa-se decidir sobre as diversas métricas usadas para medir o comportamento do usuário. É possível determinar o perfil de atividade do usuário comum através da análise dos *logs* em uma janela de tempo. Logo, os *logs* servem para definir o comportamento típico. A segunda maneira utiliza os *logs* atuais como entrada para detectar a intrusão, ou seja, o modelo de detecção de intrusão analisa os *logs* que são criados para determinar o desvio de comportamento.

A técnica de detecção baseada em regras identifica uma intrusão observando os eventos no sistema e aplicando um conjunto de regras que levam a uma decisão com relação a se determinado padrão de atividade é, ou não, suspeito. Existem duas formas para implementação da técnica: a detecção de anomalia e a técnica de identificação de penetração.

A detecção de anomalia é baseada na definição de um conjunto de regras para identificar desvios de comportamentos em relação a padrões de uso anteriores. O histórico dos registros de auditoria é analisado para identificar padrões de uso e, automaticamente, gerar regras que

descrevem esses padrões, como regras que definem o comportamento de usuário, programas, privilégios e intervalos de tempo. O comportamento atual é então observado, e cada transação é comparada com o conjunto de regras para determinar se está em conformidade com qualquer modelo de comportamento observado anteriormente. Essa técnica, assim como a detecção estatística de anomalia, não exige conhecimento das vulnerabilidades do sistema. Pelo contrário, a ideia baseia-se na observação do comportamento passado supondo que no futuro será o mesmo. Para ser eficaz é necessário um banco de dados com um grande volume de regras.

Já a técnica de identificação de penetração é baseada em um sistema especialista. As regras são um recurso importante nesse sistema, pois possibilita identificar penetrações conhecidas ou que exploram pontos fracos conhecidos. As regras também podem ser definidas de forma que identifique um comportamento suspeito. Tipicamente, as regras utilizadas são específicas para a máquina e sistema operacional. Além disso, elas são geradas por especialistas em segurança e não por meio de uma análise automatizada dos *logs*. Normalmente, são realizadas entrevistas com administradores de rede e com analistas de segurança que listam um conjunto de cenários de penetração conhecidos e de eventos importantes que ameaçam a segurança do sistema. Assim, podemos dizer que o sucesso da técnica depende da habilidade dos envolvidos na criação das regras.

Até esse ponto, os sistemas de detecção de intrusão foram classificados de acordo com o método de detecção empregado. Contudo, há diversas formas de classificar um IDS (BACE; MELL, 2001; AXELSSON, 2000), e, uma maneira comum, é em função da localização de onde os eventos são coletados. Assim, Kruegel, Valeur e Vigna (2005) e Goodrich e Tamassia (2013) definem duas classes principais de IDS: baseado em hospedeiro e baseado em rede.

Um IDS baseado em hospedeiro (*Host-based Intrusion Detection System*, HIDS) reside em um único sistema e monitora a atividade nessa máquina, incluindo chamadas de sistema, comunicação entre processos e padrões de uso e recursos. Um HIDS opera monitorando registros diários e sistema de *logs* para detectar usuários que tentam ações não autorizadas. Geralmente, esses sistemas usam regras heurísticas, ou análise estatística, para detectar quando um usuário está se desviando de seu comportamento normal, o que poderia indicar que é um invasor.

Os Sistemas de Intrusão baseados em Rede (*Network Intrusion Detection System*, NIDS) detectam comportamento malicioso baseado em padrões de tráfego e conteúdo. Para isso, um NIDS usa sensores posicionados em locais estratégicos na infraestrutura da rede física. Geralmente, realizam inspeção de pacotes no tráfego de entrada e de saída aplicando um conjunto de assinaturas de ataques, ou heurísticas, para determinar se os padrões de tráfego indicam comportamento malicioso. Há pontos críticos relacionados ao ambiente e a infraestrutura de rede

que podem inviabilizar a utilização de um NIDS. Por exemplo, em redes de alta velocidade, o custo computacional exigido para realizar a captura e a análise do tráfego é muito alto, podendo trazer impactos sobre o tempo de resposta do sistema. Já em redes comutadas, o desafio está em escolher a localização e a técnica mais adequada para o NIDS. Algumas soluções para esses desafios são usar espelhamento de tráfego, ou fazer com que todos os dados passem por uma única porta de *up-link*. Entretanto, a adoção de criptografia para a comunicação reduz a ação do NIDS de forma significativa. Isso porque as partes do cabeçalho criptografadas não serão analisadas, impedindo o NIDS reconhecer as assinaturas, ou ataques.

Por fim, um IPS (*Intrusion Prevention System*) é um NIDS que, além de capturar, analisar e informar a ocorrência de um tráfego malicioso na rede, tenta bloquear ou mesmo minimizar os impactos causados pelo invasor através de solicitações ao *firewall* (MUKHOPADHYAY et al., 2011). Um IPS ideal é aquele que, mesmo na ocorrência de falhas em seu hardware, consegue detectar tráfego malicioso em toda a rede, bloqueando-o o mais próximo de sua origem, sem impactar no funcionamento da infraestrutura (SCARFONE; MELL, 2007).

Há uma variada gama de ferramentas em *software* livre que implementam IDS e IPS no mercado. O Snort (ROESCH et al., 1999) é usado para analisar o conteúdo dos pacotes que estão trafegando em uma rede na tentativa de prever ataques. Conta com bases de assinaturas construídas por especialistas, onde são cadastrados os comportamentos indevidos conhecidos de um sistema. Na medida que novos ataques são descobertos, cabe ao especialista adicionar uma regra de detecção a base de assinaturas. Já o OSSEC (BRAY; CID; HAY, 2008) monitora os *logs* de sistema em busca de atividades geralmente observadas em situações de ataque. Trata-se de um sistema que demanda do operador a definição de eventos irregulares e, assim, a aplicação observa o número de tentativas de *logins*, número de conexões de rede ou número de comandos executados, por exemplo. Posteriormente, em conjunto com ferramentas estatísticas, é definido quais valores observados são considerados anômalos.

As principais desvantagens dessas ferramentas diz em respeito a frequência que seu banco de assinaturas é atualizado. Outro problema é o fato de gerar um número significativo de falsos positivos. Isso contribui para uma "poluição" na visualização das verdadeiras ameaças. Visando resolver esses problemas, há ferramentas comerciais que fazem uso de gráficos e também atualizam o seus bancos de assinaturas de ataque automaticamente, como a USM da AlienVault (USM, 2015).

### 2.1.3 Registros de auditoria (*logs*)

Para a detecção de intrusão, o registro de auditoria, popularmente chamado de *log*, é uma ferramenta fundamental. Um registro da atividade contínua feita por usuários, serviços e equipamentos do sistema de rede precisa ser mantido como entrada de um IDS. Tipicamente, isso pode ser feito de duas maneiras distintas: por registros de auditoria nativos ou por registros de auditoria específicos (STALLINGS; VIEIRA, 2008).

Os registros de auditoria nativos são realizados por *softwares* de contabilidade que coletam informações sobre as atividades dos usuários, estando presente em praticamente todos os sistemas operacionais multiusuário. A principal vantagem de utilizar tais dados é que nenhum *software* de coleta adicional é necessário. A desvantagem é que os registros podem não conter informações necessárias, ou podem não estar em um formato conveniente.

Por outro lado, um recurso de coleta pode ser desenvolvido para gerar somente informações pertinentes ao sistema de detecção de intrusão, esses são os registros de auditoria específicos. Uma vantagem é que esses sistemas podem ser implementados independente do fornecedor e utilizada em vários sistemas. Contudo, há um *overhead* extra envolvido pelo fato de realizar duas contabilizações (a nativa, que não pode ser suprimida, e a específica).

Em geral, os registros de auditoria nativos e específicos são utilizados pelo IDS. A confiabilidade do dado nativo muitas vezes não compensa o fato de não serem completos ou convenientes para análise. Consequentemente, os dados fornecidos por registros de auditoria específicos são necessários. Denning (1987) desenvolveu um registro de auditoria específico para a detecção, onde cada entrada de registro contém os campos de Sujeito, Ação, Objeto, Condição de exceção, Uso de recurso e Carimbo de tempo.

O Sujeito é o iniciador de uma ação, normalmente, um usuário, ou um processo, atuando em nome de usuários, ou grupos de usuários, que executam uma ação. Toda atividade começa por meio de comandos emitidos pelo Sujeito. A Ação é uma operação realizada pelo sujeito, como um *login* ou uma leitura de arquivo. Os receptores da ação são os Objetos. Exemplos de objetos incluem arquivos, programas, mensagens e registros. O campo de Condição de exceção indica o retorno da exceção (se houver).

No campo Uso de recurso expressa uma lista de elementos quantitativos, em que cada elemento indica a quantidade usada de algum recurso, como número de registros lidos e tempo decorrido na sessão. Por fim, o Carimbo de tempo identifica quando a ação ocorreu, popularmente chamado de *timestamp* (marca temporal).

Soluções de tratamento de *log* específico tentam seguir essa organização de campos visando padronizar os registros de auditoria. Um padrão fornecido por muitos sistemas baseados em *Unix* é o *Syslog* (LONVICK, 2001), que permite que o programador especifique o texto da mensagem descrevendo um evento para ser salvo no arquivo de *log*. Informações adicionais são automaticamente colocadas, como o Carimbo de tempo, que identifica quando o evento ocorreu, o Sujeito, e o Objeto em que o programa está executando. Contudo, uma dificuldade em analisar os *logs* está na maneira que algumas aplicações o utilizam. Certas aplicações utilizam o sistema de *logs* para escrever mensagens de *debugging* que não são necessariamente próprias para detecção de intrusão. Isso causa uma poluição nos *logs*. Além disso, programas específicos não seguem o padrão *Syslog*. Isso dificulta a extração de informações relevantes para análise. Por fim, as informações salvas no sistema de *logs* podem ser facilmente poluídas pelo atacante na tentativa de encobrir os seus rastros, tornando difícil a detecção.

Em uma infraestrutura de rede substancialmente grande, a quantidade de ações feita por sujeitos gerando entrada de registros podem se tornar volumosa. Portanto, o trabalho de analisar os *logs* torna-se complexo. Assim, soluções que tratem de um grande volume de informações devem ser investigadas (THERDPHAPIYANAK; PIROMSOPA, 2013).

## 2.2 Grande volume de dados - *Big Data*

No atual cenário da tecnologia da informação, o crescente volume de dados gerado pelas redes sociais, e-mails, sensores, arquivos em servidores, e também de registros de auditoria implicam no uso de uma nova definição: *Big Data* (LOHR, 2012). O termo tem sido divulgado no esforço de definir grandes quantidades de dados que aumentam com o decorrer do tempo. O objetivo de *Big Data* refere-se à prática de extrair, de quantidades massivas de informações, conhecimentos mais relevantes sobre os dados existentes.

Assim, dados menores, porém significativos, são criados, simplificando a busca e análise de novas informações. Muitas definições foram propostas com o intuito de definir o conceito de *Big Data* de tal modo que há inúmeros trabalhos que tentam organizá-las (WARD; BARKER, 2013; MAURO; GRECO; GRIMALDI, 2015).

Laney (2001), analista do META (atual grupo Gartner), propôs a noção inicial de *Big Data* com o modelo 3V (Volume, Velocidade e Variedade), estabelecendo os desafios e oportunidades trazidos pelo constante crescimento dos dados. No ano seguinte, a Gartner fez uma nova definição: "*Big Data* são grandes volumes de dados de alta velocidade, e/ou alta variedade de informações que requerem novas formas de processamento para possibilitar uma eficiente

tomada de decisão" (BEYER; LANEY, 2012).

O modelo 3V é referenciado e usado por diversos autores (EATON et al., 2012; ZASLAVSKY; PERERA; GEORGAKOPOULOS, 2013) e também pela indústria, incluindo a IBM (ZIKOPOULOS; EATON et al., 2011) e a Microsoft (MEIJER, 2011). O volume remete à massiva quantidade de dados gerados a cada instante de tempo; velocidade diz respeito a coleta e análise dos dados que devem ser rapidamente conduzida de forma que se utilize todo o potencial da técnica; e variedade indica a grande diversidade de formatos de dados, ou seja, são informações que podem ser não estruturadas, estruturadas e semiestruturadas (seçao 2.2.1).

Entretanto, outros autores estenderam o modelo 3V para 4V adicionando a característica de Veracidade (SCHROECK et al., 2012) ou Valor (DIJCKS, 2012). A veracidade remete ao conceito de confiabilidade, ou seja, quão confiável os dados são. A característica de valor diz respeito ao problema mais crítico em *Big Data* que é em como descobrir informações importantes que tenha um alto grau de relevância (valores significativos) em uma escala enorme de dados, de vários tipos e que ainda são gerados rapidamente. Essa noção de valor é discutida pelo grupo IDC, um dos mais influentes líderes no campo de pesquisa em *Big Data*. Em 2011, um relatório da IDC define *Big Data* como "uma nova geração de tecnologias e arquiteturas, implementadas de forma econômica para extrair valores de grandes quantidades de dados de variados tipos, habilitando alta velocidade de captura, descoberta e análise desses dados" (GANTZ; REINSEL, 2011).

Com o passar do tempo, novas características foram adicionadas ao modelo. A indústria, e alguns autores mais modernos, já citam um modelo de 7V que adiciona volatilidade e validade, as características de velocidade, volume, variedade, veracidade e valor (KHAN; UDDIN; GUPTA, 2014). Volatilidade está embasado naquilo que está sujeito a mudanças frequentes. Dessa forma, o momento que define quando os dados podem ser destruídos se faz por meio de uma política de retenção de dados. Validade refere-se à exatidão e precisão dos dados no que diz respeito ao uso pretendido, ou seja, os dados podem não ter problemas de veracidade, mas podem não ser válidos se não forem devidamente compreendidos. Por exemplo, o mesmo conjunto de dados pode ser válido para uma aplicação e inválido para outra.

Adicionalmente, para o grupo de trabalho de *Big Data* do NIST (NBD-WG, 2011), *Big Data* consiste em: "Dados com grande volume, velocidade de aquisição e representação de dados, que é limitada pelos tradicionais métodos, para realizar uma análise eficaz para que possam ser processados eficientemente com importantes tecnologias de *zoom*" (referindo-se a uma melhor visualização dos valores obtidos). Refletindo sobre o proposto pelo NIST, uma nova característica é adicionada à definição: a visualização, ou seja, como visualizar de forma

efetiva os resultados obtidos. Levando em conta uma definição mais usual, neste trabalho é adotado a proposta do modelo 4V. Assim, nas próximas seções são utilizados os termos volume, velocidade, variedade e valor (DIJCKS, 2012).

Tipicamente, o objetivo de *Big Data* está em resolver o problema de uma massiva quantidade de dados que precisa ser processados para gerar novas informações. Dessa forma, as fases envolvidas para concluir essa meta são: geração, aquisição, armazenamento e análise (*Analytics*) dos dados (HU et al., 2014).

### 2.2.1 Dados estruturados e não estruturados

*Big Data* lida com um grande volume de informações armazenadas em uma base de dados. Características intrínsecas, como alto volume e grande variedade de informação, tornam muito complexo, e difícil, operações simples como remoção, ordenação e sumarização de forma eficiente utilizando os tradicionais Sistemas Gerenciadores de Bases de Dados (*Data Base Management System*, DBMS), ou seja, RDBMS (*Relational Data Base Management System*) (VIEIRA et al., 2012).

O principal problema nos RDBMS está nas técnicas de gerenciamento de dados usadas em dados semiestruturados e não estruturados, pois elas perdem desempenho na medida em que o banco de dados aumenta. Dados não estruturados são aqueles que não possuem uma estrutura definida, como documentos em texto. Dados semiestruturados possuem organização bastante heterogênea (ABITEBOUL, 1997). A diferença para os dados estruturados é que estes são organizados em blocos semânticos (relações).

Nesse contexto, o gerenciamento de dados requer técnicas para manter a disponibilidade e um alto desempenho em um ambiente de dados variados e distribuídos. Assim, muitas dessas técnicas são agrupadas sob o termo “banco de dados NoSQL” (*Not Only SQL*) (LOSHIN, 2013). O principal objetivo das soluções NoSQL não é substituir os modelos de banco de dados relacionais, mas sim ser utilizado em situações onde uma maior flexibilidade na estruturação da base de dados seja necessária (CHANG et al., 2008).

Estão presentes em sistemas NoSQL características como grande flexibilidade no gerenciamento do banco de dados, escalabilidade horizontal, ausência de esquema, que permite de forma fácil a alta escalabilidade, e aumento da disponibilidade. Por outro lado, não há garantias quanto a integridade dos dados, diferente dos modelos relacionais que mantém uma estrutura rígida (LÓSCIO; OLIVEIRA; PONTES, 2011).

Há diversas formas de classificar os sistemas NoSQL (NOSQL, 2015). Os bancos de dados NoSQL podem ser categorizados quanto ao seu modelo de armazenamento (BAJPAYEE; SINHA; KUMAR, 2015; HAN et al., 2011; HE, 2015; MONIRUZZAMAN; HOSSAIN, 2013), onde os mais difundidos são:

- Chave-valor (*Key-Value*): Os dados são representados como uma coleção de pares de chaves e valores, onde cada chave aparece no máximo uma única vez na coleção. As chaves são identificadores alfanuméricos. Os valores podem ser sequências de texto simples ou listas mais complexas. Pesquisas de dados são limitadas a correspondências exatas, e normalmente só pode ser realizadas com o conteúdo das chaves, não dos valores. DynamoDB (DYNAMODB, 2015) e Azure Table Storage (AZURE, 2015) são exemplos típicos desse modelo.
- Orientado a colunas (*Columm Store*): Os dados são organizados em tabelas sendo similar a um RDBMS, porém o conteúdo é armazenado por colunas ao invés de linhas. Em cenários com poucos registros salvos, ou de recuperação de um registro específico, a abordagem tradicional de um RDBMS, ou seja, orientada a linhas, é mais vantajosa visto que cada coluna tem que ser acessada separadamente. Contudo, usando o modelo orientado a colunas, é possível empacotar os dados de maneira eficiente, já que dados semelhantes (de mesmo formato) estão próximos. Logo, é possível recuperar e armazenar mais informações em menos tempo. Assim, é muito adequado para processamento de massivos volumes de dados. Os exemplos de banco de dados orientados a coluna são HBase (HBASE, 2015) e Cassandra (CASSANDRA, 2015).
- Orientado a documentos (*Document Store*): Tem como objetivo armazenar e gerenciar documentos. O documento é codificado em um formato padrão como XML (*eXtensible Markup Language*), JSON (*Javascript Option Notation*) ou BSON (*Binary JSON*). Em contraste com o modelo chave-valor, o valor da coluna nesse modelo contém os dados semiestruturados (especificamente pares nome-valor). Uma única coluna pode conter centenas desses atributos, assim como o número e tipo de atributos gravados pode variar de linha a linha. Além disso, sua principal vantagem com relação ao modelo chave-valor, onde a procura é feita apenas no campo chave, é que no modelo orientado a documentos a procura é feita em ambos os campos (chave e valor), ou seja, é possível realizar consultas pelo conteúdo do documento. Exemplos: ElasticSearch (ELASTICSEARCH, 2015) e MongoDB (MONGODB, 2015).
- Orientado a grafos (*Graph-Based Store*): Armazena e recupera os dados aplicando a teoria dos grafos. As unidades de dados são visualizadas como nós, enquanto que as

arestas são as relações entre os dados, objetivando a interligação dos diferentes pares de dados (relação entre os dados). Essa característica é essencial para o armazenamento de informações em redes sociais, por exemplo. Além disso, seu foco na representação visual de informações torna-o mais amigável do que outros modelos. Neo4j (NEO4J, 2015) e InfoGrid (INFOGRID, 2015) são exemplos de banco de dados representados por esse modelo.

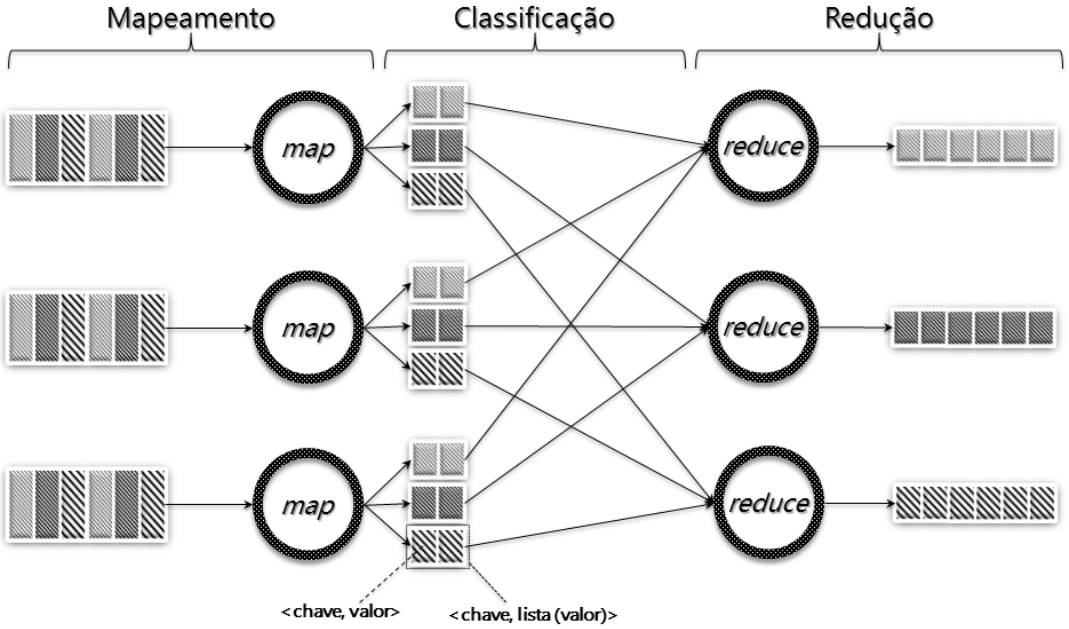
O modelo chave-valor não é o mais adequado para a busca por padrões em *logs*, visto que é necessário busca pela ocorrência exata. Além disso, essa busca deve ser feita somente no campo da chave. Já no modelo orientado a colunas pode ser explorado, pois é adequado para o agrupamento de dados semelhantes, facilitando a busca, já que ficam próximos. O problema é que cada *log* pode ter um formato diferente no campo onde a mensagem do evento é armazenada. Com modelo orientado a grafos existe a possibilidade de realizar algum tipo de correlação de dados através de como os dados são armazenados. Contudo, seu foco é na visualização de informações e, talvez, não seja preparado para a busca em arquivos texto. Por fim, o modelo orientado a documentos visa a busca de informações em conteúdos do documento sendo ideal para buscas em *logs*, já que são arquivos texto.

### 2.2.2 MapReduce

O *MapReduce* faz parte da classificação *Dwarf Mine* (COLELLA, 2004) que organiza métodos algorítmicos de acordo com seu comportamento de computação e comunicação. Essa classificação foca em padrões básicos que persistem nas aplicações, sendo independentes de implementação. A categoria *MapReduce* corresponde a aplicações com paralelismo massivo, e poucos, ou ausentes, momentos de comunicação. O paradigma *MapReduce* divide-se em 3 fases principais: o mapeamento de tarefas independentes para múltiplas unidades de processamento; computações paralelas, responsáveis por agrupar determinados valores; e, a redução dos resultados independentes agrupados na fase anterior.

O funcionamento básico do *MapReduce* é ilustrado na Figura 2.2. A fase de Mapeamento é responsável por mapear fragmentos de dados de entrada à uma chave. Na fase intermediária de classificação, os dados são agrupados pelas chaves. Por fim, a fase redução tem o objetivo de realizar uma computação sobre os dados mapeados por uma mesma chave.

Figura 2.2 – Arquitetura de funcionamento do *MapReduce*




(Fonte: Adaptado de : Hadoop (2015a))

Na fase de mapeamento, a operação *map* é responsável por associar os tipos de valores da entrada de dados a uma chave, formando tuplas <*chave, valor*>, ou seja, cada tipo de dado é assinalado com uma chave. Na fase de classificação são usadas as tuplas de chave e valor criadas na fase anterior para realizar algum algoritmo, normalmente de classificação, propiciando uma lista de valores, ou seja, os tipos de valores são agrupados pelas chaves assinaladas na fase anterior produzindo novas tuplas <*chave, lista (valor)*>. Após isso, é encaminhado para a fase de redução onde é realizada a computação em cima da fase anterior, ou seja, a operação *reduce* consome o conjunto de tuplas <*chave, lista (valor)*> gerando a saída. Nessa fase é feita união dos "pedaços" de mesma chave.

Atualmente, existem várias implementações *open source* do *MapReduce*, dentre elas o *Apache Storm* (STORM, 2015) e *Apache Spark* (SPARK, 2015) orientadas a análise de dados em tempo real, e o *Apache Hadoop* (HADOOP, 2015b) orientado a análise em *batch*.

### 2.2.3 Análise de dados

Um dos principais aspectos de *Big Data* que interessa neste trabalho é o poder computacional envolvido na análise dos dados, ou seja, *Analytics*. Para compreensão do termo,

é necessário investigar duas áreas relacionadas: correlação de eventos e mineração de dados (*Data Mining*).

Técnicas de correlação de eventos são discutidas em trabalhos como (REGUIEG et al., 2012; BARROS et al., 2007). Basicamente, no caso de correlação de eventos em *logs*, o objetivo é verificar se existe algo em comum nos registros de auditoria armazenados nos equipamentos da infraestrutura (MOTAHARI-NEZHAD et al., 2011). A mineração de dados permite a extração, não trivial, de conhecimento previamente desconhecido, e potencialmente útil, de um banco de dados (ADRIAANS; ZANTINGE, 1996).

Assim, o termo *Analytics* é frequentemente usado para cobrir qualquer tomada de decisão orientada a dados, sendo comumente utilizado por dois grandes grupos: equipes corporativas, para análise analítica, e cientistas de pesquisas acadêmicas (FISHER et al., 2012). No mundo corporativo, as equipes usam o seu conhecimento em estatística, mineração de dados, aprendizado de máquina, e visualização, para concluir informações que visam responder perguntas de valor econômico relevante para seus negócios. Por exemplo, combinando dados de fontes públicas como *Twitter*, com os dados dos seus próprios consumidores, gerando escolhas significativas para o negócio. No mundo acadêmico, cientistas analisam dados para testar hipóteses e formar teorias. Tipicamente, os cientistas escolhem suas próprias perguntas para investigação, tendo um maior controle sobre a fonte de dados. Embora existam diferenças entre as duas metodologias, em ambos os casos, *Analytics* faz uso de técnicas de mineração e correlação de dados.

## 2.3 Ferramentas de *software* livre para *Big Data*

No *Apache Hadoop* a idealização de *Analytics* corresponde ao paradigma *MapReduce* (DEAN; GHEMAWAT, 2008), onde o mapeamento é nomeado *map*, e a redução é nomeada *reduce*. O principal interesse no *Hadoop* se deve ao seu numeroso e crescente ecossistema (MONTEITH; MCGREGOR; INGRAM, 2013). Abaixo é listado os principais subprojetos desse ecossistema por trás da *Apache Software Foundation* (APACHE, 2015).

- *Hadoop Common*: contém um conjunto de utilitários e a estrutura base que dá suporte aos demais subprojetos do *Hadoop*.
- *Hadoop MapReduce*: um modelo de programação especializado no processamento de conjuntos de dados distribuídos em um aglomerado computacional. Abstrai toda a computação paralela em apenas duas funções: *Map* e *Reduce*.

- *Hadoop YARN*: é um *framework* para programação de trabalho e gestão de recursos de ambiente distribuído que amplia as funcionalidades de gerenciamento do *MapReduce*.
- *Hadoop Distributed File System (HDFS)*: é o sistema de arquivos distribuído nativo do *Hadoop*. Possui mecanismos de tolerância a falhas.
- *Avro*: sistema de serialização de dados (AVRO, 2015).
- *Flume*: permite a agregação e movimentação eficiente de grandes quantidades de dados de *log* de diferentes origens para *Hadoop* (FLUME, 2015).
- *Hbase*: é um banco de dados distribuído e escalável que dá suporte ao armazenamento estruturado e otimizado para grandes tabelas (HBASE, 2015).
- *Hive*: fornece uma infraestrutura que permite utilizar Hive QL, uma linguagem de consulta similar a SQL, bem como demais conceitos de dados relacionais, tais como tabelas, colunas e linhas, facilitando as análises complexas feitas nos dados não relacionais de uma aplicação *Hadoop* (HIVE, 2015).
- *Pig*: fornece uma linguagem de alto nível orientada a fluxo de dados chamada *Pig Latin* e um compilador capaz de transformar os programas do tipo *Pig* em sequências do modelo de programação *MapReduce* (PIG, 2015).
- *ZooKeeper*: serviço centralizado para facilitar as tarefas de configuração de nós, sincronização de processos distribuídos e grupos de serviço (ZOOKEEPER, 2015).

Contudo, há outros ecossistemas, como o *Elastic* (ELASTIC, 2015), com ferramentas que podem substituir as já citadas (BAGNASCO et al., 2015). Por exemplo, o *ElasticSearch* (GORMLEY; TONG, 2015) para banco de dados e o *Logstash* para movimentação e tratamento de *logs* (LOGSTASH, 2015). Também existem as que complementam, ou que se conectam ao ecossistema, como o *Kibana* (KIBANA, 2015) para visualização de dados e o *ES-Hadoop* (ES-HADOOP, 2015a) para interligar o *Hadoop* com o *ElasticSearch*.

## 2.4 Estado da arte: IDS baseados em técnicas de *Big Data*

Há muitas soluções que fazem uso da integração de *Big Data* com IDS. Tratando-se de *software* livre, existem soluções, como o PacketPig (PACKETPIG, 2015) que faz uso do tratamento nos dados proporcionado por *Big Data* para realizar análise nos tráfegos de dados em uma infraestrutura de rede. Um inconveniente é o fato de ser voltado para o mercado de "farejadores de tráfego" (*sniffers*) não sendo utilizado para análise de *logs*.

Por outro lado, o Splunk (SPLUNK, 2015) não tem essa limitação. Ele foi desenvolvido para facilitar o trabalho dos administradores de rede gerenciando os *logs* para uma análise rápida e eficiente, podendo observar o comportamento da rede em tempo real. Outra vantagem é o uso de gatilhos de segurança. Se um certo comportamento pré-configurado foi observado, ele dispara avisos aos administradores de segurança (CARASSO, 2012). Logo, seu uso não se limita a gerenciamento de *log*, sendo muito usado como uma solução de detecção e prevenção de intrusão. Contudo, o principal problema, por ser uma solução proprietária, é a existência de um alto nível de dependência com o fabricante, além de ser gratuito apenas para bases de dados que gerem menos de 500 MB por dia.

Outra solução proprietária, porém que foca exclusivamente no ramo de segurança, é o RSA-Pivotal (RSA-PIVOTAL, 2014). É possível correlacionar os dados de tráfego com as informações presentes nos *logs* gerando uma análise mais consistente. Contudo, tem um alto custo financeiro por ser voltado para grandes redes corporativas e, por também ser uma solução proprietária, há uma dependência com o fabricante.

Por fim, também há soluções que se especializam em examinar um tipo específico de informações. O Ortoza (ORTOZA, 2014) faz uso do *Apache Hadoop* para analisar *logs* de aplicações *web*. Apesar de fazer uso de soluções *open source*, cobra pelo uso do serviço de *Big Data* (*Big Data as a Service*). Assim, sofre das mesmas desvantagens de uma solução proprietária.

## 2.5 Considerações finais

A detecção de intrusão é importante para um sistema de rede, pois torna possível a investigação rápida dos problemas intrínsecos de vulnerabilidades nas redes, identificando e expulsando intrusos. Além disso, a detecção de intrusos possibilita a coleta de informações referentes a técnicas de intrusão podendo fortalecer os sistemas de prevenção. Tipicamente, a detecção de intrusão é realizada via análise de *logs* e esses podem se tornar grandes, inviabilizando análises manuais e tradicionais. Assim, há um interesse comum em realizar análises de *logs* para detectar intrusões utilizando técnicas de *Big Data*. Dessa forma, o capítulo 3 apresenta uma proposta para a definição e desenvolvimento de um IDS utilizando técnicas de *Big Data* e ferramentas em *software* livre.

# 3 PROPOSTA

Neste capítulo é apresentado a proposta de implementação do IDS através de técnicas de *Big Data* que tem o propósito de viabilizar a sua construção. São detalhados os requisitos do sistema, classificando-os em funcionais e não funcionais. Também são descritas as funcionalidades da interface e, por fim, é discutida a arquitetura proposta do sistema com base em uma arquitetura tradicional de um IDS.

## 3.1 Requisitos do sistema

As definições de requisitos do sistema especificam o que o sistema deve fazer (suas funções) e suas propriedades essenciais e desejáveis (SOMMERVILLE, 2007). Segundo Pfleeger (2004), um requisito de um sistema é uma característica do sistema, ou a descrição de algo que o sistema é capaz de realizar, para atingir seus objetivos. De acordo com Filho (2003), os requisitos do sistema são as características que definem os critérios de aceitação de um produto.

Tanto para Filho (2003), quanto para (SOMMERVILLE, 2007), as características dos requisitos podem ser dividias em duas: (i) funcionais, que representam os comportamentos que um sistema deve apresentar diante de certas ações; e, (ii) não funcionais, que quantificam determinados aspectos do comportamento. Nas próximas subseções os conceitos de requisitos funcionais e não funcionais serão aprofundados e exemplificados no contexto deste trabalho.

### 3.1.1 Requisitos funcionais

Os requisitos funcionais são declarações de serviços que o sistema deve prover, descrevendo o que o sistema deve fazer, como o sistema deve reagir a entradas específicas, como deve se comportar em determinadas situações e o que o sistema não deve fazer (SOMMERVILLE, 2007). Um requisito funcional expõe uma interação entre o sistema e o seu ambiente (PFLEEGER, 2004). No contexto deste trabalho, são listados os seguintes requisitos funcionais:

- Abordagem : É previsto que a análise seja *postmortem* (dados estáticos), isto é, não será em tempo real (dados dinâmicos).
- Comportamento suspeito: O IDS deve ser capaz de detectar eventos gerados por ataque de força bruta, desfiguração de página (*Defacement*) e qualquer acesso aos equipamentos da infraestrutura originadas de *IPs* externos.

- Alertas: Ao verificar um comportamento suspeito, deve ser enviado um alarme ao Centro de Operações de Rede (*Network Operations Center*, NOC) do PoP-RS. Assim, os administradores deverão analisar esse evento através de uma interface gráfica do sistema.

A especificação de um requisito funcional deve designar o que se espera que o *software* faça, sem a preocupação de como o *software* faz. É importante diferenciar a atividade de determinar requisitos da atividade de especificação que ocorre durante o *design* do *software*. No design do *software* deve-se tomar a decisão de quais a funções o sistema efetivamente terá para satisfazer as demandas dos usuários (SOMMERVILLE; SAWYER, 1997).

### 3.1.2 Requisitos não funcionais

Os requisitos não funcionais expressam qualidade e restrições sobre os serviços, ou as funções, oferecidos pelo sistema (SOMMERVILLE, 2007). Essas restrições, muitas vezes, limitam as opções para criar uma solução para o problema (PFLEEGER, 2004).

Os requisitos não funcionais são originados nas necessidades dos usuários, em restrições orçamentarias, em políticas organizacionais, em necessidades de interoperabilidade com os demais sistemas ou em fatores externos como regulamentos e legislações (SOMMERVILLE, 2007). Dessa forma, os requisitos não funcionais podem ser categorizados quanto a sua origem. Há diversas classificações de requisits não funcionais e Sommerville (2007) classifica-os em: requisitos de produto; requisitos organizacionais; e, requisitos externos.

- Requisitos de produto: especificam o comportamento do produto (Sistema de Detecção de Intrusão, IDS). Esses requisitos referem-se a atributos de qualidade que o sistema deve apresentar, tais como confiabilidade, usabilidade, eficiência, portabilidade, manutenibilidade e segurança. Neste trabalho, como medida de segurança, é esperado que, depois de finalizado, o sistema não se comunique com o ambiente externo (Internet). As futuras manutenções que o sistema possa necessitar também é uma preocupação, assim é desejado uma documentação dos *softwares* e lista de pacotes instalados (manutenibilidade). Além disso, é desejável uma interface amigável, customizável e via *browser*, para visualização dos *logs* facilitando a análise (usabilidade). A base de dados, que permite o uso do sistema, deve ser protegida para acesso apenas por usuários autorizados (segurança visando confiabilidade).
- Requisitos organizacionais: são derivados de metas, políticas e procedimentos das organizações do cliente e do desenvolvedor. Incluem requisitos de processo (padrões de pro-

cesso e modelos de documentos que devem ser usados), requisitos de implementação (tal como a linguagem de programação a ser adotada), restrições de entrega (tempo para chegar ao mercado, restrições de cronograma, por exemplo), restrições orçamentárias como custo, ou custo-benefício, por exemplo. No contexto deste trabalho, é esperado que se use os modelos de documentação do PoP-RS. Além disso, restrições de cronograma imposta pelo calendário da UFRGS para a finalização deste trabalho é levada em consideração. Com isso, o tempo de desenvolvimento não deve ultrapassar o cronograma presente no Trabalho de Graduação I (Anexo A). Restrições financeiras como custo também devem ser observadas, assim foi decidido realizar a implementação utilizando tecnologia desenvolvidas em *software* livre. Como requisito de implementação, é esperado que a análise dos *logs* deva ser feita por meio da utilização de ferramentas nomeadas como *Big Data*, sem interferência no ambiente de produção do PoP-RS.

- Requisitos externos: referem-se a todos os requisitos derivados de fatores externos ao sistema e seu processo de desenvolvimento. Podem incluir requisitos de interoperabilidade com sistemas de outras organizações, requisitos legais (tais como requisitos de privacidade) e requisitos éticos. Na composição deste trabalho, por questões éticas, é esperado não utilizar referencia que remetam a qualquer tipo de informação prejudicial ao ambiente de produção do PoP-RS, tais como endereços *IP*, portas TCP/UDP, usuários e qualquer outra informação relevante presente nos *logs*. Assim, é mantido em sigilo informações referentes a qualquer tipo de informação que possa ser utilizadas por usuários mal-intencionados. Por uma questão de interoperabilidade, é preferível o uso de *software* livre, isto é, a solução deverá se comunicar com outros *softwares* presentes no PoP-RS, sendo independente de fabricante.

Enfim, alguns desses requisitos são, provavelmente, traduzidos/implementados em funções (operacionalizados), ao longo do processo de desenvolvimento de *software* (CHUNG et al., 2012). Para o desenvolvimento do projeto, foi utilizada a metodologia estruturada, na qual adota o modelo em cascata, onde cada atividade tem que ser completada e finalizada antes que a atividade seguinte possa ser iniciada (SOMMERVILLE, 2007).

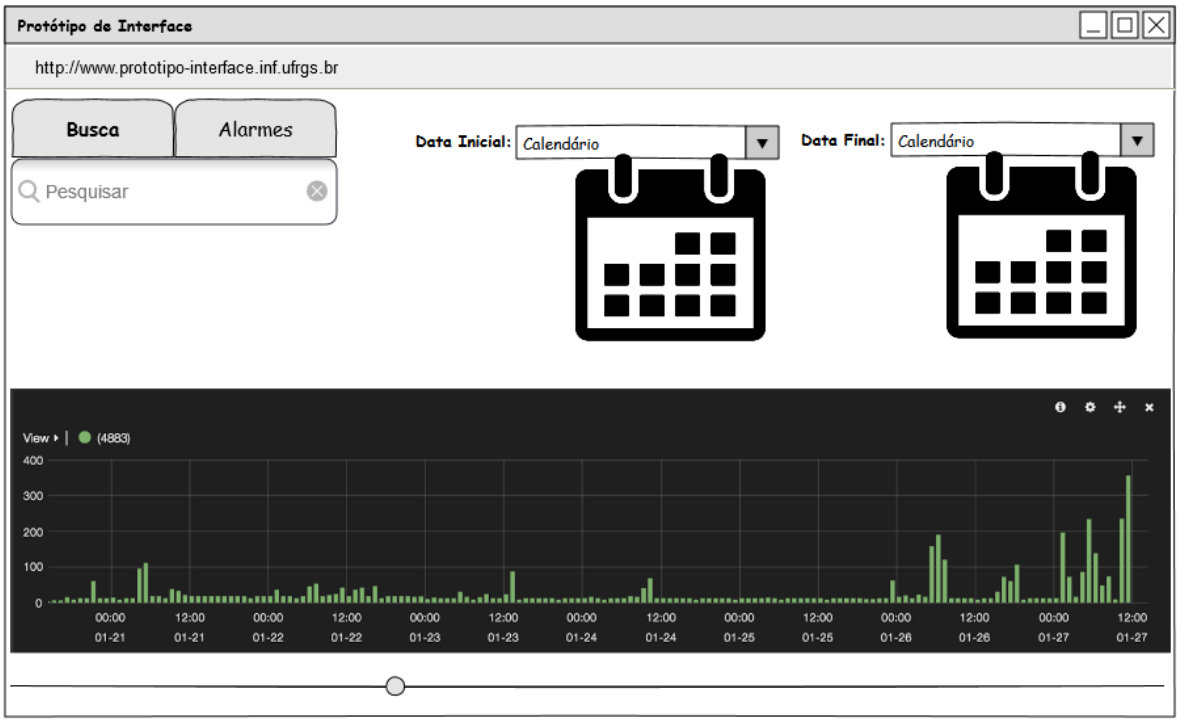
## 3.2 Interface do sistema

O objetivo da interface do sistema será visualizar os resultados das análises dos *logs*, visando a correlação visual de informação, de forma que facilite a busca por informações nos

*logs*, ajudando na detecção. É introduzido um dos principais objetivos de *Big Data*, ou seja, conseguir mostrar os resultados da análise (*Analytics*) gerando informações com *valor*.

A interface deve possuir mecanismos que facilitem a busca de informação nos *logs*, como busca por usuário e/ou endereço *IP*. Uma busca deve ser feita por meio de comandos $user == "usuario\ pesquisado"$ e/ou $ip.src == "IP\ de\ origem"$, por exemplo. As informações são expostas na interface através de histograma, onde deve ser possível analisar os *logs*, por meio de uma data inicial e uma data final. A Figura 3.1 representa o protótipo de interface para a busca de *logs*.

Figura 3.1 – Protótipo de Interface para o sistema - Busca




(Fonte: Próprio autor)

Também é desejável que o sistema sinalize, através de alarmes intuitivos, de três cores (vermelho, amarelo e verde), quando um evento suspeito é detectado. Isso é, vermelho para eventos de alta emergência, onde a infraestrutura, ou o sistema invadido, irá se tornar inutilizável; amarelo para eventos urgentes, onde uma ação imediata é necessária; e, verde para eventos de alerta, onde é detectado comportamento suspeito de usuários autorizados. O protótipo de interface da Figura 3.2 ilustra essa estrutura, onde os alarmes são visualizados de acordo com um intervalo de tempo especificado.

Figura 3.2 – Protótipo de Interface para o sistema - Alarmes


(Fonte: Próprio autor)

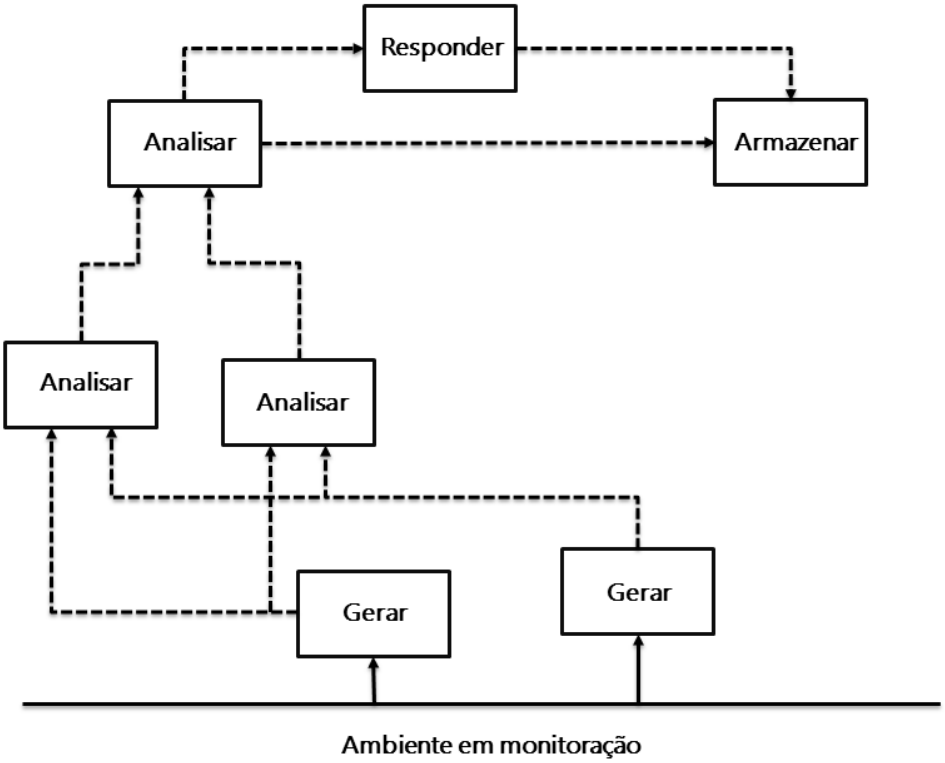
## 3.3 Arquitetura geral do sistema

Nesta seção são anunciados os princípios que conduziram a arquitetura do sistema. Inicialmente, é exposto a arquitetura de um IDS tradicional descrevendo o objetivo de seus módulos. Após isso, é apresentado a arquitetura proposta, que utiliza a fundamentação da arquitetura de um IDS tradicional, agregando módulos embasados nas fases de *Big Data*.

### 3.3.1 Arquitetura de um IDS

Tendo em vista que a arquitetura típica de um IDS é fundamentada em eventos (*logs* de segurança) (PORRAS et al., 1998), a primeira concepção de arquitetura do sistema foi baseada em (KRUEGEL; VALEUR; VIGNA, 2005) onde são encadeados módulos, ou peças de *software*, que corroboram para gerar, armazenar, responder e analisar (filtrar) os eventos. Essa arquitetura é ilustrada na Figura 3.3.

Figura 3.3 – Arquitetura de um IDS




(Fonte: Adaptado de Kruegel, Valeur e Vigna (2005))

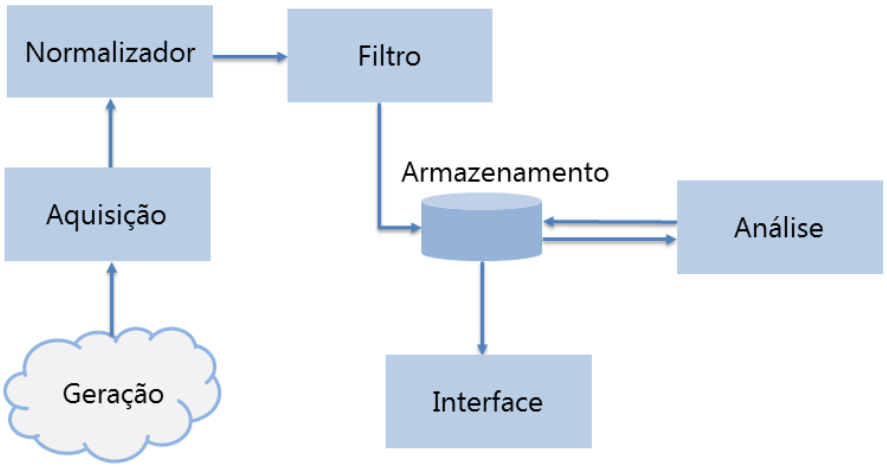
Cada um desses módulos tem propósito específico que devem satisfazer os seguintes aspectos:

- Gerar: processa os dados retirados do ambiente de rede com intuito de gerar eventos. Por exemplo, um programa que filtra os dados gerados pelo sistema operacional, ou ainda, um *sniffer* de rede que gera eventos baseados no tráfego da rede. Esse módulo tem o objetivo de adquirir o evento (*log*) que será posteriormente analisado.
- Analisar (filtrar): analisa os eventos providos por outros componentes, tipicamente representando alarmes. É possível ter na arquitetura mais de um módulo de análise. Por exemplo, um módulo que realiza a filtragem de eventos simples e que os encaminhe a outro módulo em busca de correlação nos eventos.
- Armazenar: armazena os dados garantido persistência e permitindo a análise após o acontecimento dos eventos (*postmortem*).
- Responder: responsável por reagir quando há a detecção de intrusão. Tipicamente, ações tomadas por esse componente incluem encerrar processos, reiniciar conexões de rede e modificações nas configurações do *firewall*.

### 3.3.2 Arquitetura da Proposta

Levando em conta que um dos objetivos deste trabalho está na análise de grandes volumes de *logs*, fases de *Big Data* como geração, aquisição, armazenamento e análise (*Analytics*) dos dados (HU et al., 2014) formam a base da arquitetura. Dessa forma, a arquitetura proposta, ilustrada na Figura 3.4, foi concebida incluindo módulos extras para normalizar e filtrar os *logs*.

Figura 3.4 – Arquitetura proposta




(Fonte: Próprio autor)

A nuvem de geração dos *logs* representa os equipamentos do PoP-RS. O componente de aquisição recebe esses *logs* repassando-os para o normalizador. O normalizador uniformiza o formato do *log* e encaminha para o filtro onde é feita a manipulação do *log* para ser armazenado no banco de dados (armazenamento). Esses dados são lidos do banco de dados, para ser analisados, pelo módulo de análise e, após os dados serem analisados, os resultados são novamente armazenados no banco de dados. O módulo de interface coleta os resultados do banco de dados e apresenta-os. Tipicamente, um analista de segurança, ou um administrador da rede, busca nos *logs* informações relevantes para sua análise. Essas informações relevantes variam de caso para caso. Assim, a interpretação dos dados apresentados na interface deve ser feita por um analista de segurança, ou um administrador da rede, entretanto a interpretação dos dados poderia ser feita de forma automatizada (AGRAWAL et al., 2012).

Por fim, como mencionado (seção 2.1.2), o IDS proposto não pode ser classificado como NIDS, visto que não há coleta de dados referentes ao tráfego da rede. Contudo, a solução coleta dados de equipamentos da rede, onde muitos são importantes para o funcionamento da mesma (locais estratégicos). Com isso, a classificação mais adequada para a arquitetura proposta é de

um HIDS distribuído, visto que os *logs* são coletados de hospedeiros espalhados pela infraestrutura.

## 3.4 Estudo de Viabilidade

Faz parte da proposta deste trabalho um estudo de viabilidade da solução. Um estudo de viabilidade decide se vale a pena ou não gastar tempo e esforço com sistema proposto (SOMMERVILLE, 2007). Autores como Pressman (2011) e Sommerville (2007) consideram que o estudo de viabilidade deve ser breve e focalizado, afim de verificar três aspectos:

- Se o sistema contribui para os objetivos da organização;
- Se o sistema pode ser implementado usando tecnologia atual e dentro do orçamento;
- Se o sistema pode ser integrado a outros sistemas.

Assim, durante o desenvolvimento do sistema, esses aspectos devem ser levados em consideração. Ao final do trabalho, será concluído o estudo de viabilidade.

## 3.5 Considerações finais

Neste capítulo foi apresentada a especificação do sistema a ser implementado usando as técnicas de especificações de requisitos (funcionais e não funcionais) e formalização do que é esperado da interface. Além disso, também foi apresentada a arquitetura do sistema proposto e os aspectos investigados no estudo de viabilidade. O próximo capítulo aborda as questões de desenvolvimento e implementação do sistema conforme as especificações apresentadas no presente capítulo.

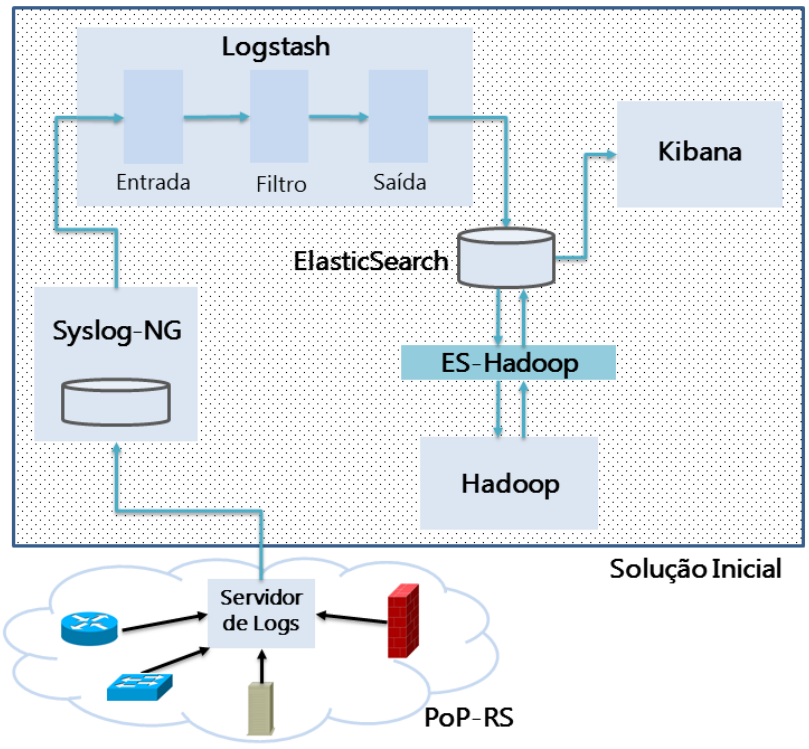
# 4 IMPLEMENTAÇÃO

Neste capítulo é apresentada a arquitetura implementada buscando atender a proposta fornecida no capítulo 3. São descritas as arquiteturas inicial e final desenvolvidas informando as ferramentas que compõe cada módulo. Assim, o desenvolvimento foi estabelecido em dois objetivos: propor a viabilização de um IDS utilizando ferramentas de *Big Data* e investigar essas ferramentas com intuito de analisar os *logs* (*Analytics*).

## 4.1 Arquitetura Inicial do sistema

Os serviços presentes na arquitetura têm o objetivo de implementar os módulos apresentados na seção 3.3.2. A Figura 4.1 ilustra a solução e o ambiente de produção da infraestrutura do PoP-RS, utilizado como entrada. As setas representam o caminho do fluxo de informações na arquitetura.

Figura 4.1 – Ambiente da arquitetura inicial implementada




(Fonte: Próprio autor)

Como já mencionado, os dados analisados são *logs* reais oriundos da infraestrutura de rede do PoP-RS. Nessa infraestrutura, há um servidor que concentra os *logs* dos equipamentos

da rede, tais como *switches*, roteadores e servidores, representado pelo servidor de *logs* na nuvem PoP-RS (Figura 4.1). Esse servidor executa o serviço *Syslog-NG* (SCHEIDLER, 2000) que é uma implementação para armazenamento de *logs* através do protocolo *Syslog* (GERHARDS et al., 2009). Nesse servidor de *logs*, foi necessário habilitar a configuração de *Relay* para a máquina repassar os *logs* para a solução implementada. Outro detalhe é que também foi necessário recompilar o serviço habilitando a *flag spoof-source* para que os *logs* retransmitidos persistissem o endereço *IP* de origem. Com isso, o servidor de *logs* do PoP-RS repassa todos os *logs* para o módulo *Syslog-NG*.

Portanto, o fluxo dos dados inicia no módulo *Syslog-NG* que recebe os *logs*. O módulo seguinte, *Logstash*, coleta os dados do módulo anterior para extração de informação relevante e salva-os no banco de dados *ElasticSearch*. o módulo *Hadoop* lê os dados do *ElasticSearch* para realizar análise (*Analytics*), com base na pré-análise feita pelo *Logstash*, e salva os resultados no *ElasticSearch*. A interface *Kibana* lê os dados do *ElasticSearch* para apresentá-los. O módulo *ES-Hadoop* conecta a solução *Hadoop* ao banco de dados *ElasticSearch*.

As características dos componentes da arquitetura inicial são detalhadas nas próximas subseções. Inicialmente são descritos as características dos módulos, qual o uso na solução e o porquê de sua escolha.

### 4.1.1 *Syslog-NG*

O protocolo *Syslog* tem como base duas informações: o tipo de evento, representado pela facilidade (*facility*), e a severidade do evento (*severity*) (LONVICK, 2001). Comumente, os servidores, roteadores e *switches* implementam esse protocolo para manipular seus registros de auditoria. As tabelas 4.1 e 4.2 apresentam as *severities* e *facilities*, respectivamente.

Tabela 4.1 – Severidade do evento

| **Severidade** | **Descrição** | **Valor** |
|---|---|---|
| emergência | O sistema está inutilizável | 0 |
| alerta | Ação imediata é necessária | 1 |
| crítico | Condições críticas | 2 |
| erro | Condições de erro | 3 |
| perigo | Condições de alerta | 4 |
| notificação | Condição normal, mas significativa | 5 |
| informação | Mensagem informaiva | 6 |
| *debugging* | Mensagem de depuração | 7 |

Tabela 4.2 – Tipos de eventos (*facility*)

| **Facilidade** | **Descrição** | **Valor** |
|---|---|---|
| *Kernel* | Mensagens do Kernel | 0 |
| *User* | Mensagens de nível usuário | 1 |
| *Mail* | Sistema de e-mail | 2 |
| *Daemons* | Serviços do Sistema Operacional | 3 |
| *Auth* | Mensagens de segurança e autenticações/autorizações | 4 |
| *Syslog* | Mensagens internas do syslog | 5 |
| *LPR* | Subsistema impressão | 6 |
| *News* | Subsistema de *News* | 7 |
| *UUCP* | Subsistema UUCP | 8 |
| *Cron* | Mensagens do serviço Cron | 9 |
| *Auth2* | Mensagens de segurança e autenticações/autorizações | 10 |
| *FTP* | Serviço transferência de arquivos | 11 |
| *NTP* | Serviço NTP | 12 |
| *Audit* | Mensagens do sistema de auditoria | 13 |
| *Console* | Mensagens da console do sistema | 14 |
| *Cron2* | Serviço Cron | 15 |
| Local0 | Uso local 0 para usuário | 16 |
| Local1 | Uso local 1 para usuário | 17 |
| Local2 | Uso local 2 para usuário | 18 |
| Local3 | Uso local 3 para usuário | 19 |
| Local4 | Uso local 4 para usuário | 20 |
| Local5 | Uso local 5 para usuário | 21 |
| Local6 | Uso local 6 para usuário | 22 |
| Local7 | Uso local 7 para usuário | 23 |

O *Syslog-NG* (*New Generation*) (SCHEIDLER, 2000) é uma implementação do protocolo *Syslog* e apresenta como principal evolução a possibilidade de organizar os eventos por seu conteúdo e não somente por facilidade e severidade. Isso é possível através do uso de expressões regulares para realizar a filtragem das mensagens presentes nos eventos e definir para onde enviá-las.

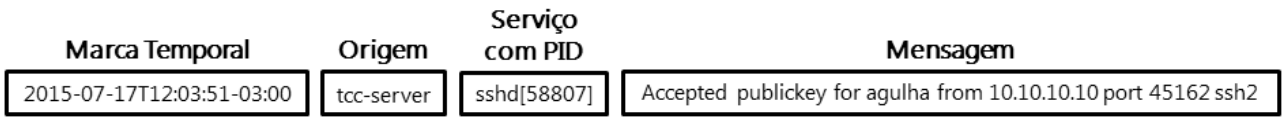
A arquitetura do *Syslog-NG* é baseada em 4 componentes: origem (*source*), destino (*destination*), filtro (*filter*) e *log*. Em *source* é definido de onde o *Syslog-NG* recebe as mensagens. No campo *destination* é determinado para onde o *Syslog-NG* enviará as mensagens. Os *filters* podem ser combinados e podem utilizar expressões regulares para separar as mensagens. A diretiva *log* combina os componentes anteriores, representando uma ação que descreve o que vier de *source* e satisfizer um determinado *filter* deverá ser encaminhado para *destination*. É

possível especificar mais de uma origem para um determinado destino, ou mais de um destino, para mensagens de uma determinada origem numa mesma instrução.

A solução *Syslog-NG* auxilia na centralização de *logs*. Segundo Santos et al. (2015), no contexto de uma infraestrutura de redes, o modelo de gerência de *logs* centralizado é desejável por dois motivos: (i) pela praticidade de ter os *logs* de toda rede em um servidor, permitindo que as consultas em busca de problemas fiquem simples e fáceis, ou seja, é uma solução que escala frente a quantidade de *logs*; (ii) pela segurança, pois torna-se um *Bastion Host*, ou seja, uma máquina fora do padrão daquela rede (com sistema operacional diferente, e/ou senhas diferentes das demais, por exemplo). Assim, caso um invasor viole um servidor da rede, as informações da invasão estarão centralizadas no servidor de *log*.

Com isso, o *Syslog-NG* é a centralização dos registro de auditoria da solução, sendo responsável pela aquisição e classificação dos eventos segundo sua *facility*. O formato que os eventos são armazenados no *log* é ilustrado na Figura 4.2. Os conteúdos nos campos de marca temporal, origem e serviço com *PID*, são criados e escritos automaticamente pelo *Syslog-NG*. O conteúdo no campo mensagem não é criado pelo *Syslog-NG*, pois trata-se de uma cópia do conteúdo do evento gerado no hospedeiro origem.

Figura 4.2 – Exemplo de formato do *log* no *Syslog-NG*




(Fonte: Próprio autor)

Assim, a solução implementada inicia por um módulo que executa o serviço *Syslog-NG* para classificar os dados por *facilities* segundo sua categoria de evento (Tabela 4.2). Esses dados são salvos localmente no formato apresentado na Figura 4.2. Por exemplo, cada *log* pertencente a *facility* de autenticação (*Auth* e *Auth2*) que entra no sistema é absorvido pela solução *Logstash* (próximo módulo).

A infraestrutura do PoP-RS tem um servidor de centralização de *logs* que executa o serviço *Syslog-NG*. Assim, a principal motivação para a escolha do *Syslog-NG* foi não interferir, significativamente, na estrutura de armazenamento de eventos da infraestrutura de produção do PoP-RS. Com isso, os eventos que chegam no servidor de *logs* do PoP-RS são repassados ao módulo *Syslog-NG* da solução. A compatibilidade entre o serviço nativo do PoP-RS e o serviço que constrói a solução também foi levada em consideração para a escolha desse módulo, diminuindo as chances de incompatibilidade com o serviço em produção. Outro motivo para o

uso do *Syslog-NG* deve-se a habilidade e treinamento pessoal na configuração do *software*, visto que se trata de um serviço já em produção. Contudo, poderia ter sido usado outra solução como o *software Flume* que permite a agregação e movimentação de dados de *log* para o *Hadoop*.

### 4.1.2 *Logstash*

O *Logstash* (LOGSTASH, 2015) é uma ferramenta que pertence ao ecossistema *Elastic* (seção 2.3) e possui características de centralização, agregação e filtragem de dados. Há dois problemas que o *Logstash* procura solucionar: praticidade e diversidade.

Inicialmente, o *Logstash* busca tornar o gerenciamento de *log* mais prático e fácil nos ambientes computacionais, ou seja, aumentar a praticidade. De fato, os administradores da rede, frequentemente, fazem uso de ferramentas clássicas como *cat*, *tail*, *sed awk* e *grep* para examinar os registros de *log*. O uso dessas ferramentas não é prático para um número maior de máquinas e tipos variados de *logs* (TURNBULL, 2014). Como dito, a praticidade, ou seja, *logs* de muitos equipamentos da rede, pode ser resolvida com um modelo de gerenciamento centralizado, como o *Syslog-NG*. Assim, o administrador conecta-se em apenas um servidor, ao invés de N servidores, para analisar os *logs*.

Contudo, essa solução não resolve o problema de tipos variados de *logs*, isto é, diversidade de eventos. Atualmente, por exemplo, há um vasto número de diferentes tipos de eventos, formatos e fusos horários presentes nos *logs*, ou seja, basicamente, há inexistência de um padrão universal. Através de sua arquitetura, o *Logstash* tenta resolver o problema da diversidade de *logs*, filtrando e analisando, as informações úteis contidas no *log*.

Assim, a arquitetura do *Logstash* divide-se em três módulos: entrada, filtro e saída. O módulo de entrada diz respeito por onde os dados chegam no *Logstash*. Na solução, a entrada está no local onde o componente anterior (*Syslog-NG*) salva os *logs*. Os filtros do *Logstash* são fornecidos em um, ou mais, arquivos de configuração, assim, caso seja necessário alterar um determinado filtro, a reconfiguração é feita em apenas um arquivo. O fluxo dos dados passa por todos os filtros na ordem em que aparecem. Por fim, a saída diz respeito ao local onde o dado final será salvo, que no caso da solução, é uma linha simples de configuração indicando o banco de dados (*ElasticSearch*).

Na realidade o *Logstash*, em tese, poderia substituir o serviço *Syslog-NG* e receber diretamente do servidor PoP-RS, mas, por questões de compatibilidade e baixa interferência na solução existente na infraestrutura de produção do PoP-RS, essa funcionalidade não foi usada.

Enfim, na solução implementada, o *Logstash* não é utilizado como um centralizador de dados. Logo, o *Logstash* é utilizado como uma ferramenta de agregação e filtragem de informações. Na filtragem, o *Logstash* seleciona os eventos relevantes que serão analisados. Na agregação de informação, o *Logstash* acrescenta *valor* aos eventos com intuito de adicionar informações relevantes, como *IP* de origem e/ou usuário usado na tentativa de *login*, no registro que será armazenado no banco de dados. Por exemplo, um filtro pode identificar se uma tentativa de conexão foi estabelecida com sucesso ou, se há muitas tentativas de acesso com erros de senha, ou usuário, o que poderia caracterizar uma possível invasão por tentativa e erro.

A Listagem 4.1 fornece a implementação de um filtro do *Logstash* com três regras para detectar essas três ações. Esse filtro está implementado na solução proposta para tratar eventos de tentativas de autenticação, via protocolo *ssh*. Resumidamente, esse filtro é usado para adicionar informações de valor, tiradas dos eventos.

```
1  filter{
2    grok {
3       match => { "message" => "Accepted %{WORD:auth_method} for %{USER:username} from %{IP:
       src_ip} port %{INT:src_port} ssh2" }
4        add_field => {
5          "ssh_type_conection" => "ssh_sucessful_login" }
6       }
7
8    grok {
9       match => {"message" => "Failed password for invalid user %{USER:username} from %{IP:
       src_ip} port %{INT:port} ssh2"}
10       add_field => {
11         "ssh_type_conection" => "ssh_brute_force_attack" }
12      }
13
14   grok {
15      match => { "message" => "Failed password for %{USER:username} from %{IP:src_ip} port %{
       INT:port} ssh2"}
16       add_field => {
17         "ssh_type_conection" => "ssh_failed_login" }
18     }
19 }
```

Listagem 4.1 – Filtro do *Logstash* para *logs* de autenticação *ssh*.

A linha 1 indica o início do filtro. Nas linhas seguintes (2 e 3), através da diretiva *grok*, é feito um *parsing* dos eventos de entrada. A diretiva *match* é utilizada para verificar se o evento recebido (*message*) ocorre na descrição indicada, ou seja, é verificado se o *log* recebido equivale a formatação designada pelo indicador =>.

Assim, caso a combinação ocorra com sucesso, uma série de informações de valor são extraídas do *log*, de acordo com a sua posição na mensagem. Essas informações de valor

são obtidas por meio de diretivas simples na forma *%(tipo:nome)*, onde é designado o tipo do dado e o nome da variável. Na linha 3, são extraídas as informações refentes ao método de autenticação (*%WORD:auth_method*), usuário (*%USER:username*), endereço IP da requisição (*%IP:src_ip*) e número de porta TCP/UDP (*%INT:src_port*). Na linha 4 é adicionado um campo para a variável *ssh_type_conection*, armazenando o valor *ssh_sucessful_login*. Desse maneira, é realizado a agregação de informação referente ao tipo de mensagem filtrada pelo *Logstash*.

Nos casos seguintes, para *ssh_brute_force_attack* (linhas 8-12) e *ssh_failed_login* (linhas 14-18), a metodologia aplicada anteriormente é a mesma. O que muda são os valores extraídos do *log*. Então, a primeira etapa para fazer novas regras é implementar um filtro como o da Listagem 4.1.

Após passar por todos os filtros, o *Logstash* prepara o evento de *log* em um formato JSON com os valores adicionados e o envia ao *ElasticSearch*. Um exemplo de evento formatado para envio é visualizado na Listagem 4.2.

```
1  {
2    "message" => "2015-07-17T12:03:51-03:00 tcc-server sshd[58807]: Accepted publickey for
       agulha from 10.10.10.10. port 45162 ssh2",
3    "@timestamp" => "2015-07-17T15:03:51.000Z",
4    "@version" => "1",
5    "auth_method" => "publickey",
6    "username" => "agulha",
7    "src_ip" => "10.10.10.10",
8    "src_port" => "45162",
9    "timestamp" => "17/Jul/2015 12:03:51 -0300",
10   "target" => "tcc-server",
11   "service" => "sshd",
12   "pid" => "58807",
13 }
```

Listagem 4.2 – Exemplo de formato de evento que será armazenado no *ElasticSearch*.

O *Logstash* não é a única ferramenta de filtragem e agregação de informações existentes em *software* livre. Dentre elas, no decorrer deste trabalho foram investigadas o *Fluentd* (FLUENTD, 2015) e o *Flume* (FLUME, 2015), porém essas soluções não são integradas nativamente ao ecossistema dos módulos subsequentes. Assim sendo, a escolha do *Logstash* se deve ao fato dessa solução também ser usada em conjunto dos módulos *Kibana* e *ElasticSearch* sendo comumente chamada de Pilha ELK (LAHMADI; BECK, 2015; APPLEYARD; ADAMS, 2015).

### 4.1.3 *ElasticSearch*

O *ElasticSearch* é um banco de dados orientado a documentos frequentemente listado como uma das ferramentas de *Big Data* (BAI, 2013). É baseado no *Apache Lucene* (LUCENE, 2015) sendo desenvolvido em código Java, porém com uma estrutura projetada para ser acessível por qualquer linguagem capaz de se comunicar por REST/HTTP. Para favorecer os programadores, existe uma série de clientes que encapsulam as operações, que normalmente seriam realizadas através de chamadas REST. Assim, o objetivo do *ElasticSearch* é apoiar o desenvolvimento de aplicações centradas em texto, como redes sociais, sistemas de *e-commerce*, *sites* de notícias e canais de educação. Uma grande vantagem do *ElasticSearch* é a sua arquitetura projetada para ser escalável e para gerenciar grandes quantidades de dados de modo simples e eficiente (GORMLEY; TONG, 2015).

Por ser um banco de dados NoSQL, conceitos principais como índice, indexação e análise merecem destaque. O local onde é realizado o armazenamento dos documentos enviados ao *ElasticSearch* é chamado de índice. Diferentemente dos bancos de dados tradicionais, as informações contidas no documento não são armazenadas na mesma forma que são enviadas ao *ElasticSearch*, porém, antes, são analisadas e transformadas. O termo indexação refere-se a esse processo de armazenamento de um documento. A análise das informações é o que torna o *ElasticSearch* poderoso, através dela é possível comparar partes de texto e modificar as palavras contidas em um documento para que torne mais fácil encontrar termos semelhantes entre as palavras.

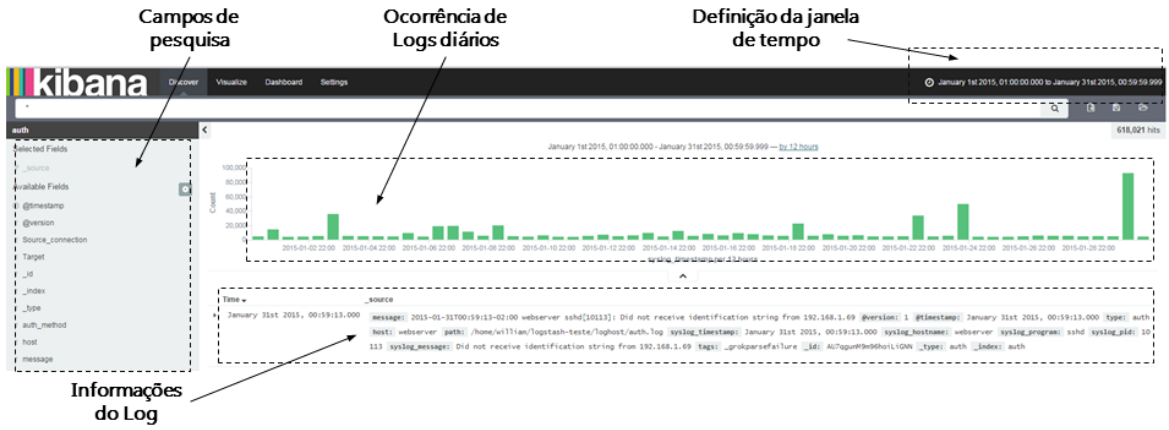
Além dessas vantagens, o *ElasticSearch* tem uma fácil integração com as ferramentas *Logstash* e *Kibana* (Pilha ELK). A saída do *Logstash* é direcionada ao banco de dados *ElasticSearch*. Pelo fato de ambas ferramentas pertencerem ao mesmo ecossistema, a integração entre elas é feita de forma simples. O *ElasticSearch* armazena os dados utilizando formato JSON que, posteriormente, será lido pela interface *Kibana*.

### 4.1.4 *Kibana*

Por fim, o *frontend* da Pilha ELK é o *software Kibana*. Tipicamente, a ferramenta *Kibana* é utilizada para análise por inspeção manual e visualização de informações presentes no *ElasticSearch* (MARINO, 2015; VAARANDI; PIHELGAS, 2014). Dessa forma, é ela que irá apresentar os dados armazenados pelo *Logstash* no *ElasticSearch*, em uma interface, via *browser*, altamente customizável com histogramas e outros painéis que propiciam uma visão

geral sobre os dados. O *Kibana* possibilita transformar os *logs* em informações úteis (*valor*) através de *Dashboards*, pois permite realizar correlação de eventos, filtrar *logs* por origem, hospedeiros, entre outras combinações (VAARANDI; NIZIŃSKI, 2013).

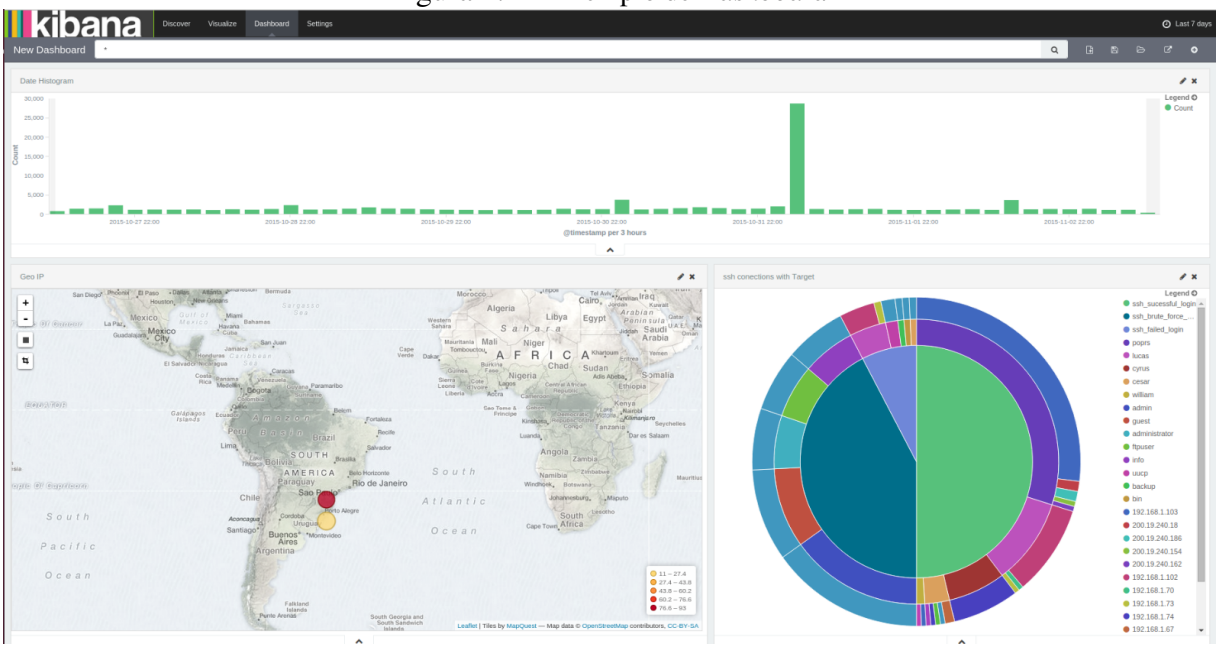
Figura 4.3 – Tela inicial do *Kibana*




(Fonte: Próprio autor)

De modo ilustrativo, a tela inicial do *Kibana* pode ser visualizada na Figura 4.3. Nessa tela, o histograma representa a distribuição diária de *logs* gerados em um intervalo mensal. A janela de tempo pode ser configurada a critério do usuário. Abaixo do histograma são listadas as informações presentes em todos *logs*.

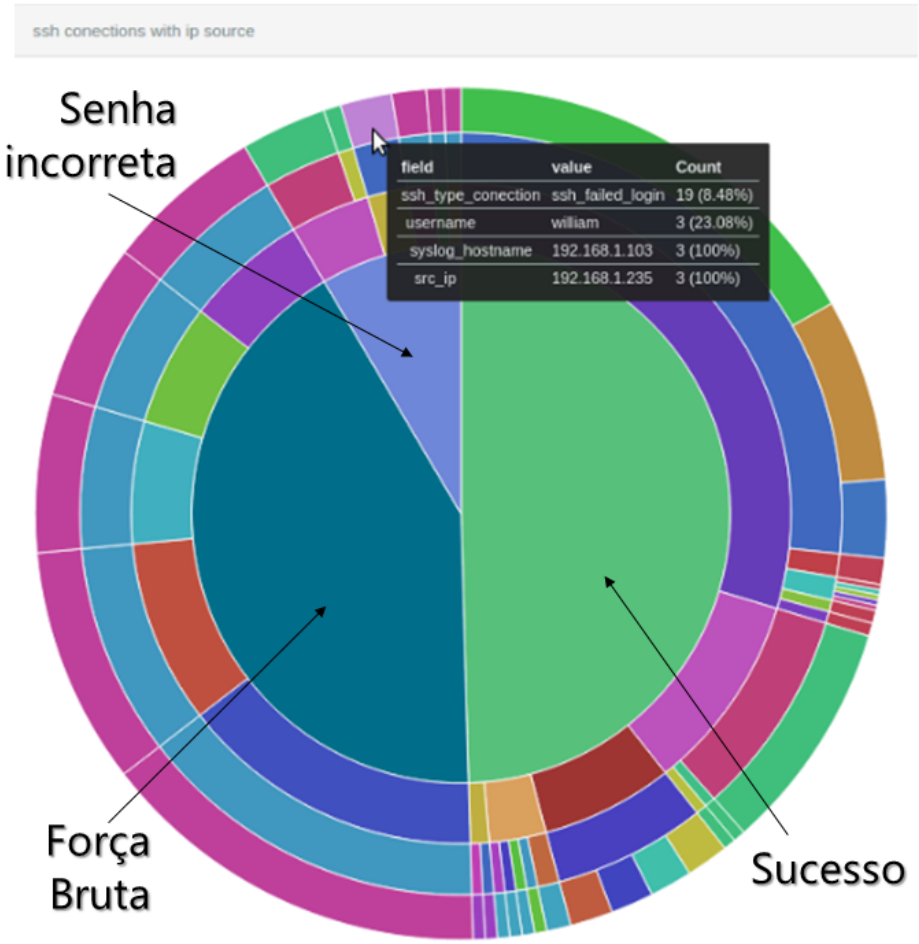
Figura 4.4 – Exemplo de *Dashboard*




(Fonte: Próprio autor)

Como dito, as informações úteis, previamente analisadas e transformadas pelo *Logstash*, podem ser visualizadas em gráficos e organizadas em uma *Dashboard*. Um exemplo de *Dashboard* é visto na Figura 4.4. Nela, está presente o histograma, geolocalização por *IP* e gráfico de correlação de eventos em pizza. Um aspecto interessante é que ao clicar em um valor específico, ou seja, filtrando um determinado valor, todos os gráficos se modificam para aquele filtro estabelecido.

Figura 4.5 – *Zoom* no gráfico de pizza *ssh*




(Fonte: Próprio autor)

Uma versão mais detalhada do gráfico de correlação em pizza é dada na Figura 4.5. São listados os acessos via protocolo *ssh* nos equipamentos da infraestrutura do PoP-RS, onde o interior do gráfico é representado por três tipos de conexão. Esses tipos de conexão foram transformados pelo *Logstash*, através do filtro da Listagem 4.1, e representam acesso por força bruta, senha incorreta e *login* bem sucedido. Nos níveis externos são visualizadas as informações referentes aos usuários, aos equipamentos, onde ocorreu o evento, e aos endereços *IP* de origem do acesso ou tentativa.

De uma certa forma, a ferramenta *Kibana* representa os princípios de *Analytics* da solução, ou seja, extrair informações de valor de uma grande quantidade de dados. Por pertencer ao mesmo ecossistema das duas ferramentas anteriores, o *Kibana* foi escolhido para ser a interface do sistema.

### 4.1.5 *Hadoop*

Extrair, de forma automática, informações relevantes de uma grande quantidade de *logs*, é o principal objetivo de se ter um *MapReduce* na arquitetura desta solução. A ferramenta em *Big Data* que implementa o paradigma *MapReduce* escolhida foi o *Apache Hadoop* (DEAN; GHEMAWAT, 2008). O objetivo da ferramenta está em analisar grandes quantidades de dados através de processamento em lote (*batch*), sendo utilizado em muitos cenários, como análise exploratória de redes sociais (BORTHAKUR et al., 2011) e análise de sequenciamento de genes (TAYLOR, 2010), por exemplo. Para conseguir ser tão genérico, a estrutura do *Hadoop* divide-se em armazenamento dos dados e processamento de informações. O armazenamento das informações é nativamente feito pelo HDFS (*Hadoop Distributed File System*). O processamento de informações é feito pelo *MapReduce*.

Nesta proposta, a implementação de uma solução baseada em *MapReduce* para extrair informações relevantes foi feita utilizando o *Hadoop*. Para isso, foi necessário configurar adequadamente suas variáveis de ambiente.

A principal motivação para a escolha do *Hadoop*, além de ser uma ferramenta *open source*, se deve ao fato de seu ecossistema ser muito rico (seção 2.3), sendo uma vantagem que auxilia o desenvolvimento devido a uma variada gama de *frameworks*. O processamento dos dados no *Hadoop* é feito em *batch*, indicado para a análise *postmortem*.

O arquivo que representa a configuração inicial para o *MapReduce*, no *Hadoop*, é dado na Listagem 4.3. As linhas iniciais (1-11) apresentam os pacotes importados. Nas linhas 16 a 21 são realizadas as configurações do local que será capturado as informações que alimentarão o *MapReduce*. Nesse exemplo, os dados são configurados para o conector *ES-Hadoop* (seção 4.1.6), passando um valor de *index* e tipo (Linha 17). Os demais detalhes de nomeação das classes de mapeamento e redução, estão documentados no código na forma de comentários.

```
import java.io.IOException;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.MapWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.elasticsearch.hadoop.mr.EsInputFormat;

public class ElasticSourceHadoopSinkJob {
   public static void main(String arg[]) throws IOException, ClassNotFoundException,
    InterruptedException{

   Configuration conf = new Configuration();
   conf.set("es.resource", "logstash-*/*"); //index/type

   // final Job job = new Job(conf,"AuthLog");
   Job job =  Job.getInstance(conf);
   job.setJarByClass(ElasticSourceHadoopSinkJob.class);

   //classes map e reduce
   job.setMapperClass(ElasticSourceHadoopSinkMapper.class);
   job.setReducerClass(ElasticSourceHadoopSinkReducer.class);

   //formato dos dados de entrada
   job.setInputFormatClass(EsInputFormat.class);

   //formato dos dados de saida
   job.setOutputFormatClass(TextOutputFormat.class);

   //formato dos dados de saida do Mapper
   job.setMapOutputKeyClass(Text.class);
   job.setMapOutputValueClass(IntWritable.class);

   //formato dos dados de saida do Reduce (caso necessario)
   job.setReduceOutputKeyClass(Text.class);
   job.setReduceOutputValueClass(IntWritable.class);

   //informa no comando onde sera gravado os dados no hdfs
   FileOutputFormat.setOutputPath(job, new Path(arg[0]));

   System.exit(job.waitForCompletion(true) ? 0 : 1);
   }
}
```

Listagem 4.3 – Configuração para inicialização do *MapReduce*

Como forma de exemplificar o uso de uma operação de *map*, a Listagem 4.4 mostra o mapeamento de chaves e valores retirados do *ElasticSearch*. As linhas 25 a 27 exemplificam a operação contendo o valor *timestamp* como chave e *user*, *src_ip* e *target* como valores. As linhas anteriores, onde é feita a busca no *ElasticSearch*, estão documentadas no código na forma de comentários.

```
1  import java.io.IOException;
2  import org.apache.hadoop.io.MapWritable;
3  import org.apache.hadoop.io.Text;
4  import org.apache.hadoop.mapreduce.Mapper;
5  import org.apache.hadoop.io.ArrayWritable;
6  import org.apache.hadoop.io.IntWritable;
7  import org.elasticsearch.hadoop.mr.WritableArrayWritable;
8  import org.apache.hadoop.util.Tool;
9
10 public class ElasticSourceHadoopSinkMapper extends Mapper <Object, MapWritable, Text,
       IntWritable> {
11   private final static IntWritable one = new IntWritable(1);
12   @Override
13    protected void map(Object key, MapWritable value, Context context) throws IOException,
       InterruptedException {
14
15    Text documentId = (Text)key;
16    MapWritable valueMap = (MapWritable)value;
17
18    //Busca no ElasticSearch os campos necessarios para as analises
19    Text user = (Text)valueMap.get(new Text("username"));
20    Text target = (Text)valueMap.get(new Text("syslog_hostname"));
21    Text timestamp = (Text)valueMap.get(new Text("syslog_timestamp"));
22    Text src_ip = (Text)valueMap.get(new Text("src_ip"));
23
24    //escreve o que vai para o output <key,value> no contexto do map, (usado no reduce depois)
       tendo como chave o timestamp
25    context.write(timestamp, user);
26    context.write(timestamp, src_ip);
27    context.write(timestamp, target);
28    }
29 }
```

Listagem 4.4 – Exemplo de implementação de *Map*

### 4.1.6 *ES-Hadoop*

Como a solução *Big Data* escolhida é o *Hadoop*, foi necessário uma ferramenta que conectasse a Pilha ELK (*Elastic*) ao ecossistema *Apache Hadoop*. Assim, o *ES-Hadoop* (ES-HADOOP, 2015a) serve como um conector entre o módulo *ElasticSearch* e o módulo *Hadoop*,

possibilitando a comunicação e troca de dados entre os módulos. Ele foi escolhido por ser uma solução pertencente ao ecossistema *Elastic* e também por suportar outras ferramentas do ecossistema *Apache Hadoop*, além do *MapReduce*, como *Hive* e *Pig*.

Dessa forma, os dados filtrados pelo *Logstash* armazenados no *ElasticSearch* poderiam ser utilizados já pré-analisados, e as análises feitas com auxílio do *Hadoop* poderiam ser salvas no *ElasticSearch* para, posteriormente, serem visualizadas pela interface *Kibana*. Assim sendo, há dois fluxo de dados, um que se origina no módulo e outro que se destina ao módulo. Esses fluxos foram representados na Figura 4.1 da arquitetura por meio das setas que chegam e saem do módulo.

## 4.2 Arquitetura Final do sistema

Como visto nas seções anteriores, programar com objetivo de analisar os *logs* é um processo não trivial. Além da implementação de toda lógica de processamento usando somente métodos *map* e *reduce*, ainda existe a configuração inicial da estrutura e de seu conector. Assim, no decorrer do desenvolvimento optou-se por utilizar uma outra solução do ecossistema *Apache Hadoop*, para minimizar dificuldades de se implementar o paradigma *MapReduce*.

A alternativa encontrada foi usar uma ferramenta do ecossistema *Apache Hadoop* que abstrai a lógica de processamento paralelo sem a necessidade de utilizar funções *map* e *reduce* explicitamente, chamada *Pig* (PIG, 2015). Dessa forma, o módulo *Hadoop*, da arquitetura inicial do sistema (Figura 4.1), recebeu uma camada superior de abstração.

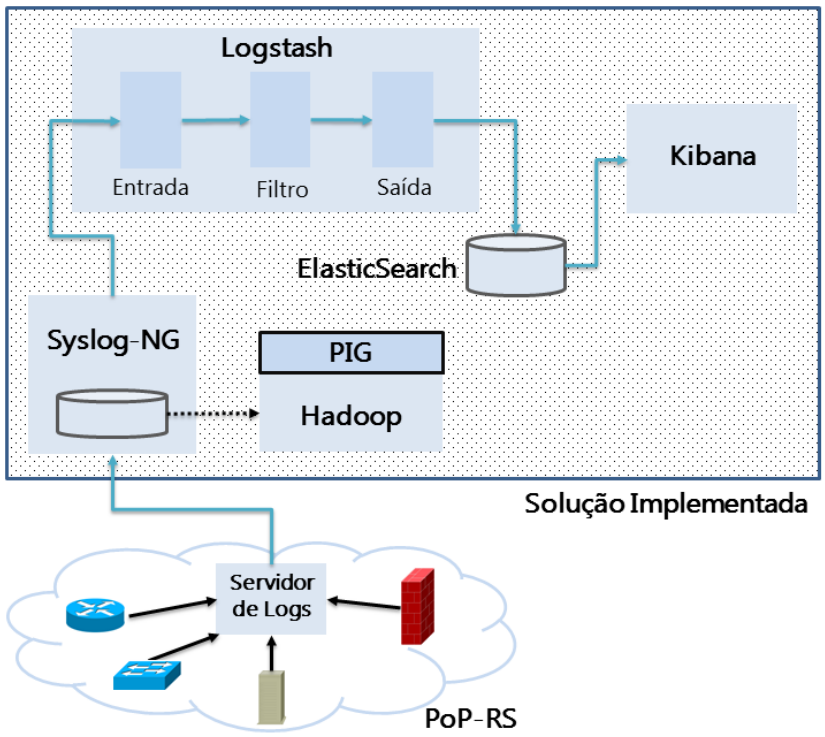
Essa mudança na arquitetura influenciou no funcionamento do conector *ES-Hadoop*. A versão do conector *ES-Hadoop* utilizada foi a 2.1.0 e, durante o desenvolvimento deste trabalho, foram encontrados dois problemas que impossibilitaram o conector transmitir os dados presentes no *ElasticSearch* ao *Pig*. O primeiro problema está relacionado ao mapeamento do *ElasticSearch* (*mapping*) realizado com o pacote externo distribuído com o conector para utilização com o *Pig*. Não foi possível carregar os dados por meio de seus valores (*username* e *timestamp*, por exemplo) adicionados pelos filtros do *Logstash* (Listagem 4.1). Assim como, não foi possível o envio de informações do *Pig* ao *ElasticSearch*.

O segundo problema, mais crítico, refere-se ao não carregamento dos dados em *Script Pig Latin* por meio da instrução *LOAD*. Assim, os dados que estavam no *ElasticSearch* não eram transferidos à aplicação em *Pig*. No decorrer da escrita deste trabalho, esse problema na operação de *LOAD* com *Pig* foi corrigido na versão 2.1.1 do conector (ES-HADOOP, 2015b). Contudo, não houve menção de previsão para resolução do problema envolvendo o *mapping*

dos dados do *ElasticSearch* para o *Pig*.

A consequência dos dados provenientes do *Pig* não serem transferidos ao *ElasticSearch* foi a inviabilização do uso do *Kibana* para visualização dos dados processados pelo *Pig*. Ainda em decorrência do problema de transferência de dados entre o *Pig* e o *ElasticSearch*, optou-se por coletar os *logs* diretamente no armazenamento do *Syslog-NG*. Assim, o conector *ES-Hadoop* acabou sendo desnecessário na arquitetura final implementada, como pode ser visto na Figura 4.6.

Figura 4.6 – Ambiente da Arquitetura final implementada




(Fonte: Próprio autor)

Consequentemente, a arquitetura final implementada foi segmentada em dois caminhos distintos. O primeiro caminho, representado pelas setas contínuas, inicia no módulo *Syslog-NG* que recebe os *logs*. O *Logstash* (módulo seguinte) coleta os dados do módulo anterior para filtragem e agregação de informações (pré-análise) salvado-os no banco de dados *ElasticSearch*. A interface *Kibana* lê os dados do *ElasticSearch* para apresentá-los. Esse é o caminho para análise visual e manual dos *logs*. No segundo caminho, representado pela seta tracejada, os dados são absorvidos pelo módulo *Pig* diretamente do componente *Syslog-NG*, apenas para fins de validação, não possuindo uma interface gráfica, ou seja, a visualização dos resultados é feita por inspeção em arquivos texto.

Ao usar o *Pig*, é criado um nível de abstração ao paradigma *MapReduce*. O programador ganha em produtividade na escrita dos programas através de instruções em uma linguagem *script* procedural, de comandos simples, que são posteriormente transformados em *jobs map* e *reduce*. Por outro lado, devido à camada de abstração, há uma perda em desempenho frente a chamadas *MapReduce*. Todo *script* é executado em um *bash* próprio do *Pig* chamado *grunt*.

A motivação para o uso do *Pig* decorre dos desafios de escrever programas que manipulam dados através do *MapReduce* no *Hadoop*, ou seja, implementar a lógica de distribuição de execução usando somente uma função de mapeamento e uma função de redução. Em processamentos complexos, é comum o programador precisar dividir o processamento em várias operações *MapReduce* e depois encadeá-las para atingir o resultado desejado.

Assim, ao invés de usar apenas as funções de mapeamento e de redução, o *Pig* permite definir o processamento como uma série de transformações por onde os dados fluem para produzir a saída desejada através de uma linguagem própria chamada *Pig Latin*. Tipicamente, um *script* em *Pig Latin* segue três passos específicos: os dados são lidos do sistema de arquivos; operações são executadas nos dados (transformando-os uma ou mais vezes); e, em seguida, a relação resultante é salva no sistema de arquivos do *Hadoop*. Esses passos são transformados em operações do tipo *map* e *reduce* de forma transparente para o programador.

Como mencionado, a escrita de programas *MapReduce* (codificados em linguagem JAVA) é mais complexa. Uma amostra dessa complexidade pode ser vista comparando os códigos presentes nas Listagens 4.3 e 4.4, onde é feito a configuração da estrutura e uma operação simples de *map*, com o *script* em linguagem *Pig Latin* usado neste trabalho, mostrado na Listagem 4.5.

```
1 REGISTER /usr/lib/pig/pig-0.15.0/lib/piggybank.jar;
2 logs = LOAD '/home/hduser/logsAuth/auth*.log' USING org.apache.pig.piggybank.storage.
    MyRegExLoader('^(\\S+)\\s+(\\S+)\\s+(\\w+)\\S+\\s+([\\S+\\s+]*)\$') AS (timestamp:
    chararray, host:chararray, program:chararray, message:chararray);
3 searchData = FILTER logs BY message MATCHES '.*agulha.*';
4 STORE searchAdata INTO '$myOutput/' USING PigStorage(' ');
```

Listagem 4.5 – *Script* em *Pig Latin*

No exemplo da Listagem 4.5, na linha 1, a instrução *REGISTER* é usada para registrar um pacote externo. Há um variado conjunto de pacotes externos que podem ser utilizados para manipulação de *log*. O pacote *piggybank.jar* é utilizado em formatos de *logs* da *Apache*. Para análise de *logs* de acesso em servidores *web*, o *piggybank* é utilizado com a função *CombinedLogLoader()* (PIGGBANK, 2015), por exemplo. O uso desses pacotes facilita a captura de informações, visto que ele utiliza padrões já esperados nos *logs*. Contudo, neste trabalho, o formato do evento armazenado no *log* contém quatro campos construídos pelo *Syslog-NG* (Figura 4.2). Neste caso particular não foi encontrado um pacote que atendesse o formato do *Syslog-*

*NG*. Assim, na linha 2, é utilizado a função *MyRegExLoader()* para descrever uma expressão regular para carregar o *log*.

Ainda na linha 2, é realizado o carregamento dos dados através da instrução *LOAD*. São informados o caminho dos dados e qual pacote será utilizado para carregamento. A instrução *USING* informa a função no pacote *piggbank* e a expressão regular utilizada para descrever os campos *timestamp*, *host*, *program* e *message*. O motivo para usar o pacote *piggybank* é a possibilidade de escrever expressões regulares para coleta dos dados. Em seguida, os campos atribuídos pela expressão regular são catalogados por nome e tipo por meio da instrução *AS*, formando o *schema*. Na linha 3 é realizado um filtro, em um campo específico do *schema* (no exemplo, o campo *message*), por uma palavra pesquisada através das instruções *FILTER*, *BY* e *MATCHES*. Por fim, na linha 4, os dados são gravados no sistema de arquivos do *Hadoop* (HDFS), por meio da instrução *STORE*. São informados o caminho dos dados e qual pacote será utilizado para gravação através da instrução *USING*. Nesse caso, foi passado como parâmetro o local para salvar os dados quando o *script Pig* foi executado ($myOutput/$).

Todas as instruções *Pig Latin* operam em relações (chamadas de operadores relacionais). Como mostrado, há um operador para carregar dados e armazenar dados no arquivo de sistema. Existe uma forma para filtrar dados iterando as linhas de uma relação. Essa funcionalidade é comumente usada para remover dados da relação que não são de fato para operações subsequentes. Como alternativa, pode-se iterar as colunas de uma relação, invés das linhas, através da operação *FOREACH* que permite operações aninhadas como *FILTER*. Ainda é possível ordenar os dados por meio da operação *ORDER* durante a iteração.

Há um variado conjunto de tipos de dados no *Pig*, suportando não só conceitos de alto nível como pacotes, como também tipos de dados simples como *ints, longs, floats, doubles, chararrays e bytearrays*. Também há uma variedade de operadores aritméticos como *add*, *subtract*, *multiply*, *divide* e *module*, bem como um conjunto de operadores de comparação, como o *MATCHES*, incluindo o reconhecimento de muitos padrões usando expressões regulares.

Além disso, o *Pig Latin* também oferece suporte a UDF (funções definidas pelo usuário), que permite a invocação de componentes externos que implementam uma lógica especifica que pode ser difícil de modelar no *Pig Latin*. Contudo, é esperado que o desempenho geral do *Pig* seja afetado devido a camada de abstração frente ao *MapReduce*. Como dito, o *Pig* abstrai o nível de programação e executa em um *bash* próprio que acaba consumindo recursos.

## 4.3 Considerações finais

Neste capítulo apresentou-se a arquitetura proposta para a implementação de um IDS usando técnicas e ferramentas de *Big Data*. A ideia inicial era realizar uma pré-análise nos *logs* provenientes do PoP-RS com o auxílio da ferramenta *Logstash* e armazenar o resultado na base de dados *ElasticSearch*. Esses dados seriam então manipulados via *MapReduce* para se obter correlações entre eventos ocorridos na infraestrutura de rede. O resultado dessa nova análise seria, novamente, armazenado no *ElasticSearch* para ser visualizado com o auxílio da ferramenta *Kibana*.

Entretanto, na fase de desenvolvimento, detectou-se problemas no acoplamento do *ElasticSearch* com o *Hadoop* e isso forçou uma adaptação da arquitetura inicial. A solução final consistiu em fazer a análise dos dados diretamente da base de dados *Syslog-NG* e, com isso, perdeu-se a visualização gráfica com o *Kibana*.

Outra dificuldade encontrada foi a implementação de programas *MapReduce* devido a complexidade de adaptar e expressar a solução apenas com primitivas *map* e *reduce*. Para contornar esse problema, utilizou-se o *Pig*, que abstrai essas operações, mas que introduz uma camada a mais de *software*, onerando o desempenho final.

É importante ressaltar, ainda, que a arquitetura implementada é para ser usada em ambiente distribuído, mas como prova de conceito e restrições físicas (disponibilidade de máquinas) foi feita em uma só máquina. Por conta disso, também optou-se por *logs* de autenticação, pois são mais significativos para um IDS.

# 5 AVALIAÇÃO EXPERIMENTAL

A detecção de intrusão é feita através de informações presentes nos registro de auditoria. Os registros de auditoria são responsáveis por armazenar os eventos e ações feitas no sistema, em *logs*, os quais são lidos sequencialmente para se buscar a ocorrência de eventos/ações suspeitos. Essa busca por comportamento suspeito pode ser realizada de duas formas: (i) em tempo real (dados dinâmicos); (ii) em *postmortem* (dados estáticos). Neste trabalho é empregado o modelo de análise *postmortem*.

Tipicamente, são buscadas informações relevantes nos *logs*, como nome de usuário, que teve a conta invadida, ou informações referentes a um determinado servidor que foi comprometido, por exemplo. Há um interesse em classificar esses eventos pela data que ocorreram com o objetivo de listar os eventos mais antigos, ou mais novos. Por exemplo, buscando quando (a partir de qual data) um certo padrão de eventos no servidor invadido começou a aparecer e quanto tempo esse padrão se manteve. Não há um padrão único a ser pesquisado com precisão, a escolha do que buscar é dependente da experiência do administrador da rede e pode variar de caso para caso.

Uma forma tradicional dos administradores de rede para analisar os *logs* é via linha de comando em *shell script*. Nos sistemas operacionais, *Shell* refere-se a todo ambiente utilizado para interação do usuário com o computador (textual ou gráfico). Um *script* é um arquivo contendo um conjunto de comandos que serão executados por um interpretador. Em sistemas Unix, o *bash* (*Bourne Again Shell*) é um *Shell* em modo texto bastante utilizado. Seu interpretador *bash* define uma linguagem de *script* chamada *Bash Script*. Em administração de redes, sobretudo em ambientes Unix (Linux, AIX, FreeBSD, por exemplo), essa é uma das linguagens de *script* mais utilizadas (COSTA, 2010).

Normalmente, a pesquisa é feita em um ou mais campos do *log*, sendo realizada por meio do comando *grep* (*Global Regular Expression Print*) (ABOU-ASSALEH; AI, 2004). Além de buscas simples por uma determinada palavra, o comando *grep* faz uso de expressões regulares para comparação de *strings*. Esse tipo de comparação é a base da análise de dados de um Sistema de Detecção de Intrusão (VALGENTI et al., 2012). Contudo, a busca através do comando *grep* é lenta para um grande conjunto de dados (SINGER; LETHBRIDGE, 1997).

A avaliação experimental conduzida neste capítulo tem por objetivo comparar o tempo de execução da arquitetura proposta com o método convencional (utilizando o comando *grep*).

## 5.1 Metodologia

Para atingir o objetivo de comparar a forma tradicional de pesquisa (*grep*), contra o novo método, por meio de ferramenta de *Big Data* (*Pig*), é utilizado um conjunto de *logs* de autenticação para realizar uma análise *postmortem* de certos padrões de eventos, executando em uma plataforma de teste. Esta seção fornece detalhes da plataforma experimental, o critério de comparação, cenários de testes e *benchmarks* usados nesta comparação. Os experimentos são validados estatisticamente usando média e desvio padrão.

### 5.1.1 Plataforma experimental

A plataforma experimental para esta avaliação é um servidor virtualizado executando sobre uma máquina hospedeira HP ProLiant DL380 G5. Essa máquina hospedeira conta com a seguinte configuração de *hardware*: processador Intel Xeon E5430 2.66 GHz, memória 18430MB, 1TB de disco rígido usando conjunto redundante de discos independentes (*Redundant Array of Independent Disks*, RAID tipo 5) e conexão de rede *Ethernet* de 1Gbps. O servidor virtualizado possui 4 CPUs virtuais (2 *Threads* por CPU), memória RAM de 4GB e 150GB de disco da máquina hospedeira. O *Hypervisor* instalado na máquina hospedeira é o XenServer, versão 6.5 (XENSERVER, 2015).

O sistema operacional utilizado no servidor foi o Ubuntu versão 12.04.5 LTS. Esse servidor executa as ferramentas presentes na arquitetura final implementada (seção 4.2). Dentre outros serviços nativos do sistema operacional, as ferramentas usadas na arquitetura são: *Syslog-NG* versão 3.6.1, *Logstash* versão 1.5.0, *ElasticSearch* versão 1.5.1, *Kibana* versão 4.1, *Apache Hadoop* versão 2.7.0 e *Pig* versão 0.15.0.

Os serviços analisados por esta avaliação experimental são os relacionados a tecnologia *Big Data*, ou seja, os *softwares Hadoop* e *Pig*. Contudo, vale salientar que os serviços nativos do sistema operacional e os demais serviços da arquitetura, estão executando concorrentemente. Isto é, em razão dos recursos de *hardware* disponíveis, todos os serviços executam na mesma plataforma.

### 5.1.2 Critérios para análise

O desempenho é medido com relação ao tempo de execução para gerar o resultado pesquisado (tempo de resposta), representado no formato de minutos:segundos,milissegundos (MM:SS,mmm). Assim, compara-se o tempo de resposta do *bash*, executando o comando *grep*, frente ao *Pig*, executando um *script* em *Pig Latin* correspondente a mesma pesquisa. Para um conjunto de testes são variados o tamanho da entrada e a quantidade de campos pesquisados.

O desempenho também é investigado para o caso de realizar buscas em diferentes campos dos *logs*. A estrutura desses campos (Figura 4.2) corresponde ao *schema* de dados do *Pig*. A estrutura do *log*, contém os seguintes campos:

- *timestamp*: a marca temporal do evento, corresponde a uma *string* no formato ISO 8601.
- *host*: nome do hospedeiro envolvido no evento, corresponde a uma *string* (campo específico).
- *program*: o serviço que gerou o evento, também corresponde a uma *string* (campo específico).
- *message*: a descrição do evento enviada pelo hospedeiro, corresponde a *strings*.

A medição do tempo foi feita de duas formas. Para a análise tradicional, isso é, com *bash* foi usado o comando $time$. Para análise com *script Pig* foi utilizado o tempo que o *grunt* ("*bash*" do *Pig*) fica executando. Esse tempo é dado como saída pela ferramenta após a finalização do *script*. Em ambos os casos existe uma precisão de milissegundos.

### 5.1.3 Cenários de teste

Os cenários de teste empregados correspondem a pesquisas típicas por informações em *logs*. Na busca, caso ocorra a equivalência nos campos do *log*, o evento é mostrado. Em casos onde mais de uma informação é pesquisada, a busca retorna quando as duas informações estão presentes nos campos do *log*. De forma geral, o objetivo dos cenários é confrontar a busca com o comando *grep* (através do *bash*) e *Pig* (através de *script* em linguagem *Pig Latin*). Foram empregados três cenários de teste:

- Cenário I - busca em um campo: é realizado a busca em somente um campo do *log*. Há duas variações para essa busca: (i) pesquisar em um campo que corresponde a um valor único (campo específico), como o campo com o nome do hospedeiro (*host*); e, (ii)

pesquisar uma palavra em um campo contendo *strings*, como um nome de usuário no campo que descreve o evento (*message*).

- Cenário II - busca em N campos: é realizado a busca em dois ou mais campos do *log*. Há duas variações para essa busca: (i) pesquisar em dois campos, correspondente a uma data e a um usuário; e, (ii) pesquisar em três campos, correspondente a uma data, um hospedeiro e um usuário.
- Cenário III - busca com ordenação: após ordenar os *logs* de forma decrescente (eventos mais novos primeiro) é realizado a busca por uma data e um usuário. Assim, o comando $sort$ foi utilizando no *bash* para classificação em ordem crescente e a instrução *ORDER asc* no *Pig* para a mesma finalidade.

### 5.1.4 Conjunto de testes (*Benchmarks*)

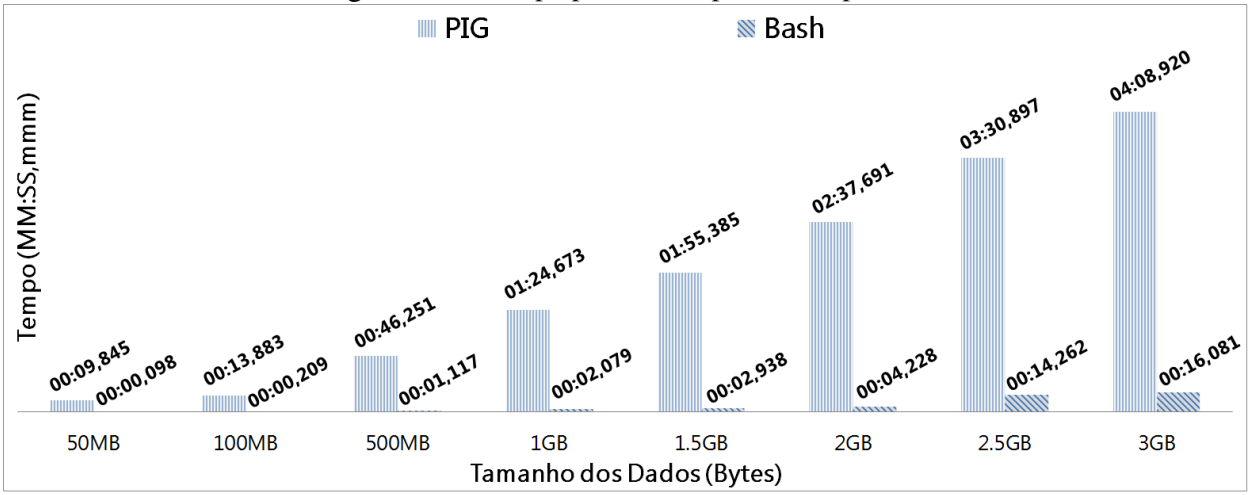
Foram utilizados como entrada para a análise experimental os *logs* reais de autenticação extraídos dos equipamentos pertencentes ao PoP-RS. Optou-se por *logs* de autenticação, pois são mais significativos para um IDS, já que nesses *logs* existe a informação de qual usuário conectou ao equipamento. O conjunto de dados usados correspondem a seis meses de coleta de dados, sendo, em média, 500 MB por mês de *log*.

## 5.2 Cenário I: Busca por um campo

No cenário I é realizado a medida do tempo, em milissegundos, para buscar por informações presentes em apenas um campo do *log*. Nesse cenário, há duas variações. Primeiro, a busca é feita em um campo específico, ou seja, busca por hospedeiro no campo *host*. Segundo, a busca é feita em um campo com *strings*, ou seja, busca por usuário no campo *message*. O objetivo da medida é comparar o desempenho entre instruções do *Pig* e o comando *grep* em *script bash* com intuito de verificar qual o mais rápido na busca por informações em um campo do *log*. Além disso, existe a curiosidade de descobrir se a busca em um campo específico tem melhor desempenho que a busca em um campo com *strings*.

A Figura 5.1 apresenta os tempos de execução para a busca em um campo específico do *log*, em função do tamanho do arquivo de *log*. Por sua vez, a Tabela 5.1 complementa essa informação fornecendo, além da média das amostras, o desvio padrão (DP). Para cada tamanho de *log*, foram realizadas 10 medidas (amostras).

Figura 5.1 – Tempo para buscar por um hospedeiro




(Fonte: Próprio autor)

Tabela 5.1 – Média e Desvio Padrão para busca em um campo específico

| Técnica | Medida | 50MB | 100MB | 500MB | 1GB | 1.5GB | 2GB | 2.5GB | 3GB |
|---|---|---|---|---|---|---|---|---|---|
| *Pig* | Média | 00:09,845 | 00:13,883 | 00:46,251 | 01:24,673 | 01:55,385 | 02:37,691 | 03:30,897 | 04:08,920 |
| | DP | 00:00,212 | 00:00,142 | 00:00,735 | 00:01,148 | 00:03,109 | 00:02,068 | 00:05,782 | 00:06,358 |
| *Bash* | Média | 00:00,098 | 00:00,209 | 00:01,117 | 00:02,079 | 00:02,938 | 00:04,228 | 00:14,262 | 00:16,081 |
| | DP | 00:00,009 | 00:00,014 | 00:00,025 | 00:00,038 | 00:00,099 | 00:00,536 | 00:02,628 | 00:02,238 |

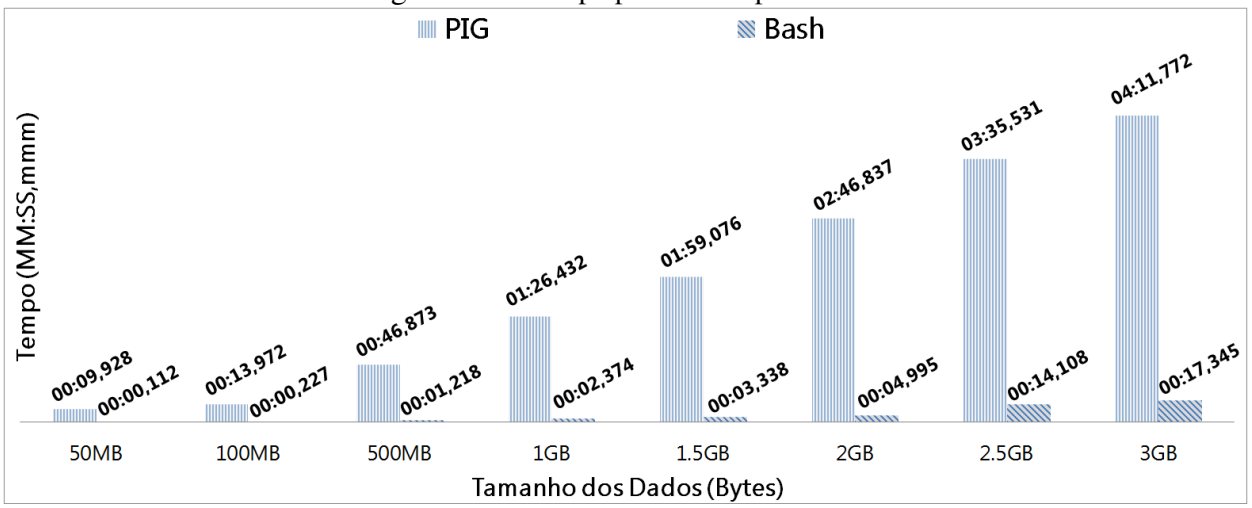
Em todos os casos, o *script* em *bash* tem um menor tempo de resposta se comparado com o *Pig*. Isso deve ao fato que existe um *overhead* para iniciar os componentes presentes na estrutura do *Hadoop*. Além disso, outra perda de desempenho refere-se ao *Pig* executar em uma camada acima do *Hadoop*, ou seja, em um nível maior de abstração (seção 4.2).

Um detalhe quanto ao tamanho dos *logs* está na transição de 2GB para 2.5GB, onde o tempo de execução mais que dobra na busca com *grep*. Uma possível explicação é que, em algum ponto intermediário nessa faixa de tamanho de dados, comece a ocorrer muitas falta de dados na memória (Falta de Página), obrigando o sistema operacional a acessar o sistema de arquivos (área de *swap*) ocasionando perda de desempenho.

Na segunda variação do cenário I, foi realizada a pesquisa nos *logs* com intuito de encontrar um usuário presente no campo *message*. Essa busca é feita em um campo do *schema* que contém *strings* ao usar o *Pig*.

A Figura 5.2 mostra os tempos de execução na busca em um campo com *strings*, em função do tamanho do arquivo de *log*. A Tabela 5.2 acrescenta, além da média, o desvio padrão (DP) das amostras. Novamente, para cada caso, foram realizadas 10 medições (amostas).

Figura 5.2 – Tempo para buscar por usuário




(Fonte: Próprio autor)

Tabela 5.2 – Média e Desvio Padrão para um campo com *strings*

| Técnica | Medida | 50MB | 100MB | 500MB | 1GB | 1.5GB | 2GB | 2.5GB | 3GB |
|---|---|---|---|---|---|---|---|---|---|
| *Pig* | Média | 00:09,928 | 00:13,972 | 00:46,873 | 01:26,432 | 01:59,076 | 02:46,837 | 03:35,531 | 04:11,772 |
| | DP | 00:00,266 | 00:00,084 | 00:00,736 | 00:01,394 | 00:01,857 | 00:05,663 | 00:05,809 | 00:04,950 |
| *Bash* | Média | 00:00,112 | 00:00,227 | 00:01,218 | 00:02,374 | 00:03,338 | 00:04,995 | 00:14,108 | 00:17,345 |
| | DP | 00:00,006 | 00:00,014 | 00:00,023 | 00:00,087 | 00:00,290 | 00:01,078 | 00:01,595 | 00:01,452 |

Os resultados são equivalentes aos da primeira variação, onde foi realizada a busca em um campo específico. A busca em *bash*, continua sendo mais eficiente, visto a existência de um *overhead* que não se paga para o tamanho dos dados na utilização do *Pig*. Além disso, os tempos de execução, de ambas variações do cenário I, utilizando a ferramenta *Pig* são praticamente os mesmos.

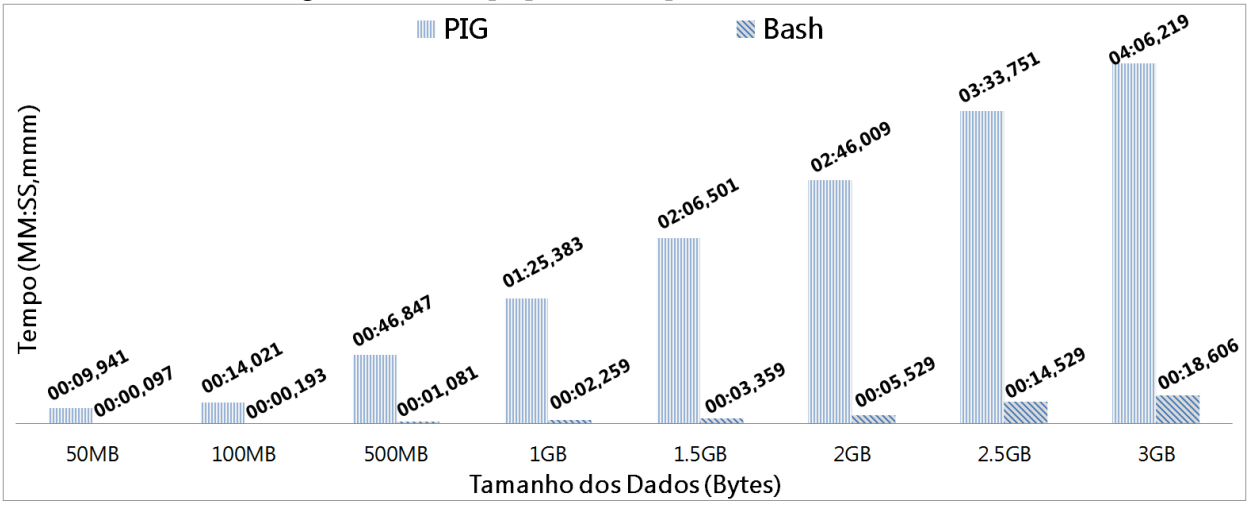
Contudo, em tamanhos de dados maiores, a busca em um campo específico, é levemente mais lenta (poucos segundos) que o da busca por *strings*. Uma provável causa para esse comportamento deve-se ao fato da busca por *strings* ser feita no campo *message* onde há mais de uma palavra a ser pesquisada. Consequentemente, é provável que mais operações de *map* e *reduce* são realizadas para encontrar o valor de usuário procurado.

## 5.3 Cenário II: Busca em N campos

No cenário II é realizado a medida do tempo, em milissegundos, para buscar por informações presentes em N campos do *log*. Inicialmente, a busca é feita em dois campos do *log*. Busca-se por uma data, no campo *timestamp*, e por um usuário no campo *message*. Na sequência, a busca é feita em três campos do *log*. São buscados uma data, no campo *timestamp*, um hospedeiro, no campo *host* e um usuário, no campo *message*. O objetivo dessa medida é comparar o desempenho entre instruções do *Pig* e comandos *grep* em *pipeline*, no *bash*, na busca por dois e três campos do *log*, com a intenção de verificar qual técnica é mais rápida. Além disso, há um interesse em descobrir qual operação tem o melhor desempenho: a busca em dois campos ou busca em três campos.

A Figura 5.3 mostra os tempos de execução obtidos na busca por dois campos (data e usuário), em função do tamanho do arquivo de *log*. Em seguida, a Tabela 5.3 complementa essa informação fornecendo, além da média das amostras (10), o desvio padrão (DP).

Figura 5.3 – Tempo para buscar por uma data e um usuário




(Fonte: Próprio autor)

Tabela 5.3 – Média e Desvio Padrão para dois campos

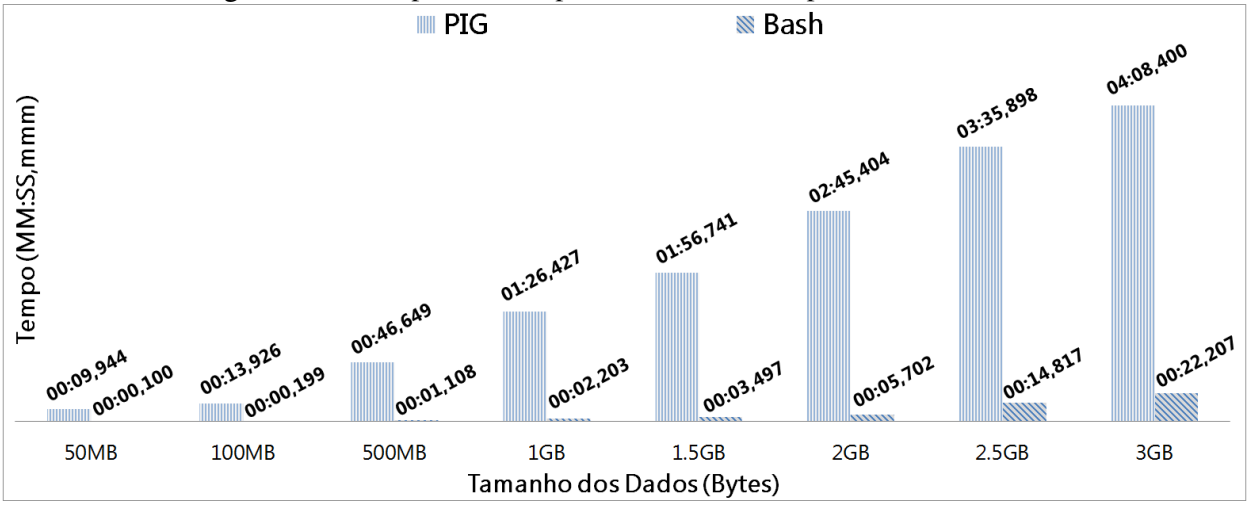
| Técnica | Medida | 50MB | 100MB | 500MB | 1GB | 1.5GB | 2GB | 2.5GB | 3GB |
|---|---|---|---|---|---|---|---|---|---|
| *Pig* | Média | 00:09,941 | 00:14,021 | 00:46,847 | 01:25,383 | 02:06,501 | 02:46,009 | 03:33,751 | 04:06,219 |
| | DP | 00:00,162 | 00:00,301 | 00:00,440 | 00:01,740 | 00:03,002 | 00:06,508 | 00:07,431 | 00:07,695 |
| *Bash* | Média | 00:00,097 | 00:00,193 | 00:01,081 | 00:02,259 | 00:03,359 | 00:05,529 | 00:14,529 | 00:18,606 |
| | DP | 00:00,009 | 00:00,020 | 00:00,023 | 00:00,089 | 00:00,657 | 00:01,156 | 00:01,641 | 00:01,671 |

Os resultados seguem o mesmo padrão do cenário I, provavelmente, pelo mesmo motivo, isso é, usando os comandos *grep* no *bash* o tempo de execução é menor pois com o *Pig* há uma arquitetura robusta e pesada que consome mais recursos (perda de desempenho). Assim sendo, o *pipeline* de *greps* é mais rápido em todos os casos da busca em dois campos.

Na sequência, com o objetivo de verificar o impacto no tempo de busca ao se incluir novos campos, analisou-se o caso em que foi realizada a pesquisa por três campos do *log*. Para isso, busca-se por um intervalo de tempo (campo *timestamp*), em uma máquina específica (campo *host*) acessada por um usuário (campo *message*) em particular.

A Figura 5.4 apresenta os tempos de execução, para cada tamanho de *log*, ao buscar informações em três campos (data, hospedeiro e usuário). Por sua vez, a Tabela 5.4 complementa essa informação com o desvio padrão (DP), além da média das amostras (20 medições por arquivo).

Figura 5.4 – Tempo de busca por uma data, um hospedeiro e um usuário




(Fonte: Próprio autor)

Tabela 5.4 – Média e Desvio Padrão para três campos

| Técnica | Medida | 50MB | 100MB | 500MB | 1GB | 1.5GB | 2GB | 2.5GB | 3GB |
|---|---|---|---|---|---|---|---|---|---|
| *Pig* | Média | 00:09,944 | 00:13,926 | 00:46,649 | 01:26,427 | 01:56,741 | 02:45,404 | 03:35,898 | 04:08,400 |
| | DP | 00:00,129 | 00:00,108 | 00:00,611 | 00:01,921 | 00:01,005 | 00:05,513 | 00:06,563 | 00:07,940 |
| *Bash* | Média | 00:00,100 | 00:00,199 | 00:01,108 | 00:02,203 | 00:03,497 | 00:05,702 | 00:14,817 | 00:22,207 |
| | DP | 00:00,012 | 00:00,013 | 00:00,049 | 00:00,091 | 00:01,353 | 00:01,438 | 00:02,079 | 00:05,798 |

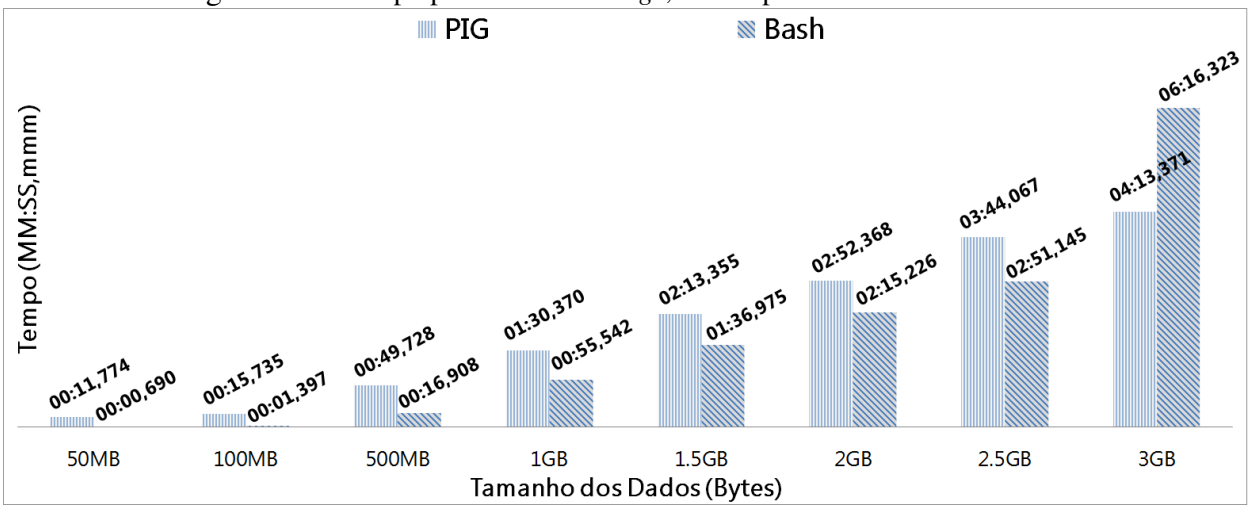
Os tempos de execução fornecidos na Figura 5.4 mostram que não há diferença significativa na busca por dois ou três campos. Além disso, também é notado que em comparação ao cenário I, onde é buscado apenas um campo, também não há diferença substancial, já que os resultados dos tempos de execução estão na mesma ordem de grandeza.

## 5.4 Cenário III: Busca com ordenação

Nesse cenário, é realizado a medida do tempo, em milissegundos, para classificar os *logs* em ordem decrescente pela data e buscar por informações presentes em dois campos do *log*. A busca é feita por uma data, no campo *timestamp*, e por um usuário no campo *message*. O objetivo dessa medida é verificar se a classificação dos *logs* tem impacto relevante no desempenho das técnicas utilizadas, antes de realizar as buscas. Assim, neste cenário foi adicionado uma operação de ordenação por *timestamp* antes das buscas por data e usuário. Como mencionado, para classificação em *bash* foi utilizado o comando $time$ e para o *Pig* foi utilizado a instrução *ORDER asc*.

A Figura 5.5 apresenta os tempos de execução, em função do tamanho do *log*, para ordenar e buscar em dois campos. Em seguida, a Tabela 5.5 complementa essa informação fornecendo, além da média, o desvio padrão (DP) das amostras (20 medidas por arquivo de *log*).

Figura 5.5 – Tempo para ordenar os *logs*, buscar por uma data e um usuário




(Fonte: Próprio autor)

Tabela 5.5 – Média e Desvio Padrão para busca com ordenamento

| Técnica | Medida | 50MB | 100MB | 500MB | 1GB | 1.5GB | 2GB | 2.5GB | 3GB |
|---|---|---|---|---|---|---|---|---|---|
| *Pig* | Média | 00:11,774 | 00:15,735 | 00:49,728 | 01:30,370 | 02:13,355 | 02:52,368 | 03:44,067 | 04:13,371 |
| | DP | 00:00,229 | 00:00,141 | 00:00,438 | 00:01,336 | 00:09,333 | 00:03,856 | 00:05,146 | 00:06,552 |
| *Bash* | Média | 00:00,690 | 00:01,397 | 00:16,908 | 00:55,542 | 01:36,975 | 02:15,226 | 02:51,145 | 06:16,323 |
| | DP | 00:00,022 | 00:00,021 | 00:01,447 | 00:06,466 | 00:09,382 | 00:10,665 | 00:11,851 | 00:11,838 |

Os resultados presentes na Figura 5.5 mostram que ordenar os dados no *bash* tem um impacto significativo no tempo total da busca. Além do tempo de execução ser mais lento do que os casos correspondentes nos cenários anteriores, no último caso (3GB de dados analisados) o *script* em *bash* é, inclusive, mais lento do que o *script* em *Pig*. Provavelmente, o comando $sort$ não seja tão otimizado (algoritmo mais robusto) quanto a instrução *ORDER asc* do *Pig*. Por outro lado, na escala de tempo com o *Pig*, os tempos não cresceram na mesma proporção vista em *bash*, ou seja, com o *Pig* o aumento de tempo não é representativo (não foi muito significativo).

Isso pode ser analisado frente a duas posições. Primeira, esse comportamento dos testes com busca para o *Pig*, indica que o predominante é o tempo de inicialização do arquitetura do *Pig*, já que, em todos os cenários o tempo de execução foi sempre na mesma ordem de grandeza. Segunda, isso reflete que as operações de *MapReduce* tem um melhor desempenho (especializadas) para ordenação frente as operações em *bash*. Há até um operador exclusivo (*ORDER*) para ordenamento no *Pig* (seção 4.2) que deve refletir no mapeamento direto para fase de classificação do *MapReduce*. Assim, para operações com processamento de ordenação, o *Pig* tem um melhor desempenho do que o *bash* para dados com volume de 3GB.

## 5.5 Considerações finais

O *Pig* tem como desvantagem o gasto de processamento para iniciar a estrutura do *Hadoop* (executar as operações de *MapReduce*) ao invés de se preocupar em apenas coletar as informações pesquisadas. Além disso, há um gasto de desempenho no que diz respeito a gerencia dos recursos, ou seja, existe um *overhead* na distribuição das funções de mapeamento e redução. Assim, nos casos analisados, para realização de buscas em *logs* com tamanho total de até 3GB não é aconselhado o uso do *Pig* visto que realizar a busca com um simples *grep* é mais vantajoso por ter maior desempenho.

Contudo, caso ocorra uma operação de classificação, o *Pig* leva vantagem frente ao *bash* para uma quantidade de dados significativa (entre 2.5GB e 3GB). Com relação ao número de campos pesquisados, não houve diferença significativa nos tempos de execução nos cenários escolhidos. Já o volume dos dados tem um impacto grande, ou seja, quanto mais dados, maior o tempo de execução. Além de que, nos casos onde há um possível aumento no número de falta de página, há uma perda de desempenho que pode ser resolvida aumentando a quantidade de memória RAM do servidor.

Com relação ao estudo de viabilidade (seção 3.4) para a solução proposta implementada, após a análise de desempenho feita neste capítulo, é possível decidir se vale a pena ou não gastar tempo e esforço com sistema proposto. Assim, verifica-se os três aspectos referentes a contribuição do sistema, se pode ser implementado e se pode ser integrado:

- Contribuição: O sistema contribui de duas formas. Primeiro, foi possível construir um sistema para analisar os *logs* por meio de ferramentas *Big Data*, contribuindo para futuras investigações desse gênero no PoP-RS. Segundo, foi implementado uma análise manual, por inspeção visual, através de uma pilha de *softwares*, denominada pilha ELK, que contribui para visualização gráfica dos eventos de autenticação ocorridos na infraestrutura do PoP-RS.
- Implementação: O sistema foi implementado para um caso de prova de conceito, onde foi utilizado apenas *logs* da *facility* de autenticação.
- Integração: O sistema foi integrado ao ambiente de produção do PoP-RS, sem interferir no seu funcionamento. Foi feita a transferência dos eventos da infraestrutura do PoP-RS para a solução proposta.

Dessa forma, é aconselhado a continuação do desenvolvimento do sistema proposto. Apesar do mau desempenho da ferramenta *Big Data* nos cenários I e II, no cenário III, onde foi feita a classificação por data, a ferramenta se mostrou melhor que a análise tradicional para o caso de 3GB de dados. Além disso, o crescente aumento no tamanho dos *logs*, também é um indicativo que tecnologias *Big Data* devem ser mais exploradas. Uma das possibilidades está na procura de correlação de eventos nas demais *facilities*, ou seja, maior quantidade de *logs*.

# 6 CONCLUSÃO

No decorrer deste trabalho foram contextualizados os conceitos e tecnologias referentes a IDS e *Big Data*. Inicialmente, foi visto que o IDS é a segunda frente de defesa contra ataques em uma infraestrutura e uma forma do IDS detectar intrusos é, tipicamente, por meio de análise de *logs*. Essa análise é feita buscando padrões específicos nos *logs*. Esses *logs*, para uma infraestrutura com diversos serviços e muitos equipamentos, tendem a se tornarem grandes. Assim, técnicas de *Big Data* podem ser interessantes para efetuar sua análise.

O objetivo deste trabalho foi estudar a viabilidade de um IDS utilizando técnicas de *Big Data*. A arquitetura proposta foi implementada e integrada facilmente a infraestrutura de rede existente, sem interferir no ambiente de produção do PoP-RS. O sistema dá suporte a uma contribuição inicial para a análise de *logs* por meio de ferramentas *Big Data* para o PoP-RS. Levando em conta que pesquisas por padrões em *logs* é uma forma básica utilizada por um IDS, foi comparado o desempenho de buscar informações em *logs* através da forma tradicional (comandos *grep*) contra uma ferramenta *Big Data* (*script Pig*). Como dados de entrada, foram utilizados os *logs* reais do PoP-RS filtrados na *facility* de autenticação. Nesses *logs* há informações referentes ao acesso de usuários, ou serviços, aos equipamentos da rede, sendo mais interessantes para uma prova de conceito de um IDS. Não foram analisados outros tipos de *logs* em parte devido a dificuldade em descrevê-los utilizando expressões regulares na ferramenta *Pig*.

Durante a fase de implementação da proposta, problemas associados a conexão entre *Pig* e *ElasticSearch* impossibilitaram utilizar a pré-análise do *Logstash*, forçando a análise ser feita diretamente na base de dados do módulo *Syslog-NG*. Com relação ao desempenho da arquitetura implementada, quando analisado os resultados da avaliação experimental, constatou-se que o desempenho da ferramenta *Pig* não foi adequado frente a formas tradicionais de busca por padrões em *logs* (comando *grep*). Tanto em busca por um campo específico ou com *strings*, quanto em busca por dois e três campos, o comando *grep* executou mais rápido que o *Pig*. É necessário executar testes mais complexos, com mais campos, com *log* de diferentes *facilities*, para verificar se esses fatores amortizam custos. Esse fraco desempenho ocorre pois o custo de executar toda arquitetura do *Hadoop*, que está em um nível abaixo do *Pig*, não é compensado. Entretanto, para o caso de operações de ordenamento, e o tamanho dos *logs* for entre 2.5GB e 3GB, o uso do *Pig* começa a ter um melhor desempenho. Uma provável causa deve-se ao fato de existir a fase de classificação no *MapReduce*. Contudo, vale salientar que, como prova de conceito, o *Pig* foi executado em apenas uma máquina, o que, certamente, não é o mais

adequado para um sistema que foi concebido para ser executado em um ambiente distribuído.

As contribuições deste trabalho estão na forma de experiência e aprendizado através do estudo e uso das ferramentas de *Big Data*, tais como *Hadoop*, *Pig*, *ElasticSearch*, *Kibana* e *Logstash*, por exemplo. Além disso, um sistema para análise de *logs* de autenticação foi implementado na infraestrutura do PoP-RS (Pilha ELK) auxiliando a análise manual dos *logs* por inspeção visual. É possível expandir esse sistema para analisar outros tipos de *logs* (outras *facilities*), bastando apenas acrescentar novos filtros no *Logstash* e criar novos gráficos no *Kibana*.

Como trabalhos futuros, há certos pontos a serem melhor investigados. Foi notado, ao final do trabalho, a viabilidade para utilização de um outro caminho para as análises na arquitetura final proposta implementada. A partir do módulo *Pig*, os dados passariam para o módulo *Logstash* que encaminharia para o banco de dados *ElasticSearch* e, dessa forma, as análises seriam visualizadas no *Kibana*. Outro ponto interessante seria utilizar uma infraestrutura distribuída para designar as operações de *map* e *reduce*, ou seja, usar o *Pig* de forma distribuída, invés de local, em mais de uma máquina (uso do *Hadoop YARN*). Assim, poderia ser verificado, por exemplo, se os tempos envolvidos em operações de gerencia de distribuição das operações é compensado por ter mais poder de processamento. Outro aspecto relevante seria analisar o impacto em utilizar um *framework* de tempo real como o *Apache Spark* ou *Apache Storm*, ao invés do *Apache Hadoop* que é *postmortem*. Por fim, outra questão pertinente seria averiguar *logs* com maior volume de informação e estudar os seus campos para descrevê-los através de expressões regulares e, consequentemente, realizar correlação com mais de uma *facility*, ou seja, utilizar outros tipos de *log* (não só de autenticação). Tipicamente, em infraestruturas com muitos servidores *web* existe a geração de um alto volume de *logs* de acesso, por exemplo.

Dessa forma, como orientação para os trabalhos futuros, é indicado o uso do *Pig* para análise de *logs* desde o início da implementação invés de *MapReduce*. Por usar uma linguagem *script* procedural orientada a fluxo de informações, o *Pig* facilita no momento de pesquisar informações nos *logs* sem ter a necessidade de produzir resultados através do paradigma *MapReduce*. Isto é, programar a lógica do processamento em apenas duas primitivas (mapeamento e redução) é uma atividade, por vezes, nada fácil.

# REFERÊNCIAS

ABITEBOUL, S. **Querying semi-structured data**. [S.l.]: Springer, 1997.

ABOU-ASSALEH, T.; AI, W. **Survey of global regular expression print (grep) tools**. [S.l.], 2004.

ADACHI, T. Gestão de segurança em internet banking: estudo de casos brasileiros. 2004.

ADRIAANS, P.; ZANTINGE, D. Data mining. harlow. **England: Addison Wesley**, 1996.

AGRAWAL, D. et al. Challenges and opportunities with big data. **A community white paper developed by leading researchers across the United States**, 2012.

APACHE. 2015. Disponível em: <http://www.apache.org/>. Acesso em: 15 out. 2015.

APPLEYARD, R.; ADAMS, J. Using the elk stack for castor application logging at ral. In: **International Symposium on Grids and Clouds (ISGC)**. [S.l.: s.n.], 2015. v. 15, n. 20.

AVRO. 2015. Disponível em: <http://avro.apache.org/>. Acesso em: 15 out. 2015.

AXELSSON, S. **Intrusion detection systems: A survey and taxonomy**. [S.l.], 2000.

AZURE Table Storage. 2015. Disponível em: <http://msdn.microsoft.com/en-us/library/dd179423.aspx>. Acesso em: 19 out. 2015.

BACE, R.; MELL, P. **NIST special publication on intrusion detection systems**. [S.l.], 2001.

BAGNASCO, S. et al. Monitoring of iaas and scientific applications on the cloud using the elasticsearch ecosystem. In: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2015. v. 608, n. 1, p. 012016.

BAI, J. Feasibility analysis of big log data real time search based on hbase and elasticsearch. In: IEEE. **Natural Computation (ICNC), 2013 Ninth International Conference on**. [S.l.], 2013. p. 1166–1170.

BAJPAYEE, R.; SINHA, S. P.; KUMAR, V. Big data: A brief investigation on nosql databases. 2015.

BARROS, A. et al. Correlation patterns in service-oriented architectures. In: **Fundamental Approaches to Software Engineering**. [S.l.]: Springer, 2007. p. 245–259.

BEAL, A. **Segurança da informação: princípios e melhores práticas para a proteção dos ativos de informação nas organizações**. [S.l.]: Atlas, 2005.

BEYER, M. A.; LANEY, D. The importance of 'big data': a definition. **Stamford, CT: Gartner**, p. 1–9, 2012.

BISHOP, M. **Introduction to computer security**. [S.l.]: Addison-Wesley Boston, MA, 2005.

BORTHAKUR, D. et al. Apache hadoop goes realtime at facebook. In: ACM. **Proceedings of the 2011 ACM SIGMOD International Conference on Management of data**. [S.l.], 2011. p. 1071–1080.

BRAY, R.; CID, D.; HAY, A. **OSSEC host-based intrusion detection guide**. [S.l.]: Syngress, 2008.

CAMPELLO, R. S.; WEBER, R. F. Sistemas de detecçao de intrusao. **Minicurso procedente do 19º Simpósio Brasileiro de Redes de Computadores**, 2001.

CAMPOS, A. N. **Sistema de segurança da informação - Controlando os Riscos**. [S.l.]: Florianópolis: Visual Books, 2014.

CARASSO, D. Exploring splunk. **published by CITO Research, New York, USA, ISBN**, p. 978–0, 2012.

CASSANDRA. 2015. Disponível em: <https://cassandra.apache.org/>. Acesso em: 19 out. 2015.

CENTRO DE ESTUDOS, RESPOSTAS E TRATAMENTO DE INCIDENTES DE SEGURANçA NO BRASIL. 2015. Disponível em: <http://cartilha.cert.br/mecanismos/>. Acesso em: 20 out. 2015.

CHANG, F. et al. Bigtable: A distributed storage system for structured data. **ACM Transactions on Computer Systems (TOCS)**, ACM, v. 26, n. 2, p. 4, 2008.

CHUNG, L. et al. **Non-functional requirements in software engineering**. [S.l.]: Springer Science & Business Media, 2012. v. 5.

CLEMENT, A. Nsa surveillance: Exploring the geographies of internet interception. **iConference 2014 Proceedings**, iSchools, 2014.

COLELLA, P. Defining software requirements for scientific computing. presentation, 2004.

COSTA, D. G. **Administração de redes com scripts: Bash Script, Python e VBScript**. [S.l.]: Brasport, 2010.

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. **Communications of the ACM**, ACM, v. 51, n. 1, p. 107–113, 2008.

DENNING, D. E. An intrusion-detection model. **Software Engineering, IEEE Transactions on**, IEEE, n. 2, p. 222–232, 1987.

DIJCKS, J. P. Oracle: Big data for the enterprise. **Oracle White Paper**, 2012.

DYNAMODB. 2015. Disponível em: <http://aws.amazon.com/dynamodb/>. Acesso em: 19 out. 2015.

EATON, C. et al. **Understanding big data**. [S.l.]: McGraw-Hill New York, 2012.

ELASTIC. 2015. Disponível em: <https://www.elastic.co/>. Acesso em: 15 out. 2015.

ELASTICSEARCH. 2015. Disponível em: <http://www.elasticsearch.org/>. Acesso em: 19 out. 2015.

ES-HADOOP. 2015. Disponível em: <https://www.elastic.co/products/hadoop>. Acesso em: 15 out. 2015.

ES-HADOOP 2.1.1. 2015. Disponível em: <https://www.elastic.co/downloads/past-releases/elasticsearch-apache-hadoop-2-1-1>. Acesso em: 15 out. 2015.

FILHO, W. de P. P. **Engenharia de software**. [S.l.]: LTC, 2003. v. 2.

FISHER, D. et al. Interactions with big data analytics, interactions, v. 19 n. 3. **May+ June**, 2012.

FLUENTD. 2015. Disponível em: <http://www.fluentd.org/>. Acesso em: 15 out. 2015.

FLUME. 2015. Disponível em: <https://flume.apache.org/>. Acesso em: 15 out. 2015.

GANTZ, J.; REINSEL, D. Extracting value from chaos. **IDC iview**, n. 1142, p. 9–10, 2011.

GERHARDS, R. et al. Rfc 5424: The syslog protocol. **Request for Comments, IETF**, 2009.

GOODRICH, M. T.; TAMASSIA, R. **Introdução à segurança de computadores**. [S.l.]: Bookman, 2013.

GORMLEY, C.; TONG, Z. **Elasticsearch: The Definitive Guide**. [S.l.]: "O'Reilly Media, Inc.", 2015.

HADOOP and Friends. [S.l.], 2015. Disponível em: <https://www.elastic.co/videos/elasticsearch-hadoop-and-friends/?view=1>. Acesso em: 14 out. 2015.

HADOOP Document. 2015. Disponível em: <http://www.hadoop.apache.org/>. Acesso em: 19 out. 2015.

HAN, J. et al. Survey on nosql database. In: IEEE. **Pervasive computing and applications (ICPCA), 2011 6th international conference on**. [S.l.], 2011. p. 363–366.

HBASE. 2015. Disponível em: <http://hbase.apache.org/>. Acesso em: 15 out. 2015.

HE, C. Survey on nosql database technology. **Journal of Applied Science and Engineering Innovation Vol**, v. 2, n. 2, 2015.

HIVE. 2015. Disponível em: <https://hive.apache.org/>. Acesso em: 15 out. 2015.

HU, H. et al. Toward scalable systems for big data analytics: A technology tutorial. **Access, IEEE**, IEEE, v. 2, p. 652–687, 2014.

INFOGRID. 2015. Disponível em: <http://infogrid.org/trac/>. Acesso em: 19 out. 2015.

INTERNET ENGINEERING TASK FORCE. **Internet Security Glossary**. [S.l.], 2000. Disponível em: <https://www.ietf.org/rfc/rfc2828.txt>.

KHAN, M. Ali-ud-din; UDDIN, M. F.; GUPTA, N. Seven v's of big data understanding big data to extract value. In: IEEE. **American Society for Engineering Education (ASEE Zone 1), 2014 Zone 1 Conference of the**. [S.l.], 2014. p. 1–5.

KIBANA. 2015. Disponível em: <https://www.elastic.co/products/kibana>. Acesso em: 15 out. 2015.

KRUEGEL, C.; VALEUR, F.; VIGNA, G. **Intrusion detection and correlation: challenges and solutions**. [S.l.]: Springer Science & Business Media, 2005. v. 14.

LAHMADI, A.; BECK, F. Powering monitoring analytics with elk stack. In: **9th International Conference on Autonomous Infrastructure, Management and Security (AIMS 2015)**. [S.l.: s.n.], 2015.

LANEY, D. 3-d data management: Controlling data volume. **Velocity and Variety, META Group Original Research Note**, 2001.

LANGNER, R. Stuxnet: Dissecting a cyberwarfare weapon. **Security & Privacy, IEEE**, IEEE, v. 9, n. 3, p. 49–51, 2011.

LOGSTASH. 2015. Disponível em: <https://www.elastic.co/products/logstash>. Acesso em: 15 out. 2015.

LOHR, S. The age of big data. **New York Times**, v. 11, 2012.

LONVICK, C. Rfc 3164: The bsd syslog protocol. **Network Working Group, May**, v. 9, 2001.

LÓSCIO, B. F.; OLIVEIRA, H. R. d.; PONTES, J. C. d. S. Nosql no desenvolvimento de aplicações web colaborativas. **VIII Simpósio Brasileiro de Sistemas Colaborativos**, 2011.

LOSHIN, D. **Big data analytics: from strategic planning to enterprise integration with tools, techniques, NoSQL, and graph**. [S.l.]: Elsevier, 2013.

LUCENE, A. **Apache lucene**. 2015. Disponível em: <https://lucene.apache.org/>. Acesso em: 12 nov. 2015.

LYNCH, C. Big data: How do your data grow? **Nature**, Nature Publishing Group, v. 455, n. 7209, p. 28–29, 2008.

MARIANI, L.; PASTORE, F. Automated identification of failure causes in system logs. In: IEEE. **Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on**. [S.l.], 2008. p. 117–126.

MARINO, E. D. Kibana dashboards for grid services. 2015.

MAURO, A. D.; GRECO, M.; GRIMALDI, M. What is big data? a consensual definition and a review of key research topics. In: **AIP Conference Proceedings**. [S.l.: s.n.], 2015. v. 1644, p. 97–104.

MEIJER, E. The world according to linq. **Queue**, ACM, v. 9, n. 8, p. 60, 2011.

MONGODB. 2015. Disponível em: <http://www.mongodb.org/>. Acesso em: 19 out. 2015.

MONIRUZZAMAN, A.; HOSSAIN, S. A. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. **International Journal of Database Theory & Application**, v. 6, n. 4, 2013.

MONTEITH, J. Y.; MCGREGOR, J. D.; INGRAM, J. E. Hadoop and its evolving ecosystem. In: CITESEER. **IWSECO@ ICSOB**. [S.l.], 2013. p. 57–68.

MOTAHARI-NEZHAD, H. R. et al. Event correlation for process discovery from web service interaction logs. **The VLDB Journal—The International Journal on Very Large Data Bases**, Springer-Verlag New York, Inc., v. 20, n. 3, p. 417–444, 2011.

MUKHOPADHYAY, I. et al. A comparative study of related technologies of intrusion detection & prevention systems. **Journal of Information Security**, Scientific Research Publishing, v. 2, n. 01, p. 28, 2011.

NEO4J. 2015. Disponível em: <http://neo4j.com/>. Acesso em: 19 out. 2015.

NIST BIG DATA WORKING GROUP (NBD-WG). **Big data: The next frontier for innovation, competition, and productivity**. 2011. Disponível em: <https:http://bigdatawg.nist.gov/MGI_big_data_full_report.pdf>. Acesso em: 18 out. 2015.

NOSQL. **List of NoSQL databases**. 2015. Disponível em: <http://nosql-database.org/>. Acesso em: 19 out. 2015.

ORTOZA. **Beyond Web Application Log Analysis using Apache Hadoop**. 2014. Disponível em: <http://www.orzota.com/wp-content/uploads/2014/04/loganalysis-paper.pdf>. Acesso em: 18 out. 2015.

PACKETPIG. **PacketPig - Open Source Big Data Security Analytics**. 2015. Disponível em: <https://github.com/packetloop/packetpig>. Acesso em: 19 out. 2015.

PFLEEGER, S. L. Engenharia de software: teoria e prática. **2ª Edição, Prentice Hall**, 2004.

PIG. 2015. Disponível em: <https://pig.apache.org/>. Acesso em: 15 out. 2015.

PIGGBANK. 2015. Disponível em: <https://pig.apache.org/docs/r0.7.0/api/org/apache/pig/piggybank/storage/apachelog/package-summary.html>. Acesso em: 24 nov. 2015.

POP-RS/RNP. **Ponto de Presença da Rede Nacional de Ensino e Pesquisa no Estado do Rio Grande do Sul**. 2015. Disponível em: <http://www.pop-rs.rnp.br/>. Acesso em: 20 out. 2015.

PORRAS, P. Stat–a state transition analysis tool for intrusion detection. University of California at Santa Barbara, 1993.

PORRAS, P. et al. The common intrusion detection framework architecture. **CIDF working group, http://www. gidos. org/drafts/architecture. txt,(accessed: 5 September, 2001)**, 1998.

PRESSMAN, R. S. **Engenharia de software: Uma abordagem profissional**. 7th. ed. [S.l.]: McGraw Hill Brasil, 2011.

REGUIEG, H. et al. Using mapreduce to scale events correlation discovery for business processes mining. In: **Business Process Management**. [S.l.]: Springer, 2012. p. 279–284.

ROESCH, M. et al. Snort: Lightweight intrusion detection for networks. In: **LISA**. [S.l.: s.n.], 1999. v. 99, n. 1, p. 229–238.

RSA-PIVOTAL. **RSA-Pivotal Security Big Data Reference Architecture**. 2014. Disponível em: <http://www.emc.com/collateral/white-paper/h12878-rsa-pivotal-security-big-data-reference-architecture-wp.pdf>. Acesso em: 19 out. 2015.

RUSSELL, D.; GANGEMI, G. **Computer Security Basics**. [S.l.]: "O'Reilly Media, Inc.", 1991.

SANTOS, M. T. et al. **Gerência de redes de computadores**. [S.l.]: Escola Superior de Redes, 2015.

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

SCHEIDLER, B. **syslog-ng reference manual**. 2000.

SCHNEIER, B.; VIEIRA, D. **Segurança. com: segredos e mentiras sobre a proteção na vida digital**. [S.l.]: Campus, 2001.

SCHROECK, M. et al. Analytics: The real world use of big data. ibm institute for business value—executive report. **IBM Institute for Business Value**, 2012.

SÊMOLA, M. et al. **Gestão da segurança da informação**. [S.l.]: Elsevier Brasil, 2003. v. 1.

SINGER, J.; LETHBRIDGE, T. What's so great about 'grep'? implications for program comprehension tools. **WWW: http://wwwsel. iit. nrc. ca/ singer/grep/greptxt. html**, Citeseer, 1997.

SOMMERVILLE, I. **Engenharia de software**. 8th. ed. [S.l.]: São Paulo: Pearson Addison Wesley, 2007.

SOMMERVILLE, I.; SAWYER, P. **Requirements engineering: a good practice guide**. [S.l.]: John Wiley & Sons, Inc., 1997.

SPARK. 2015. Disponível em: <http://spark.apache.org/>. Acesso em: 15 out. 2015.

SPLUNK. 2015. Disponível em: <http://www.splunk.com/en_us/homepage.html>. Acesso em: 19 out. 2015.

STALLINGS, W.; VIEIRA, D. **Criptografia e segurança de redes: princípios e práticas**. [S.l.]: Pearson Prentice Hall, 2008.

STORM. 2015. Disponível em: <https://storm.apache.org/>. Acesso em: 15 out. 2015.

TAYLOR, R. C. An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. **BMC bioinformatics**, BioMed Central Ltd, v. 11, n. Suppl 12, p. S1, 2010.

THERDPHAPIYANAK, J.; PIROMSOPA, K. Applying hadoop for log analysis toward distributed ids. In: ACM. **Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication**. [S.l.], 2013. p. 3.

TURNBULL, J. The logstash book. **Kindle Ed**, 2014.

USM. 2015. Disponível em: <https://www.alienvault.com/solutions/intrusion-detection-system>. Acesso em: 15 out. 2015.

VAARANDI, R.; NIZIŃSKI, P. Comparative analysis of open-source log management solutions for security monitoring and network forensics. In: **Proceedings of the 2013 European Conference on Information Warfare and Security**. [S.l.: s.n.], 2013. p. 278–287.

VAARANDI, R.; PIHELGAS, M. Using security logs for collecting and reporting technical security metrics. In: IEEE. **Military Communications Conference (MILCOM), 2014 IEEE**. [S.l.], 2014. p. 294–299.

VALGENTI, V. C. et al. Gpp-grep: high-speed regular expression processing engine on general purpose processors. In: **Research in Attacks, Intrusions, and Defenses**. [S.l.]: Springer, 2012. p. 334–353.

VIEIRA, M. R. et al. Bancos de dados nosql: conceitos, ferramentas, linguagens e estudos de casos no contexto de big data. **Simpósio Brasileiro de Bancos de Dados**, 2012.

WARD, J. S.; BARKER, A. Undefined by data: a survey of big data definitions. **arXiv preprint arXiv:1309.5821**, 2013.

WINKLER, I. S.; DEALY, B. Information security technology? don't rely on it. a case study in social engineering. In: **USENIX Security**. [S.l.: s.n.], 1995.

XENSERVER. 2015. Disponível em: <http://xenserver.org/>. Acesso em: 27 nov. 2015.

ZASLAVSKY, A.; PERERA, C.; GEORGAKOPOULOS, D. Sensing as a service and big data. **arXiv preprint arXiv:1301.0159**, 2013.

ZIKOPOULOS, P.; EATON, C. et al. **Understanding big data: Analytics for enterprise class hadoop and streaming data**. [S.l.]: McGraw-Hill Osborne Media, 2011.

ZOOKEEPER. 2015. Disponível em: <https://zookeeper.apache.org/>. Acesso em: 15 out. 2015.

# ANEXO A — TRABALHO DE GRADUAÇÃO - I

- Título: Detecção de Intrusão por meio de Correlação de Logs através de Big Data
- Data: Junho de 2015

# Detecção de Intrusão por meio de Correlação de *Logs* através de *Big Data*

**William R. de C. Vidal, Alexandre S. Carissimi**

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{wrcvidal,asc}@inf.ufrgs.br

***Abstract.*** *This paper presents an architecture of an IDS that uses Big Data techniques for log correlation. IDS is a very important element in the network infrastructure to detect intruders. This work contributes with a study of IDS concept, classifying them and also explains the essentials aspects involving Big Data.*

***Resumo.*** *Este trabalho apresenta uma arquitetura de um IDS que faz uso das técnicas de Big Data para correlacionar logs. IDS é um elemento de suma importância em uma infraestrutura de rede para detecção de invasores. O trabalho contribui com um estudo dos conceitos de IDS classificando-os e também esclarecendo os aspectos essenciais que envolvem Big Data.*

## 1. Introdução

"A intrusão não autorizada em um sistema ou rede de computadores é uma das ameaças mais sérias à segurança de computadores" [Stallings, 2008]. Para um sistema de rede conectado à Internet é muito relevante à implementação de técnicas como *firewalls* e zonas desmilitarizadas (*demilitarized zone* - DMZ) que formam a primeira frente de ação para ajudar os administradores de rede a prover segurança de seus serviços. Contudo, o melhor sistema de segurança inevitavelmente falhará [Stallings, 2008].

Dessa forma, uma segunda frente de ação em um sistema de segurança é descobrir quais ataques e quando foram realizados. Um Sistema de Detecção de Intrusão (*Intrusion Detection Systems* - IDS) é uma tentativa de detectar ataques através de monitoramento dos sistemas de rede ou a partir do comportamento da infraestrutura. Em redes em que há uma massiva transferência de informações, como em *backbones*, a detecção de intrusão se torna um problema ainda maior devido à alta variabilidade de dados e a velocidade dos enlaces.

Por exemplo, no âmbito do Ponto de Presença da RNP no estado do Rio Grande do Sul [PoP-RS/RNP, 2015] é prestado serviços de conectividade para mais de 100 instituições. Em sua infraestrutura, há mais de 30 servidores e diversos equipamentos de rede, tais como *switches* e roteadores de grande a pequeno porte. Além disso, são implementados diversos serviços como DNS, e-mail e NTP, por exemplo. Por essa razão, é importante a detecção, não tardia, de uma invasão a essa infraestrutura. A totalização dos *logs* relacionados aos serviços de segurança no PoP-RS supera o tamanho de 1GB por dia. Os *logs* nada mais são que "diários de bordo" de serviços e equipamentos e também são chamados de registros de auditoria [Stallings, 2008]. É neles que se encontra o histórico de eventos dos sistemas e que, em última análise, ajuda na detecção de intrusos.

O problema é que, para uma análise humana nesse cenário, é extremamente difícil visualizar a correlação de informações que possibilitem uma detecção de invasão ou falha de segurança [Vernekar e Buchade, 2013]. O tamanho dos arquivos de *logs*, e também o fato que cada serviço possui um formato próprio para os *logs*, torna impraticável uma análise manual [Mariani e Pastore, 2008]. Entretanto, é possível, de forma automatizada, uniformizar e correlacionar os logs e, por exemplo, inferir que uma tentativa de intrusão está acontecendo [Abad et al., 2003].

Há uma grande gama de soluções de IDS para o setor de segurança, onde se destacam, entre outros, o Snort [Roesch, 1999] e Ossec [Bray et al., 2008]. Inclusive, há soluções comerciais para análise de grandes quantidades de dados [Splunk, 2015; Ortoza, 2014]. Contudo, o interesse deste trabalho está em utilizar *software* livre na construção de um IDS. Assim, Este trabalho propõe uma forma de analisar *logs* de diferentes fontes e de grande tamanho, utilizando técnicas de *Big Data* [Lynch, 2008]. A principal vantagem do sistema proposto é dispensar a análise manual/humana nos *logs* para gerar alertas.

O artigo está organizado de forma que reflita um estudo na área de IDS (seção 2), onde é apresentada a motivação para um IDS, seus métodos de detecção e sua classificação quanto à localização. Após, na seção 3, é discutido os conceitos de *Big Data*, suas formas de processamento e questões de armazenamento de dados. Na seção 4 é fornecida a arquitetura do sistema proposto e na seção 5 é apresentado o cronograma de atividades para o segundo semestre.

## 2. Sistema de Redes e a Intrusão

Nos últimos anos, fomos apresentados às espionagens da NSA. A intrusão hostil, ou indesejada, por usuários ou *softwares* tem se configurado em um problema significativo para sistemas de rede. A intrusão por usuário é aquela que, por exemplo, pode ser feita através de um *logon* não autorizado em uma máquina. Já uma intrusão por *software* pode ter a forma de vírus, verme ou cavalo de Troia.

Os Sistemas de Detecção de Intrusão são desenvolvidos para fornecer um aviso inicial de intrusão, para que a ação defensiva possa ser tomada para impedir ou minimizar os danos [Stallings, 2008]. Há diversas formas de classificar um IDS [Bace e Mell, 2001; Axelsson 2000]. Neste artigo é adotado o modelo proposto por [Stallings, 2008], em que os Sistemas de Detecção de Intrusão são classificados de acordo com o método de detecção empregado.

### 2.1. Conceitos Básicos

Para um melhor entendimento do tema proposto, nesta seção são introduzidos os conceitos básicos da área de segurança em redes usados no decorrer do artigo: vulnerabilidade, ameaça, ataque e política de segurança. As definições são dadas pela RFC 2828 - *Internet Security Glossary* [RFC 2828, 2015].

Vulnerabilidade é uma falha, ou fraqueza, no *design*/implementação, ou no gerenciamento do sistema computacional, que pode ser explorada para violação da política de segurança do sistema. Uma política de segurança define o que pode ser permitido e o que deve ser proibido em uma infraestrutura de rede. Tipicamente, uma vulnerabilidade é um ponto de entrada para o sistema. Uma ameaça consiste na

possibilidade de violação, acidental, ou proposital, de uma vulnerabilidade em um sistema computacional causando danos. Os danos são causados por ataques.

Ataque é o acesso, não autorizado, do sistema computacional. Os ataques podem ser ativos ou passivos e podem ser iniciados de dentro do perímetro de segurança (*insider*) ou do exterior do perímetro de segurança (*outsider*). No ataque ativo é realizada a tentativa de alterar os recursos do sistema ou afetar o seu funcionamento padrão. Já um ataque passivo, tenta "aprender" ou fazer uso das informações do sistema, contudo não afeta os seus recursos. Os ataques são realizados por indivíduos que são classificados em três classes: mascarado, infrator e clandestino [Stallings, 2008].

Um indivíduo Mascarado é aquele que não tem permissão de usar o computador e que penetra nos controles de acesso de um sistema para explorar a conta de um usuário legítimo. Provavelmente é alguém de fora do perímetro de segurança. Já um usuário Infrator é aquele usuário legítimo que acessa recursos para os quais não tem autorização de acesso, ou que está autorizado para tal acesso, mas faz mau uso de seus privilégios. Geralmente, o Infrator é de dentro do perímetro de segurança. Por último, um usuário Clandestino é aquele que se apodera do controle de supervisor do sistema e utiliza esse controle para escapar de auditorias e controles de acesso, ou para suprimir provas parciais de auditoria. Um usuário Clandestino pode ser alguém de dentro ou de fora do perímetro de segurança.

A gravidade dos ataques de intrusos pode variar bastante. Em um extremo, os ataques que acontecem não interferem no funcionamento do sistema, os indivíduos simplesmente querem explorar a interligação das redes e ver o que é possível realizar. No outro extremo, há os indivíduos que tentam ler dados privilegiados, realizar modificações não autorizadas nos dados, ou interromper o sistema.

### 2.2. Registros de Auditoria - *Logs*

O registro de Auditoria, popularmente chamado de *log*, é uma ferramenta fundamental para a detecção de intrusão, porque fornece o histórico de eventos que podem conter informações relevantes dos serviços e equipamentos do sistema de rede. Sendo assim, tipicamente há duas maneiras de gerar essas informações: Por registros de auditoria nativos ou por registros de auditoria específicos [Stallings, 2008].

Os registros de auditoria nativos existem em praticamente todos os Sistemas Operacionais multiusuário. Basicamente, um programa coleta dados sobre a atividade do usuário e armazena em um arquivo. A principal vantagem é que não é preciso nenhum *software* adicional para realizar a coleta de informações. Contudo, *logs* nativos podem não conter informações relevantes, ou contê-las em um formato inconveniente.

Os registros de auditoria específicos são aqueles guiados por ferramentas destinadas a coletar e armazenar eventos relacionados com o comportamento do sistema. Esse é o caso, por exemplo, dos sistemas de detecção de intrusão. Uma vantagem é a flexibilidade que as informações são geradas (independe do sistema) e o fato que podem ser enviadas a um, ou mais, sistemas responsáveis por seu armazenamento e tratamento. Por outro lado, o *overhead* envolvido na operação de envio é a principal desvantagem.

De maneira geral, um bom IDS necessita das duas fontes de informações: registros de auditoria nativos e específicos. Os dados nativos por serem fortemente

confiáveis, mas, em muitos casos, não são completos, ou convenientes, para a análise do IDS. Assim, os dados providos por registros de auditoria específicos se fazem necessários.

### 2.3. Métodos de Detecção de Intrusão

A detecção de intrusão é baseada na suposição de que o comportamento do intruso difere daquele de um usuário legítimo de maneira que pode ser quantificada. A distinção clara e exata entre um ataque de um intruso e o uso normal dos recursos por um usuário autorizado é difícil. Espera-se que haja alguma sobreposição nos comportamentos. Essa dificuldade, aliada a sobreposição de comportamentos leva ao que se denomina de falso negativo e falso positivo. Um falso positivo é a situação na qual o IDS informa que aconteceu uma atividade maliciosa quando, na verdade, trata-se de uma atividade legítima. Um falso negativo é quando a atividade maliciosa passa imperceptível pelo IDS [CERT-BR, 2015].

Por exemplo, um falso positivo seria identificar como intrusos usuários legítimos. Para limitar os falsos positivos é possível estabelecer regras e limites mais rígidos para visualizar um comportamento médio indevido, mas, nesse caso, corre-se o risco de gerar falsos negativos, ou seja, não diferenciar os intrusos dos usuários legítimos.

Assim sendo, existe uma linha tênue do que é efetivamente uma intrusão. Há um equilíbrio relativo à escolha de detecção de intrusos. Na tentativa de enfrentar o problema de falsos positivos e falsos negativos foram propostas formas para definir comportamentos como a detecção estatística de anomalia e a detecção baseada em regras. Há ainda uma forma hibrida que combina essas duas técnicas. Esses mecanismos de detecção são discutidos a seguir.

#### 2.3.1. Detecção por Estatística de Anomalia

Na detecção estatística de anomalias realiza-se inicialmente uma coleta de dados de usuário legítimos por um período de tempo. Após isso, esses dados, estatisticamente, são aplicados a um comportamento observado para determinar, com um alto nível de confiabilidade, se esse comportamento de usuário é legítimo ou não. Existem duas categorias de Detecção Estatística de Anomalia: a detecção de limiar e a detecção baseada em perfil.

Através da detecção de limiar, é definido um limite, independente de usuário, para frequência de ocorrência de vários eventos, como a falha de autenticação na máquina. Isso envolve a contagem do número de ocorrências de um evento específico ao longo de um intervalo de tempo. Se a contagem ultrapassar o limiar, então uma intrusão é presumida. O limiar e a janela de tempo precisam ser determinados. Esses limiares tendem a gerar muitos falsos positivos e falsos negativos, visto que existe uma variabilidade entre os usuários. Contudo, eles podem ser úteis em conjunto de técnicas mais sofisticadas.

Utilizando a detecção baseada em perfil, é criado um perfil de atividades para cada usuário. Esse perfil é utilizado para detectar as mudanças no comportamento das contas individuais. Um perfil pode ser formado por um conjunto de parâmetros, de modo que o desvio em apenas um único parâmetro pode não ser suficiente para sinalizar um alerta. O fundamento dessa técnica está em analisar os registros de *log* do sistema.

De maneira geral, detecção estatística de anomalia tenta definir um comportamento normal de uso.

### 2.3.2. Detecção Baseada em Regras

Na detecção baseada em regras é feito um conjunto de regras para decidir se um determinado comportamento pertence a um intruso. Existem duas formas para implementação da técnica: a detecção de anomalia e a técnica de identificação de penetração.

A detecção de anomalia é baseada na definição de um conjunto de regras para identificar desvios de comportamentos em relação a padrões de uso anteriores. O histórico de *logs* é analisado para identificar padrões de uso e gerar automaticamente regras que descrevem esses modelos. Essas regras podem representar padrões de comportamento de usuário, programas, privilégios, intervalos de tempo entre outros. O comportamento atual é então observado, e cada transação é comparada com o conjunto de regras para determinar se está em conformidade com qualquer modelo de comportamento observado anteriormente.

Já a técnica de identificação de penetração é baseada em um sistema especialista que procura por comportamentos suspeitos. Tipicamente, as regras usadas nesses sistemas são específicas para a máquina e sistema operacional. Além disso, essas regras são geradas por especialistas, e não por meio de uma análise automatizada dos registros de *log*. Normalmente, são realizadas entrevistas com administradores de rede e com analistas de segurança que listam um conjunto de cenários de penetração conhecidos e eventos importantes que ameaçam a segurança do sistema. Assim, podemos dizer que o sucesso da técnica depende da habilidade dos envolvidos na criação das regras.

### 2.3.3. Detecção Híbrida

A detecção híbrida faz uso da detecção estatística de anomalia e detecção baseada em regras, simultaneamente. Tem o objetivo de detectar os ataques conhecidos e os não conhecidos. Com isso, eleva a taxa de detecção e reduz a taxa de falsos positivos.

A característica típica do seu funcionamento se baseia em testes nos dois modelos anteriores. Os dados são analisados de acordo com as regras contidas no modelo de detecção baseada em regras, caso não ocorra uma combinação, a análise é encaminhada para o modelo de detecção por estatística de anomalia.

## 2.4. Classificação de IDS com relação à localização

Os Sistemas de Detecção de Intrusão podem ser classificados em função da localização de onde os eventos são coletados. Sendo assim, [Kruegel et al., 2005] definem três classes de IDS: baseado em hospedeiro, baseado na aplicação e baseado em rede.

Os Sistemas de Intrusão baseados em hospedeiro (*Host Intrusion Detection System* - HIDS) tentam detectar ataques contra um equipamento específico na rede. Para isso, é feita uma análise nos seus arquivos de *logs* e também na captura dos pacotes que entram/saem da máquina para detectar sinais de intrusões, como recusa de serviços, *backdoors*, cavalos de Troia, tentativas de acesso não autorizados e execução de códigos maliciosos, por exemplo. Tipicamente, há duas fontes dos dados que o HIDS usa. As informações do sistema e os registros diários do sistema de *logs*.

As informações do sistema são providas pelo próprio sistema operacional. Ele pode avaliar informações sobre seu trabalho interno e eventos de relevância à segurança. Há programas específicos que trabalham na coleta e na exibição como os *softwares ps*, *vmstat*, *top* e *netstat*, por exemplo. A informação provida é usualmente completa e confiável, visto que é retirada diretamente do *kernel*. Além disso, alguns sistemas operacionais fornecem mecanismos para sistematicamente, e continuamente, coletar essas informações.

Um padrão fornecido por muitos sistemas baseados em Unix é o *Syslog* [Lonvick, 2001], que permite que o programador especifique o texto da mensagem descrevendo um evento para ser salvo no arquivo de *log*. Informações adicionais são automaticamente colocadas, como o tempo que o evento ocorreu e o hospedeiro que o programa está executando.

Contudo, uma dificuldade em analisar os *logs* está na maneira que algumas aplicações o utilizam. Certas aplicações usualmente utilizam o sistema de *logs* para escrever mensagens de *debugging* que não são necessariamente próprias para detecção de intrusão. Isso causa uma poluição nos *logs*. Além disso, programas específicos não seguem o padrão *Syslog*. Isso pode dificultar a extração de informações relevantes para análise. Por fim, as informações salvas no sistema de *logs* podem ser facilmente poluídas pelo atacante na tentativa de encobrir os seus rastros, tornando difícil a detecção.

Os Sistemas de Intrusão baseados em Aplicação tentam detectar ataques contra uma aplicação específica. As informações presentes nos *logs*, necessárias para tentar identificar uma intrusão, podem ser geradas pelo sistema operacional, como no caso do *Syslog*, ou obtidas por mecanismos de auditoria específicos.

Os mecanismos de auditoria específicos podem ser implementados de diferentes formas. Uma delas é modificando o código fonte do programa. É possível adicionar, ou interpor, o código responsável pela informação da auditoria nas interfaces usadas pela aplicação como, por exemplo, em uma chamada de sistema, ou em uma chamada de biblioteca. Ainda é possível usar extensões providas pela aplicação para implementar a funcionalidade de coleta de informações de auditoria. Essa abordagem garante ótima flexibilidade, porém não são todas as aplicações que oferecem tais extensões.

Por fim, os Sistemas de Intrusão baseados em Rede (*Network Intrusion Detection System* - NIDS) que tentam detectar ataques pela análise do tráfego externo da rede. Para isso, um NIDS usa sensores posicionados em locais estratégicos na infraestrutura da rede física. Um gerenciador recebe os eventos e realiza o tratamento adequado para cada situação.

Existem razões relacionadas ao ambiente e a infraestrutura de rede que podem inviabilizar a utilização de um NIDS. Por exemplo, em redes de alta velocidade, o custo computacional exigido para realizar a captura e a análise do tráfego é muito alto, podendo trazer impactos sobre o tempo de resposta do sistema. Já em redes comutadas, o desafio está em escolher a localização e a técnica mais adequada para o NIDS. Algumas soluções para esses desafios são usar espelhamento de tráfego, ou fazer com que todos os dados passem por uma porta de *up-link*.

Entretanto, a adoção de criptografia para a comunicação reduz a ação do NIDS de forma significativa. Isso porque as partes do cabeçalho criptografadas não serão

analisadas, já que o NIDS não conseguiria reconhecer as assinaturas, ou ataques, que se baseiam em dados cifrados.

## 3. *Big Data*

O objetivo desta seção é apresentar os conceitos que envolvem *Big Data*, os quais serão relevantes para o entendimento da proposta final. Aborda-se a sua definição, as formas de processamento e, por fim, o armazenamento dos dados.

### 3.1. Definição

Nos últimos anos, o termo *Big Data* tem sido difundido na tentativa de definir grandes volumes de dados, principalmente, dados não estruturados, como os de sensores, de redes sociais, e-mails, arquivos em servidores, e até mesmo registros do sistema (*logs*). Dados esses que só aumentam com o passar do tempo. De certo modo, *Big Data* tem o objetivo de tratar quantidades massivas de informações, estruturadas ou não, para gerar novas análises e entendimento sobre os conhecimentos existentes. Dessa forma, dados menores e relevantes são gerados, facilitando a busca de novas informações. Assim, na tentativa de definir o conceito de *Big Data*, muitas definições foram propostas, de tal forma que há vários trabalhos tentando organizá-las [Ward e Barker, 2013; De Mauro et al., 2014].

Em 2001, a noção inicial do que é *Big Data,* foi proposta por Douglas Laney, analista do META (atual grupo Gartner), estabelecendo os desafios e oportunidades trazidos pelo constante crescimento dos dados com o modelo de 3V [Douglas, 2001] : Volume, Velocidade e Variedade. Com o passar dos anos, o grupo Gartner fez uma nova definição: "*Big Data* são grandes volumes de dados de alta velocidade, e/ou alta variedade de informações que requerem novas formas de processamento para possibilitar uma eficiente tomada de decisão" [Beyer e Laney, 2012].

No modelo em três dimensões (3V), volume refere-se à massiva quantidade de dados gerados a cada instante de tempo; velocidade refere-se a coleta e análise dos dados que devem ser rapidamente conduzida de forma que se utilize todo o potencial da técnica; e variedade indica a grande diversidade, ou seja, são informações que podem ser não estruturadas, estruturadas e semiestruturadas. O modelo 3V é usado por diversos autores [Eaton et al., 2012; Zaslavsky et al., 2013] e também pela indústria, incluindo a IBM [Zikopoulos e Eaton, 2011] e a Microsoft [Meijer, 2011].

Porém, outros autores estenderam o modelo para 4Vs adicionando características novas para o conceito 3V como Veracidade [Schroeck et al., 2012] ou Valor [Dijcks, 2012]. A veracidade refere-se ao quão confiável os dados são. Já a característica de valor quer indicar que o problema mais crítico em *Big Data* é como descobrir informações com valores significativos, ou seja, informações importantes que tenha um alto grau de relevância em uma escala enorme de dados, de vários tipos e que ainda são gerados rapidamente. Essa noção de valor é discutida pelo grupo IDC, um dos mais influentes lideres no campo de pesquisa em *Big Data*. Em 2011, um relatório da IDC define *Big Data* como "uma nova geração de tecnologias e arquiteturas, implementadas de forma econômica para extrair valores de grandes quantidades de dados de variados tipos, habilitando alta velocidade de captura, descoberta e análise desses dados" [Gantz e Reinsel, 2011].

Adicionalmente, para o NIST (*US National Institute of Standards and Technology*), *Big Data* consiste em: "Dados com grande volume, velocidade de aquisição e representação de dados que é limitada pelos tradicionais métodos para realizar uma análise eficaz para que possam ser processados eficientemente com importantes tecnologias de *zoom*" (referindo-se a uma melhor visualização dos valores obtidos). Refletindo sobre o proposto pelo NIST, uma nova característica é adicionada à definição, contabilizando um quinto V: a visualização, ou seja, como visualizar de forma efetiva os resultados obtidos.

A indústria, e alguns autores mais modernos, já citam um modelo de 7V que adiciona volatilidade e validade, as características de velocidade, volume, variedade, veracidade e valor [Khan et al., 2014]. Como o nome indica, a volatilidade está relacionada à característica daquilo que está sujeito a mudanças frequentes. Assim, deve-se ter uma política de retenção de dados, ou seja, quando o período de retenção expirar, os dados podem ser destruídos. A característica de validade é similar a de veracidade, mas não a mesma. A validade refere-se à exatidão e precisão dos dados no que diz respeito ao uso pretendido, ou seja, os dados podem não ter problemas de veracidade, mas podem não ser válidos se não for devidamente compreendidos. Por exemplo, o mesmo conjunto de dados pode ser válido para uma aplicação e inválido para outra.

Contudo, levando em conta uma definição mais usual, neste trabalho é adotado a proposta do modelo 4V. Assim, nas próximas seções são utilizados os termos volume, velocidade, variedade e valor [Dijcks, 2012].

### 3.2. Tipos de Processamento

Tipicamente, *Big Data* tenta resolver o problema de uma massiva quantidade de dados que precisa ser processados para gerar novas informações. Sendo assim, o processamento em *batch* é bastante aplicado. Contudo, há aplicações que trabalham com outra proposta. São as aplicações onde os dados são gerados constantemente em uma alta velocidade e variedade. Assim, esses dados formam um fluxo ininterrupto de informações que precisam ser processadas em tempo real.

Em ambos os casos, há um grande volume de dados envolvidos onde é necessário gerar valores relevantes para a análise final. Atualmente, existem duas ferramentas que se destacam para esses dois modos de processamento: O Apache Hadoop [Hadoop, 2015], para processamento em *batch* e o Apache Storm [Storm, 2015], para processamento de *stream*.

#### 3.2.1. Apache Hadoop

O Hadoop é um *framework open-source* que implementa o paradigma *MapReduce* [Dean e Ghemawat, 2004]. O Hadoop foi idealizado para análise de grande quantidade de dados, é usado para muitos casos onde o processamento em lote é largamente utilizado. Sua estrutura é dividida em duas partes: armazenamento dos dados e processamento de informações.

O armazenamento de dados é feito pelo HDFS (*Hadoop Distributed File System*). Trata-se de um sistema de arquivos distribuído, altamente tolerante a falhas, projetado para executar sobre hardware comum (máquinas de prateleira). Contudo, a

plataforma do Hadoop oferece suporte a diversos sistemas de arquivos [Shvachko, 2010].

O responsável pelo processamento de informações é o *MapReduce*, que é um conceito de programação paralela proposto pela Google em 2004. A ideia é escrever aplicações distribuídas para processamento de grandes quantidades de dados. O *MapReduce* é baseado em duas primitivas. A primitiva *map* é responsável por mapear fragmentos de dados de entrada a uma chave. A primitiva *reduce* tem o objetivo de realizar uma computação sobre os dados mapeados por uma mesma chave. Em sua essência, é um modelo mestre/escravo.

Os dados de entradas são introduzidos no sistema de arquivos e distribuídos entre os nós para realização do processamento (*map*). Ao final do processo os valores resultantes são devolvidos ao nó originador (*reduce*).

O Hadoop é muito utilizado em empresas que tem um grande volume de dados estáticos. Tipicamente, o Hadoop é usado para indexação web como solução para empresas de mineração de dados que fazem uso de *e-commerce*.

### 3.2.2. Apache Storm

O Hadoop, como visto anteriormente, é empregado para processamento em *batch* (dados estáticos), porém, em algumas situações é necessário que a análise seja feita em tempo de execução. Para esses casos, há a ferramenta Apache Storm.

O Apache Storm analisa os dados em tempo de execução (dados dinâmicos). Os dados de entrada do Storm são fluxos de informação que nunca terminam. Diferentemente das tarefas do Hadoop, os fluxos de dados no Storm processam os dados à medida que eles chegam. O Storm é voltado para aplicações como o *Twitter*.

## 3.3. Armazenamento de dados

O grande volume de dados que *Big Data* trata normalmente é armazenado em coleções de bases de dados com alto volume e grande variedade. Essas características acabam tornando muito difícil, e complexo, a realização de operações simples, como remoção, ordenação e sumarização de forma eficiente, utilizando os Sistemas Gerenciadores de Bases de Dados (*Data Base Management System* - DBMS) tradicionais, ou seja, RDBMS (*Relational Data Base Management System)* [Vieira et al., 2012].

O principal problema dos RDBMS está na dificuldade em lidar com dados não estruturados e escalar horizontalmente, ou seja, não perder o desempenho na medida em que o banco de dados aumenta. Assim, o gerenciamento de dados requer técnicas para manter a disponibilidade e um alto desempenho em um ambiente de dados variados e distribuídos. Muitas dessas técnicas de gerenciamento de dados são agrupadas sob o termo “banco de dados NoSQL” [Loshin, 2013]. O termo NoSQL é comumente referido a “*Not Only* SQL”. O objetivo das soluções NoSQL não é substituir os modelos de banco de dados relacionais, mas sim ser utilizado em situações onde uma maior flexibilidade na estruturação da base de dados seja necessária [Chang et al., 2008].

Tipicamente, sistemas NoSQL prometem características como grande flexibilidade no gerenciamento do banco de dados, escalabilidade horizontal, ausência de esquema, o que permite, de forma fácil, a alta escalabilidade e o aumento da

disponibilidade. Contudo, não garantem a integridade dos dados, diferente dos modelos relacionais que mantém uma estrutura rígida [Lóscio et al., 2011].

Os bancos de dados NoSQL podem ser categorizados segundo o modelo de armazenamento dos dados. Dessa forma, há diversas maneiras de classificá-los [NoSQL, 2015]. Uma delas segue o proposto por [Han et al., 2011] que apresenta três modelos: chave-valor, orientado a colunas e orientado a documentos. Adicionalmente, há o modelo orientado a grafos [Moniruzzaman e Hossain, 2013; He, 2015; Bajpayee et al., 2015]. Neste trabalho são investigados os quatro modelos citados.

No modelo chave-valor (K*ey-Value Store*) os dados são representados como uma coleção de pares de chaves e de valores, de tal forma que cada chave aparece no máximo uma única vez na coleção. As chaves são identificadores alfanuméricos. Os valores podem ser sequências de texto simples ou listas mais complexas. As pesquisas por dados, normalmente, só podem ser realizadas com o conteúdo das chaves, não dos valores, e são limitadas a correspondências exatas. São exemplos típicos desse modelo de banco de dados o Redis [Redis, 2015] e DynamoDB [DynamoDB, 2015].

O modelo orientado a colunas (C*olumm Store*) organiza os dados em tabelas. Esse modelo é similar a um RDBMS, porém, o conteúdo é armazenado por colunas ao invés de linhas. Cada dado é estruturado em três partes: nome único que representa a coluna referenciada, um valor que identifica o tipo de dado e um *timestamp* que o sistema usa para identificar um conteúdo válido. É um modelo escalável, visto que o aumento no tamanho dos dados não irá resultar em uma diminuição na velocidade de processamento. Assim, é muito adequado para processamento de volumes massivos de dados. Os exemplos de banco de dados orientados a coluna são o HBase [HBase, 2015] e Cassandra [Cassandra, 2015].

O modelo orientado a documentos (*Document Store*) foi projetado para armazenar e gerenciar documentos eletrônicos. Um documento é codificado em um formato padrão de dados como XML, JSON (*Javascript Option Notation*) ou BSON (*Binary* JSON). Em contraste com o modelo chave-valor, o valor da coluna contém os dados semiestruturados (especificamente, pares nome-valor). Uma única coluna pode conter centenas desses atributos, assim como o número e tipo de atributos gravados podem variar de linha a linha. Além disso, sua principal vantagem, com relação ao modelo chave-valor, é a procura em ambos os campos, ou seja, pode realizar consultas pelo conteúdo do documento. São exemplos de banco de dados orientados a documentos o ElasticSearch [ElasticSearch, 2015] e MongoDB [MongoDB, 2015].

A teoria dos grafos é aplicada ao modelo orientado a grafos (*Graph-Based Store*) para armazenar e recuperar dados. Esse modelo prima pela interligação dos diferentes pares de dados. As unidades de dados são visualizadas como nós, enquanto que as arestas são representadas pelas relações entre os dados. É o único modelo, dos quatros estudados, que se preocupa com as relações entre os dados. Essa característica é essencial para o armazenamento de informações em redes sociais, por exemplo. Além disso, seu foco é na representação visual de informações tornando-o mais amigável do que outros modelos. Neo4j [Neo4j, 2015] e InfoGrid [InfoGrid, 2015] são exemplos de banco de dados representados por esse modelo.

## 4. Proposta

A proposta do trabalho é implementar um IDS que use o método de detecção híbrida para identificar intrusos e que emprega técnicas de *Big Data* para correlacionar os *logs*. A arquitetura do IDS é fundamentada no modelo proposto por [Kruegel et al., 2005] que baseia-se em uma sequência de blocos encadeados que servem para adquirir, armazenar, filtrar e analisar os eventos e gerar alertas.

Inicialmente, necessitamos de um normalizador de *logs* na arquitetura, porque existem diversos equipamentos gerando *logs* de formatos distintos no ambiente de produção do PoP-RS. Esse *software* deve ser capaz de uniformizar todos os modelos de arquivos para o formato *Syslog*. Para correlacionar os *logs* será utilizado a técnica *MapReduce* do Apache Hadoop, que usa o HDFS para armazenamento de dados através do DBMS HBase (nativo do Hadoop). Como o Hadoop trabalha com processamento em *batch*, será utilizada uma janela de tempo para a análise e para coleta de dados. A escolha do Hadoop é devido a este ser uma solução em *software* livre.

O IDS proposto neste trabalho será independente de posição na infraestrutura, visto que os *logs* de diferentes fontes são enviados para o armazenador de *logs* centralizado (Figura 1). Dessa forma, o IDS possuirá características dos três modelos de localização, conforme apresentado na seção 2.4.

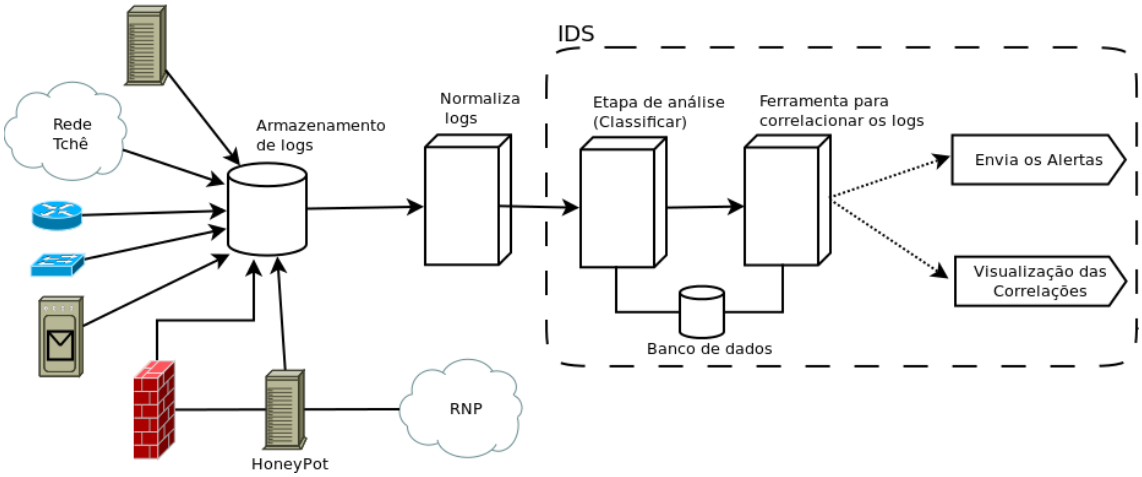



**Figura 1. Ambiente de produção e arquitetura proposta para o IDS.**

A forma de alerta inicial será dada através de envio de *e-mail* para um administrador cadastrado. Além disso, a solução deverá ter uma interface para os *logs* com o objetivo de melhorar a visualização e busca dos mesmos.

## 5. Cronograma

As atividades, para realização do Trabalho de Graduação II (TG-II), são divididas em cinco etapas principais: implementação, testes, validação, escrita e apresentação.

A etapa da implementação se divide no desenvolvimento do IDS do tipo híbrido, construção do módulo de correlacionamento com o Hadoop e o armazenamento dos *logs*. Os testes iniciais serão realizados a fim de verificar se o protótipo desenvolvido atende a funcionalidade desejada. Para validação final da ferramenta serão empregados os dados de produção PoP-RS. Durante o período de testes e de validação, será

realizado, em paralelo, a redação do TG-II e, por fim, elaborada sua apresentação. O do cronograma de atividades pode ser visto na Tabela 1.

| Etapas | Agosto | Setembro | Outubro | | Novembro | Dezembro |
|---|---|---|---|---|---|---|
| Implementação | x | x | | | | |
| Testes | | x | x | | | |
| Validação | | | | x | x | |
| Escrita | | x | x | | x | |
| Apresentação | | | | | | x |

**Tabela 1. Cronograma para o segundo semestre de 2015.**

## Referências


Abad, C., Taylor, J., Sengul, C., Yurcik, W., Zhou, Y., e Rowe, K. (2003). *Log* correlation for intrusion detection: A proof of concept. In *Computer Security Applications Conference, 2003*. *Proceedings. 19th Annual*, IEEE (pp. 255-264).

Axelsson, S. (2000). Intrusion detection systems: A survey e taxonomy (Vol. 99). Technical report.

Bace, R., e Mell, P. (2001). NIST special publication on intrusion detection systems. *BOOZ-ALLEN AND HAMILTON INC MCLEAN VA*.

Bajpayee, R., Sinha, S. P., e Kumar, V. (2015). Big Data: A Brief investigation on NoSQL Databases.

Beyer, M.A. e Laney, D., (2012). The Importance of "Big Data": A Definition. Gartner Publications, pp.1-9.

Bray, R., Cid, D., e Hay, A. (2008). OSSEC host-based intrusion detection guide. Syngress.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., e Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems* (TOCS), 26(2), 4.

Cassandra. https://cassandra.apache.org/, acessado em junho de 2015.

CERT-BR. Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil. http://cartilha.cert.br/mecanismos/, acessado em junho de 2015.

De Mauro, A., Greco, M. e Grimaldi, M. (2014). What is Big Data? A Consensual Definition and a Review of Key Research Topics. In *4th International Conference on Integrated Information.*

Dean, J., e S. Ghemawat (2004). Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation*. USENIX Association.

Dijcks, J. (2012). Oracle: Big data for the enterprise. Oracle White Paper.

Douglas, L. (2001). 3-d data management: Controlling data volume, velocity and variety. META Group Research Note.

Dynamodb. http://aws.amazon.com/dynamodb/, acessado em junho de 2015.

Eaton, C., Deroos, D., Deutsch, T., Lapis, G., e Zikopoulos, P. (2012). Understanding big data.

ElasticSearch. https://www.elastic.co/, acessado em junho de 2015.

Gantz, J., e Reinsel, D. (2011). Extracting Value from Chaos.

Hadoop Document. http://www.hadoop.apache.org/, acessado em maio de 2015.

Han, J., Haihong, E., Le, G., e Du, J. (2011). Survey on NoSQL database. In Pervasive computing and applications (ICPCA), In *6th international conference*, IEEE (pp. 363-366).

HBase. http://hbase.apache.org/, acessado em junho de 2015.

He, C. (2015). Survey on NoSQL Database Technology. Journal of Applied Science and Engineering Innovation Vol, 2(2).

InfoGrid. http://infogrid.org/trac/, acessado em junho de 2015.

Khan, M., Uddin, M. F., e Gupta, N. (2014). Seven V's of Big Data understanding Big Data to extract value. In *American Society for Engineering Education (ASEE Zone 1), 2014 Zone 1 Conference of the IEEE* (pp. 1-5).

Kruegel, C., Valeur, F., e Vigna, G. (2005). "Intrusion detection and correlation: challenges and solutions" (Vol. 14). Springer Science and Business Media.

Lynch, C. (2008). Big data: How do your data grow?. Nature, 455(7209), 28-29.

Lóscio, B. F., Oliveira, H. R. D., e Pontes, J. C. D. S. (2011). NoSQL no desenvolvimento de aplicações Web colaborativas. *VIII SIMPÓSIO BRASILEIRO DE SISTEMAS COLABORATIVOS, Paraty, RJ: SBC*.

Loshin, D. (2013). Big data analytics: from strategic planning to enterprise integration with tools, techniques, NoSQL, and graph. Elsevier.

Lonvick, C. (2001). RFC 3164: The BSD sys*log* protocol. Network Working Group.

Mariani, L., e Pastore, F. (2008). Automated identification of failure causes in system *logs*. In *Software Reliability Engineering. ISSRE 2008. 19th International Symposium,* IEEE (pp. 117-126).

Meijer, E. (2011). The world according to LINQ. Communications of the ACM, 54(10), 45-51.

MongoDB. http://www.mongodb.org/, acessado em junho de 2015.

Moniruzzaman, A. B. M., e Hossain, S. A. (2013). NoSQL Database: New Era of Databases for Big data Analytics-Classification, Characteristics and Comparison. *International Journal of Database Theory & Application*, *6*(4).

Neo4j. http://neo4j.com/, acessado em junho de 2015.

NoSQL: List of NoSQL databases. http://nosql-database.org/, acessado em junho de 2015.

Ortoza (2014) “Beyond Web Aplication *Log* Analysis using Apache Hadoop”, http://www.orzota.com/wp-content/uploads/2014/04/loganalysis-paper.pdf

PoP-RS/RNP, Ponto de Presença da Rede Nacional de Ensino e Pesquisa do Estado no Rio Grande do Sul, http://www.pop-rs.rnp.br/, acessado em junho de 2015.

Redis. http://www.redis.io/, acessado em junho de 2015.

RFC 2828. http://www.rfc-base.org/txt/rfc-2828.txt, acessado em junho de 2015

Roesch, M. (1999). Snort: Lightweight Intrusion Detection for Networks. In LISA (Vol. 99, No. 1, pp. 229-238).

Schroeck, M., Shockley, R., Smart, J., Romero-Morales, D., e Tufano, P. (2012). Analytics: The real-world use of big data. IBM Institute for Business Value - executive report, IBM Institute for Business Value.

Shvachko, K., Kuang, H., Radia, S., e Chansler, R. (2010). The hadoop distributed file system. In *Mass Storage Systems and Technologies* (MSST), 2010 IEEE 26th Symposium on (pp. 1-10).

Splunk. http://www.splunk.com/, acessado em junho de 2015.

Stallings, W. (2008) “Criptografia e segurança de redes: princípios e práticas”, Pearson Prentice Hall, 4th edição.

Storm. https://storm.apache.org/, acessado em junho de 2015.

Vernekar, S. S., e Buchade, A. (2013). MapReduce based log file analysis for system threats and problem identification. In *Advance Computing Conference (IACC), 2013 IEEE 3rd International*. IEEE (pp. 831-835).

Vieira, M. R., Figueiredo, J. M. D., Liberatti, G., e Viebrantz, A. F. M. (2012). Bancos de Dados NoSQL: conceitos, ferramentas, linguagens e estudos de casos no contexto de Big Data. SIMPÓSIO BRASILEIRO DE BANCO DE DADOS. São Paulo.

Ward, J. S. e Barker, A. (2013). Undefined By Data: A Survey of Big Data Definitions. arXiv preprint arXiv:1309.5821.

Zaslavsky, A., Perera, C., e Georgakopoulos, D. (2013). Sensing as a service and big data. arXiv preprint arXiv:1301.0159.

Zikopoulos, P., e Eaton, C. (2011). Understanding big data: Analytics for enterprise class hadoop and streaming data. McGraw-Hill Osborne Media.