THREAT IDENTIFICATION FROM ACCESS LOGS USING ELASTIC STACK

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Suhanthan Vethanayagam

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

November 2020

Fargo, North Dakota

North Dakota State University
Graduate School

Title

THREAT IDENTIFICATION FROM ACCESS LOGS USING ELASTIC STACK

By

Suhanthan Vethanayagam

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Anne Denton
Chair

Dr. Simone Ludwig

Dr. Sudarshan Srinivasan

Approved:

November 22, 2020
Date

Dr. Simone Ludwig
Department Chair

ABSTRACT

Access log management is an essential part of cybersecurity. Lack of insight into user authentication patterns can hinder readiness and reaction to the growing threat of cyberattacks. Central Authentication Service (CAS) log is underutilized in threat detection due to its detailed and complex logging nature. This paper investigates the feasibility of turning unfriendly CAS logs into helpful datapoints utilizing Elastic Stack (Filebeat, Logstash, Elasticsearch and Kibana) to detect anomalies.

CAS logs are collected by Filebeat and forwarded to Logstash. The deployment of a custom Grok filter in Logstash facilitates the normalization of complex CAS logs and the resulting data is indexed in Elasticsearch. A Python program using Elasticsearch's aggregate function was developed to query the indexed data and compare password and multi-factor submission counts. This mechanism was found to have potential in detecting threats. Finally, Kibana's rich visualization capabilities allow for exploring and shaping of data in innovative ways.

ACKNOWLEDGMENTS

I would like to take this opportunity to thank everyone that helped me through this process. First of all, I would like to thank my graduate advisor, Dr. Anne Denton, whose continued guidance and support have been instrumental in finishing this paper. I would also like to thank my other graduate committee members, Dr. Simone Ludwig and Dr. Sudarshan Srinivasan, for their time and feedback.

Thank you to my employer, Enterprise Computing and Infrastructure at NDSU, for allowing me to pursue my graduate degree and also permitting to use some of the resources needed for this project. Thank you to my colleague and friend, Richard Frovarp, for all the assistance he provided in completing this project.

I wish to thank my friends for all their encouragement. Finally, I would like to thank my family for their endless support, love, and patience throughout this journey.

DEDICATION

To my late father and educator, Mr. Vethanayagam,

and to

the strong and independent women who raised me,

my mother, Urmila Vethanayagam, and grandmother, Alagammah Sinnathurai.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

LIST OF ABBREVIATIONS

CAS.....Central Authentication Service

SSO .....Single Sign-On

MFA.....Multi-Factor Authentication

LDAP .....Lightweight Directory Access Protocol

SAML .....Security Assertion Markup Language

TGT.....Ticket Granting Ticket

ST.....Service Ticket

CSV.....Comma-Separated Values

HTTPD.....HyperText Transfer Protocol Daemon

AWS.....Amazon Web Services

JSON.....JavaScript Object Notation

SPL .....Search Processing Language

SQL.....Structured Query Language

NoSQL .....Not Only Structured Query Language

RDBMS.....Relational Database Management System

API .....Application Programming Interface

HTTP.....HyperText Transfer Protocol

REST.....REpresentational State Transfer

DSL .....Domain Specific Language

RHEL .....Red Hat Enterprise Linux

UFW.....Uncomplicated Firewall

KQL .....Kibana Query Language

AI .....Artificial Intelligence

IP .....Internet Protocol

1. INTRODUCTION

1.1. Motivation

Digital threats increasingly necessitate monitoring of any online activity. With the frequency and lethality of cyberattacks increasing rapidly, it is paramount that IT service providers pay close attention to user behaviors for any suspicious activities. A successful cyber defense mechanism should have multiple layers and elements [1]. It involves humans, security policies, and technologies and every single component involved must work together to make the defense effective [1]. Log management is crucial to mitigating digital threats. Security and authentication logs could shine a light on unusual behavior, traffic, and access patterns [2].

North Dakota State University (NDSU) utilizes two major single sign-on (SSO) methods - Central Authentication Service (CAS), and Shibboleth Authentication. Slightly more than half of NDSU web applications use CAS. Naturally, logs generated by CAS contain crucial information about user login behavior. Analysis of *cas_audit.log* data could provide valuable insights into historic user access information and patterns.

At an institution such as NDSU, there are thousands of users accessing the university's systems regularly, and they all have different backgrounds and varying levels of computer security knowledge. For many users, convenience usually trumps security. IT professionals, therefore, continue to deal with the issue of people using the same or similar passwords for all their services. Furthermore, with academic collaborations around the world, inbound network traffic to NDSU is not limited to a specific geographic region. This could make identifying threats challenging. The emergence of cryptocurrencies, and the temptation to produce them inexpensively, has led to new types of cyber threats – cryptojacking [4]. Because of the vast amount of computing resources available on campuses, these threats pose an elevated risk to institutions of higher education. As a

university, NDSU has a unique set of users with multiple roles to manage, including students who are part-time employees and employees who are part-time students. This could potentially lead to lateral movement attacks, in which, bad actors begin by regularly targeting lower-level users in a system and then progressively attempt to escalate their targets to gain more privileges.

The Ninth Annual Cost of Cybercrime Study, a report published in 2019 by Accenture, argues that "humans are still security's weakest link" [5]. While continued cyber awareness campaigns remain one of the most powerful tools to educate the public, it cannot be assumed that everyone will always make smart cybersecurity choices. Continuous monitoring of CAS logs could help avoid any blind spots and enhance NDSU's already strong cybersecurity apparatus.

1.2. Organization

This paper consists of six chapters. The first chapter introduces the reader to the subject matter and discusses the challenges relating to cybersecurity. This chapter also explains the importance of monitoring user access logs in order to combat cyber threats. The second chapter presents a background of the IT infrastructure involved in this project and the problem domain. The third chapter describes the technologies used in the proposed solution and explains the role each plays in solving the problem. The fourth chapter discusses how the proposed solution's architecture was designed in practice and then implemented. Chapter five looks at the solution, evaluates its efficiency and prospects. It also describes how the solution can be useful in solving real world problems. Finally, chapter six summarizes all the tasks completed and discusses any potential future work.

1.3. Problem Statement

Central Authentication Service (CAS) has been used to provide single sign-on (SSO) login capabilities at NDSU for almost a decade. In the recent past, NDSU has also adopted multi-factor

authentication (MFA) as an added security measure. CAS generates detailed audit logs for every security step in the authentication process. This results in lengthy CAS logs that are difficult to navigate and discern. Even though these logs contain a wealth of user authentication data, most of the data were not readily available for human consumption due to CAS's complex and disjointed logging pattern. Largely ignored, the logs were underutilized by IT Security in their effort to gain insight into user authentication patterns that could help establish cybersecurity policies.

The solution was to develop a system that structured CAS logs, made them more user friendly, and allowed anyone to easily analyze the output. Given the current budgetary constraints, this solution needed to make use of open-source components. The developed solution had to be a customizable system that would outlast CAS version upgrades and any future design changes. It also needed to be flexible so that it could be deployed to process and manage other types of logs in the future.

The developed solution was to use Elastic Stack (a collection of open-source projects - Filebeat, Logstash, Elasticsearch and Kibana) to facilitate the processing, managing, and analyzing of CAS logs. Each part of the Elastic Stack is a lightweight component and acts as a building block of a pipeline. This allows for individual handling of each part and provides the flexibility and customization needed for the configuration and deployment of the solution as opposed to a single, large product such as Splunk. This design is particularly advantageous when it comes to scalability. Unlike the commercial product Splunk, this open-source solution is, of course, very cost effective.

2. BACKGROUND

2.1. Central Authentication Service (CAS)

CAS is an open-source single sign-on (SSO) authentication mechanism for web applications. CAS is used to authenticate user access to multiple services employing a single set of username and password credentials [6]. The function of CAS can be compared to that of using Facebook or Google credentials to log into other third-party web applications. The term CAS is often used to refer to both the authentication protocol and the server it uses for single sign-on authentication [7]. CAS's centralized design eliminates the need for each web application to store and maintain sensitive information on their own and also reduces the number of potential attack vectors. In fact, users' security credentials are not even shared with web applications during the authentication process. CAS can also help with password fatigue by reducing the number of credentials users must remember and thus encourages good password habits [8].

CAS was developed at Yale University and academic institutions were the first to adopt this technology. According to the Apereo Foundation, the custodial organization of this mechanism, CAS has now been embraced by several large private sector companies around the world [9]. CAS supports clients written in a variety of languages, is well documented, and its open-source development is supported by an active community of contributors [9]. CAS is compatible with multiple pluggable authentication technologies, including Lightweight Directory Access Protocol (LDAP), databases, and Active Directory [10]. In addition to CAS's own authentication protocol, web applications (CAS clients) could optionally use other protocols such as Security Assertion Markup Language (SAML), REpresentational State Transfer (REST), OAuth, OpenID, etc. to communicate with the CAS server [10]. This flexible compatibility makes CAS a versatile system. As previously stated, CAS supports second-factor authentication and MFA products from

multiple providers can be used with CAS [10]. Here at NDSU, CAS is utilized to manage SSO access to web applications, such as Google Suite, Library, DocuSign, Moodle, Medicat, etc. Figure 1 below shows the CAS system architecture [13].

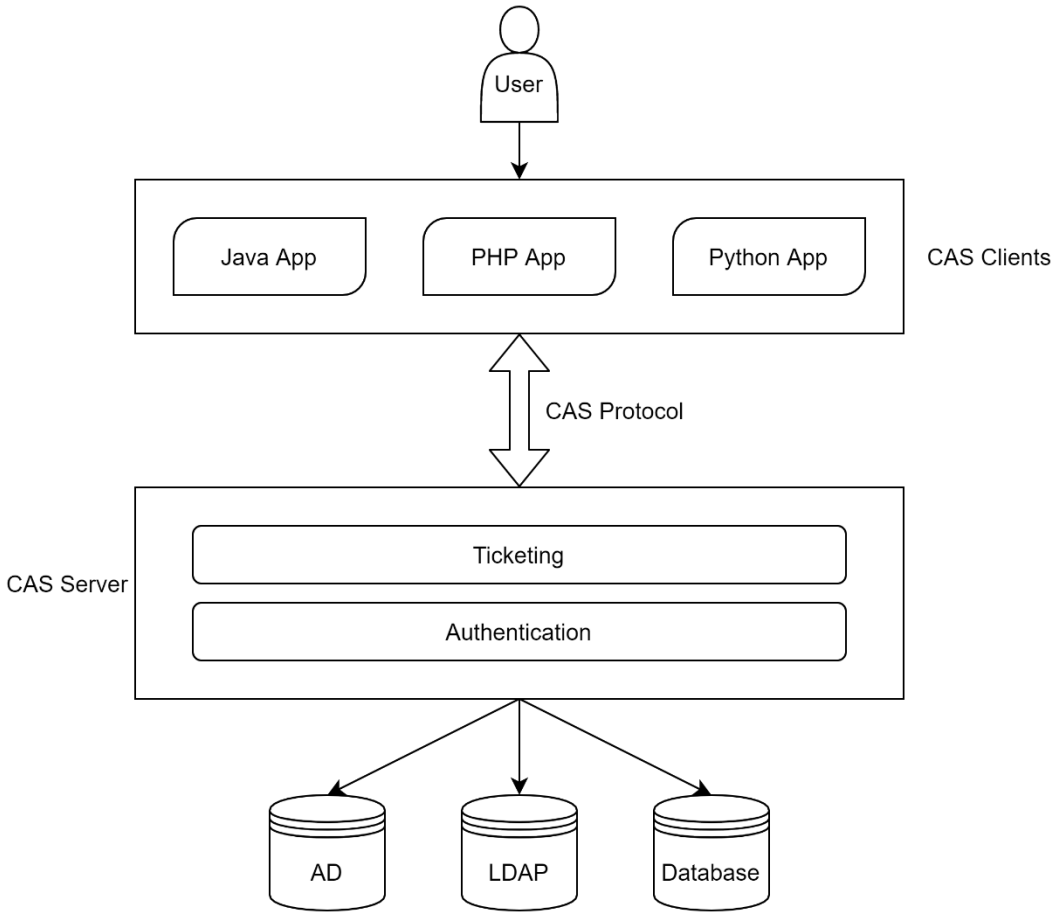

Figure 1: CAS High-Level Architecture

CAS system has four major components:

1. CAS Server: The system that is tasked with authenticating users' credentials and granting access to the web application.
2. CAS Client: Any web application that uses the CAS Server as its authentication mechanism.
3. Ticket Granting Ticket (TGT): The SSO cookie created by the CAS server after a successful authentication. The TGT acts as the session key for the user's SSO session

allowing authentication without having to enter credentials multiple times during that specific window of time [11].

4. Service Ticket (ST): A key piece of information passed back to the user via browser by CAS server upon a successful credential validation [11]. In return, browser would submit this to the web application for access. Web application would check the ST's validity with CAS server prior to granting final access.

2.2. Multi-Factor Authentication (MFA)

Multi-factor is an authentication method that requires two or more pieces of information for any successful access into a computer system. This enhances security by reducing the sole reliance on a password that may or may not be strong and unique. Required multi-factor authentication information can be grouped into three different categories - something you know (password, PIN), something you have (smart card, SMS code, smart phone push), or something you are (fingerprint, iris scan, face) [12]. For a successful authentication, user credentials must come from at least two of the categories listed above.

Most of the services at NDSU require two factors to login and a commercial product called Duo is used to deliver the extra factor. Available options to users at NDSU for receiving a second factor include phone calls, text messages, and notifications on a mobile device. As a backup method, a set of predetermined access codes delivered via SMS is another way to successfully negotiate the second factor authentication.

Although multi-factor authentication significantly improves cybersecurity and reduces malicious activities, it is not a foolproof system. The rate of adoption is still not where it should be, and there are still many private and public services not secured by multi-factor authentication. Even when the mechanism is in place, it could still be susceptible to phishing, social engineering,

and other man-in-the-middle attacks. In particular, SMS based MFA methods have come under scrutiny for being less secure than other available options. SMS based delivery is vulnerable to SIM swap and other advanced signal interception attacks [13]. The dependency on telecom companies and their ability to read text messages make SMS based MFA options less than ideal. For average users, regardless of the way they choose to receive the extra factor, MFA is still immensely helpful in thwarting cyberattacks. The general population, therefore, should be encouraged to embrace this method. Figure 2 below describes CAS Server and MFA Workflow.

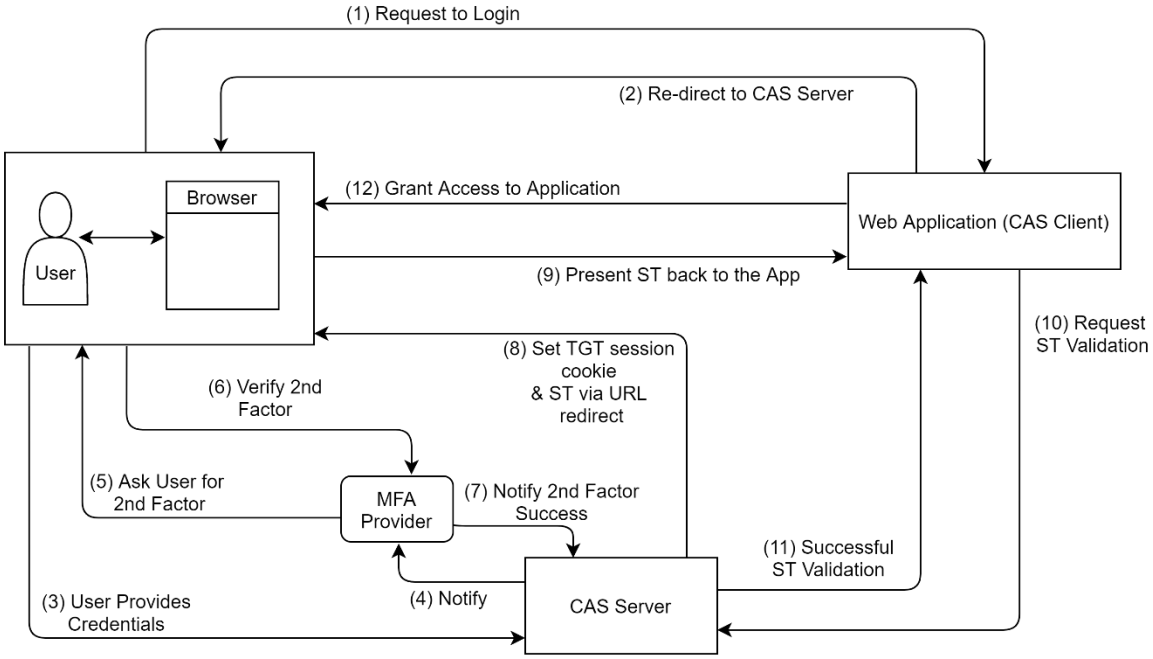

Figure 2: CAS Server and Protocol Workflow with MFA

2.3. CAS Logs

As stated in the problem statement in Section 1.3, CAS logs are often difficult to track and understand due to their length and detailed logging pattern. As the name suggests, CAS multi-factor login is a multi-step process that is relatively slower than simply using a username and password to login. From users entering their credentials to completing the MFA action, there are a lot of events taking place in the background. While the logs are written, each component of one

user's login can be interrupted by login events belonging to other users. CAS logs, therefore, are not grouped by users. Furthermore, each action of the login process for a user is logged in separate blocks in a multiline pattern, and information such as the username, source IP, destination IP, etc. are repeated in each block. Details corresponding to a single login attempt are split into bits and pieces and anyone trying to see a specific login attempt will have to track down multiple blocks of logs.

This disjointed logging creates a 'needle in the haystack' situation. Detailed logging is generally desirable and advantageous, but as shown in Figure 3 below, it makes it difficult to follow what is going on just by reading the logs. It would be nearly impossible to keep up with the events of a user's login attempt, especially when the system is busy. To alleviate this, a solution was needed to make it easier to identify suspicious activities - a solution that would automatically structure the CAS logs in real-time or near real-time and output actionable, user-friendly data.

```
2020-10-20 04:07:32,712 INFO [org.apereo.inspektr.audit.support.Slf4jLoggingAuditTrailManager] -
Audit trail record BEGIN
=============================================================
WHO: jane.doe
WHAT: [event=mfa-duo,timestamp=Tue Oct 20 04:07:32 CDT
2020,source=PredicatedPrincipalAttributeMultifactorAuthenticationPolicyEventResolver]
ACTION: AUTHENTICATION_EVENT_TRIGGERED
APPLICATION: CAS
WHEN: Tue Oct 20 04:07:32 CDT 2020
CLIENT IP ADDRESS: 10.10.10.10
SERVER IP ADDRESS: 20.20.20.20
=============================================================


2020-10-20 04:07:32,713 INFO [org.apereo.inspektr.audit.support.Slf4jLoggingAuditTrailManager] -
Audit trail record BEGIN
=============================================================
WHO: jane.doe
WHAT: [event=mfa-duo,timestamp=Tue Oct 20 04:07:32 CDT
2020,source=PrincipalAttributeMultifactorAuthenticationPolicyEventResolver]
ACTION: AUTHENTICATION_EVENT_TRIGGERED
APPLICATION: CAS
WHEN: Tue Oct 20 04:07:32 CDT 2020
CLIENT IP ADDRESS: 10.10.10.10
SERVER IP ADDRESS: 20.20.20.20
=============================================================
```

Figure 3: Sample CAS Logs

2.4. Literature Review on Log Management

Log is a diagnostic tool that captures all the events of a computer system [3]. The explosive growth and prevalence of computers over the years have dramatically increased the size and complexity of logs generated. These changes have warranted the need to manage logs more efficiently. Log management is crucial to solving issues related to computer security, regulatory compliance, and system operations [14]. It consists of multiple phases, including collection, centralized aggregation, storage, analysis, monitoring, and retention [3]. The sheer volume of data generated through modern computing logs require proper aggregation techniques and centralized storage methods. Inconsistent log content and log formats are two of the biggest challenges faced in log management [3]. The lack of structure in logs hinders seamless analysis and insight gathering, thus requiring the need for log transformation and filtration prior to indexing logs and retrieving them.

Modern operating systems already contain tools [15] for basic text processing and extraction. Such tools include grep, awk, and sed [16]. These simple tools, however, are not capable of advanced analysis. Furthermore, the increase in security threats limits the effectiveness of simple tools that are built into operating systems, as they are entirely reactive and manual in nature. As for computer security, reactive methods may not be adequate. More advanced, dedicated log analysis and management tools are required to keep up with changing digital threats.

Ideally, log analysis and event alerting must happen in real-time or near real-time to be effective. This is where purpose-built tools enter the log management landscape. The options range from commercial products to open-source projects. One such commercial product, and a leading log management tool, is Splunk [17]. On the open-source side, Elastic Stack is a powerful log analytics and search engine. Both Splunk and Elastic Stack are capable of fulfilling all the phases

of log management, from collection to retention. Splunk is a commercial product with functionality comparable to Elastic Stack, which is used in the proposed solution.

The Splunk system is made up of three parts – Forwarders, Indexers, and Search Heads [18]. Forwarders collect and forward data from multiple sources to the indexing phase. The Indexers' job is to store data, which they do in denormalized form to aid retrieval performance. These Indexers store data without any predefined schema and only apply mappings at search time, depending on the issued query [18]. This allows for users to explore their log data even with very little knowledge about them. Splunk indexes data based on timestamps and in the event of missing timestamps, Indexers are capable of using file modification time to process data [19]. Search Head provides a web interface for users to issue queries on the indexed data and get results in various visual forms [18]. Splunk's Search Processing Language (SPL) uses Unix pipeline command syntax that gets translated into Structured Query Language (SQL) queries in the background [19]. This makes it easier for users to run complex queries, such as `join`, without much proficiency in SQL. Splunk performs automatic event processing at index time [20] and also accommodates custom data transformation at search time [21]. These custom transformations in Splunk are managed by two configuration files - `transforms.conf` and `props.conf` [21] - using regular expressions. A regular expression is a series of characters or text that describes a search pattern [22].

Splunk is currently deployed at NDSU with very minimal custom configurations for transforming logs. Splunk's built-in, automatic event processing capabilities are employed to collect, store, and retain Windows events logs. Logs not supported by Splunk's automatic event processing are stored in their original form without transformation. However, NDSU utilizes Splunk's Search Head to create a few custom reports and charts of Windows events.

3. APPROACH

This chapter briefly describes major components of the proposed solution - Elastic Stack. This chapter also explains how each component of Elastic Stack meets the needs set out in the problem statement.

Elastic Stack is a data management and analytics solution that is a combination of four different open-source projects from the Elastic Company. The stack is comprised of Elasticsearch, Logstash, Kibana and Beats. Elastic Stack gives the ability to streamline data gathering from multiple sources. It then helps with processing, analyzing, and ultimately visualizing the data in readily consumable and user-friendly ways. Partly due to its open-source nature and numerous customizable configurations, Elastic Stack may present with some initial difficulties to get the system up and running. It does, however, provide more control and flexibility with designing the proposed solution. Elastic Stack makes gathering valuable insights from data easier and facilitates monitoring, diagnosing, and troubleshooting issues related to IT infrastructure and security.

3.1. Beats

The first step in any log management process is gathering logs, and Beats fulfills the role of a data shipper. It is lightweight [23], sits at the very bottom of the Elastic Stack hierarchy, and forwards data for further processing, storing, and analysis. Ideally, logs should be sent to a centralized location that can help establish patterns when analyzing historic data. This would ensure that crucial events are not missed. Due to its lightweight nature, Beats can sit on servers without consuming many resources. Beats can output data directly to Elasticsearch for storing or can first send data to Logstash for transforming and parsing [23]. There are multiple types of Beats shippers available for various types of data; Filebeat, Metricbeat, Packetbeat, Auditbeat, Winlogbeat are some of the commonly used types [23]. For the purpose of shipping CAS logs

from the server, Filebeat would be used and logs would be first sent to Logstash for further transformation.

3.2. Logstash

Logstash is a server-side data processing program [24] that can accept data from various sources, transform them, and then send them out for storing. Logstash is made up of three parts – input, filter, and output. Logstash can handle data in multiple formats with varying degrees of size and complexity [24]. Data transformation in Logstash is done by filter plugins. Logstash, with the help of filters, can help turn unstructured logs into structured and uniformed data. Filters are also used to exclude unwanted parts before logs get indexed. Once the transformation is done, Logstash is capable of sending the resulting data to a multitude of destinations in different formats [24]. Outputs can be delivered to a range of cloud storages, databases, and commercial-events-alerting products. They can be written to local disk in comma-separated values (CSV), text, and other formats [24]. In this project, Logstash would be sending the output to Elasticsearch for storing, further analysis, and querying.

3.3. Grok Patterns

In order to analyze and query the data and produce meaningful results, the data has to be structured. Filters are responsible for this critical task of enriching data in Logstash. Grok filter is one of the most widely used filter plugins that can turn arbitrary text into structured and queryable data [25]. Dissect is another type of filter available to be used in Logstash. However, according to Elastic documents, “Grok is a better choice when the structure of the text varies from line to line” [25]. This variation is exactly the case with CAS logs. Grok is essentially a pattern-matching method against regular expression that consistently maps certain parts of the data into dedicated fields [25]. Once the mapping is done, custom labels can be given to these mappings and additional

actions can be performed. A large number of prebuilt Grok patterns are already available for a range of logs, such as Syslog, HyperText Transfer Protocol Daemon (HTTPD), MongoDB, Amazon Web Services (AWS), etc. Furthermore, due to its open-source nature, Logstash lets users create their own filter patterns that can accomplish any new, unique tasks.

Grok pattern uses the `%{SYNTAX:SEMANTIC}` syntax [25] – `SYNTAX` is the character sequence to be matched and `SEMANTIC` is the custom label one could use for that specific matching. For example, `10/03/2020` could be mapped to the `DATE` pattern and `8.8.8.8` could be mapped to the `IP` pattern. As for the labels for these mappings, they could be any custom name such as, `event_date`, `server_ip` respectively.

3.4. Elasticsearch

Elasticsearch is a distributed search and analytics engine and sits at the core of the entire Elastic Stack. Elasticsearch is built on top of Apache Lucene, an open-source search engine software library [27]. The top-down hierarchy of Elasticsearch starts with indices which are logical partitions, and each index holds documents [27]. There can be multiple indices in each node (server). However, it is important to note that performance and number of indices have an inverse relationship. Documents are "JavaScript Object Notation (JSON) objects, and the base unit of storage" in Elasticsearch [28]. Indexed documents are independent from each other due to denormalization. This means each one of them has all the information (keys and values) necessary to return a result and is not reliant on any other documents. Within documents, data is stored in the form of key value pairs [28], where key is the name of the field and value can be any type of data such as strings, number, Boolean, etc. The number of documents in indices is only limited to the physical size of the server.

Elasticsearch is able to split its indices into shards [28] - horizontal partitions of data. These shards can reside on a single server or multiple nodes to balance load and boost performance. Since indices are stored in shards, Elasticsearch is easily scalable [28] horizontally and adding more nodes becomes easier. Each shard can have multiple replicas, thus ensuring redundancy to mitigate issues related to hardware failures [27].

Elasticsearch generally falls under the category of a Not Only Structured Query Language (NoSQL) database as opposed to a relational database management system (RDBMS). While it may be difficult to directly compare and contrast each aspect of both Elasticsearch and RDBMS, there are some notable differences between the two. Elasticsearch does not do transactions in the traditional way and does not have the ability to rollback any submitted documents [29]. It does, however, use write-ahead-logs to prevent data loss [29].

Elasticsearch does not require a schema or mapping in advance and will do some automatic guessing [29] during document ingestion into indices. Dynamic mapping is the ability to assign field types automatically when previously unknown fields are encountered in a document [30]. From the work performed for this project, it has become clearer that specifying mappings is far more advantageous and should be done for better outcomes. Even though mappings are optional, users who know their data well, are highly encouraged to consider explicit mappings for their data.

Elasticsearch is a document-oriented database and for objects to become searchable, they need to be indexed first. In RDBMS, data is typically split into columns by normalizing tables [31]. Unlike RDBMS, every document gets denormalized by Elasticsearch prior to indexing [29]. Due to denormalization, there is no need for expensive joining and hence retrieval performance is optimized. There are a few drawbacks to denormalization – increased space consumption due to duplicate storing and, more importantly, updating becomes inefficient [29]. Due to

denormalization, continuous updating to documents gets harder because changes must be applied to all the instances where the object in question resides. There is an argument to be made that Elasticsearch is better suited for write-once, read-multiple-times, types of operations [29].

Elasticsearch is designed for tasks involving search and retrieval efficiency [29], such as full-text search, fuzzy queries, and autocomplete tasks. However, it may not be the appropriate choice as the primary database and record keeper. In this project, Elasticsearch is a sound choice for the purpose of dealing with logs because log contents rarely change. Logs are written once and read many times. Once data is indexed in Elasticsearch, users can run complex queries against their data and use aggregations to retrieve complex summaries. The primary way of interacting with Elasticsearch is via REST Application Programming Interfaces (APIs) [32]. HTTP methods `get`, `post`, `put`, `head`, and `delete` are supported [33]. Elasticsearch has its own Domain Specific Language [34] (DSL) based on JSON to define queries. It also supports Structured Query Language (SQL).

3.5. Kibana

The final building block of Elastic Stack in the log management pipeline is Kibana. It is a web-based, open-source analytics, and visualization tool for data stored in Elasticsearch indices. Using a command line tool such as cURL to query Elasticsearch may not be the optimal way to interact with indexed data. Kibana allows users to visualize the results in much better ways by creating custom dashboards, histograms, graphs, charts, and so on. It is also capable of performing more complex tasks, including geo mapping, location analysis and time series analysis. Kibana's rich graphical and visualization capabilities help effectively communicate and share complex data with a wider audience, including non-technical stakeholders.

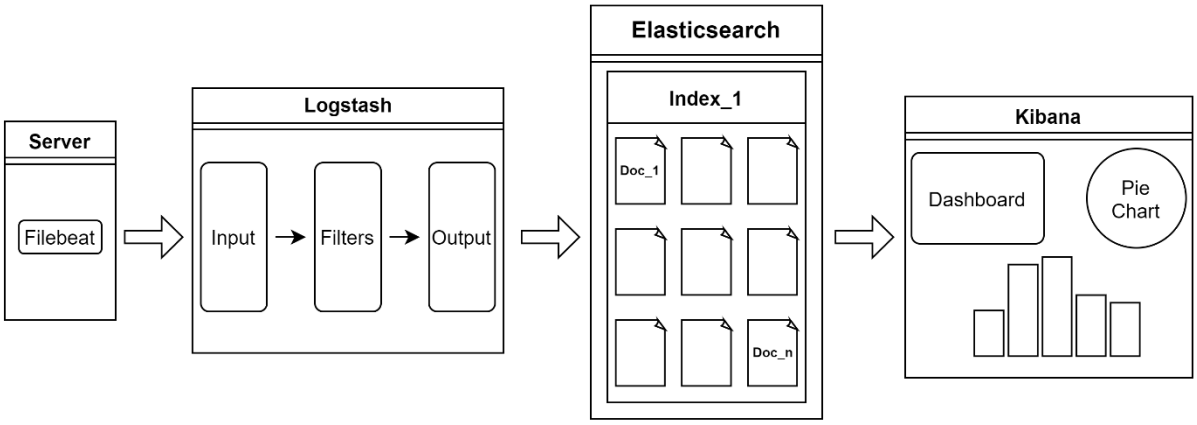

Figure 4: Elastic Stack Pipeline

4. IMPLEMENTATION

4.1. Data Set

For the purpose of initial testing, understanding CAS events in each block and developing the Grok filter, log data belonging to a single user's login process were used. These logs would be called *trial data*. The *trial data* was obtained using a development CAS server with very minimal network traffic. This almost-isolated environment helped obtain data belonging to a single test user's login process, with no interruption from other users. The steps were captured and documented as the test user simulated a normal login attempt. This also provided an opportunity to observe each user action and simultaneously its corresponding log event.

Later on, NDSU CAS logs generated between June 1, 2020 and September 16, 2020 were used for the actual experiment and analysis. These logs would be called *sample data*. This data set comes from the CAS production server and it captures real login attempts by students, staff, and faculty at NDSU. Nearly fourteen weeks' worth of logs is a reasonable sample of data to explore, evaluate, and draw conclusions.

The first round of implementation was performed strictly using the *trial data* for the sake of simplicity. Once that initial test was accomplished, the actual *sample data* was collected, transformed, and indexed for analysis and evaluation purposes.

4.2. Elastic Stack Architecture and Setup

The Elastic Stack design described in this section is a proof-of-concept deployment. In order to deploy at a production level, this setup will require further adjustments and more resources that are, however, outside of the scope of this project.

4.2.1. Filebeat

As a client-side program, Filebeat does not require its own hardware and simply resides on the server it collects logs from. In this case, Filebeat gets installed on the CAS server, which is a Red Hat Enterprise Linux (RHEL) system. Filebeat can be directly downloaded from the Elastic website in the form of an RPM file. Figure 5 shows the commands involved in Filebeat setup [35].

```
curl -L -O https://artifacts.elastic.co/downloads/beats/filebeat/\
filebeat-7.6.0-x86_64.rpm

sudo rpm -vi filebeat-7.6.0-x86_64.rpm

sudo /etc/init.d/filebeat stop
```

(1)

(2)

Figure 5: Filebeat Setup

1) Download the Filebeat RPM to the client from Elastic repository
2) Install the RPM and make sure it is not running at this point

4.2.2. Elasticsearch, Kibana, and Logstash

While it is possible to host the rest of the Elastic Stack components on the same (CAS production) server as Filebeat, it is highly recommended to deploy them on a dedicated server hardware. This is to avoid any performance issues that may arise on the CAS server that is in live-production mode serving NDSU users. Considering the experimental nature of this deployment and its small scale, a computer with modest resources and running Ubuntu operating system was selected to host the remaining three components. Figure 6 shows all the commands involved in the setup of Elasticsearch, Kibana, and Logstash [35].

```
# wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch \
| sudo apt-key add –

# sudo apt-get install apt-transport-https

# echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" \
| sudo tee -a /etc/apt/sources.list.d/elastic-7.x.list

# sudo apt-get update && sudo apt-get install elasticsearch
# sudo apt-get update && sudo apt-get install kibana
# sudo apt-get update && sudo apt-get install logstash

# sudo systemctl stop elasticsearch kibana logstash
```

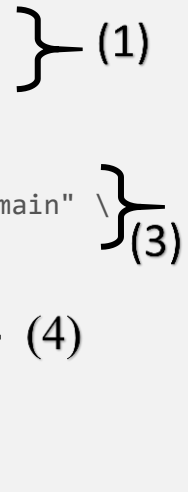

Figure 6: Elasticsearch, Kibana and Logstash Setup

1) Download and install the Elastic Public Signing Key
2) Install the prerequisite `transport-https` package
3) Save the repository definition to the host
4) Install Elastic Search, Kibana, and Logstash packages, respectively
5) Make sure all three services are stopped at this point

4.2.3. Nginx

Since Kibana is usually only accessible on the localhost, Nginx, a software-based reverse proxy server, was set up on the same computer. This allows for Kibana to be accessible via web browsers, providing flexible connection from anywhere. A reverse proxy is a software or hardware-based server that acts as an intermediary between a server and different clients [36]. User requests would be forwarded to Kibana through this reverse proxy and the results would also be returned back to the user through Nginx. This also allows for a dedicated user to be set up with a password so that the access to Kibana web page is secure and limited to that specific user. Figure 7 shows the commands used to set up the Nginx application [37].

```
sudo apt-get update && sudo apt-get install nginx
echo "USER:`openssl passwd -apr1`" | sudo tee -a /etc/nginx/htpasswd.users
sudo vi /etc/nginx/sites-available/site
sudo ln -s /etc/nginx/sites-available/site /etc/nginx/sites-enabled/site
sudo systemctl restart nginx
sudo ufw allow 'Nginx Full'
```

(1) (2) (3) (4) (5) (6)

Figure 7: Nginx Proxy Server Setup

1) Install Nginx
2) Create the `USER` with a password
3) Create Nginx server block file - Figure 8 shows the contents of this server block file [37]

```
server {
    listen 80;

    server_name XXX.XXX.XXX.XXX;

    auth_basic "Restricted Access";
    auth_basic_user_file /etc/nginx/htpasswd.users;

    location / {
        proxy_pass http://localhost:5601;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

Figure 8: Nginx Server Block File

4) Enable the new configuration by creating a symbolic link
5) Restart Nginx
6) Adjust firewall rules using Uncomplicated Firewall (UFW) to allow connections to Nginx

   – UFW is the default firewall configuration tool for Ubuntu operating system.

After this setup, Kibana would be accessible via the IP address of Elastic Stack host at:

`http://server_ip/status`

4.3. Log Collection

Now that all the required elements are in place, the log ingestion process can begin. The first step is to make any appropriate configuration changes to Filebeat so that it can collect CAS logs and forward them to Logstash hosted on another computer. On a RHEL client as is the case in this project, the Filebeat directory is located at `/etc/filebeat` and the configuration file is called `filebeat.yml`.

As shown in Figure 3 earlier, CAS logs are written in a multiline format. Multiline event handling should be taken care of prior to the logs reaching Logstash to avoid data corruption later. To handle multiline events properly, the `multiline` settings [38] in the configuration file need to be set up in order to specify which lines are part of a single event or block – in other words, the beginning and ending of each block should be specified. Each block of CAS log starts with the `YYYY-MM-DD` date format. Therefore, that specific date format is the beginning of the line and also the pattern that had to be matched to turn multiline format log into a single line entry. The regular expression `'^[0-9]{4}-[0-9]{2}-[0-9]{2}'` matches the date format at the beginning of each block of CAS logs. The first part represents the 4-digit year and each of the digits could be any number between 0 and 9. The second and third parts represent the 2-digit month and 2-digit day, respectively. Again, the 2-digits in month and day could range from 0 to 9. Figure 9 shows the Filebeat configuration file.

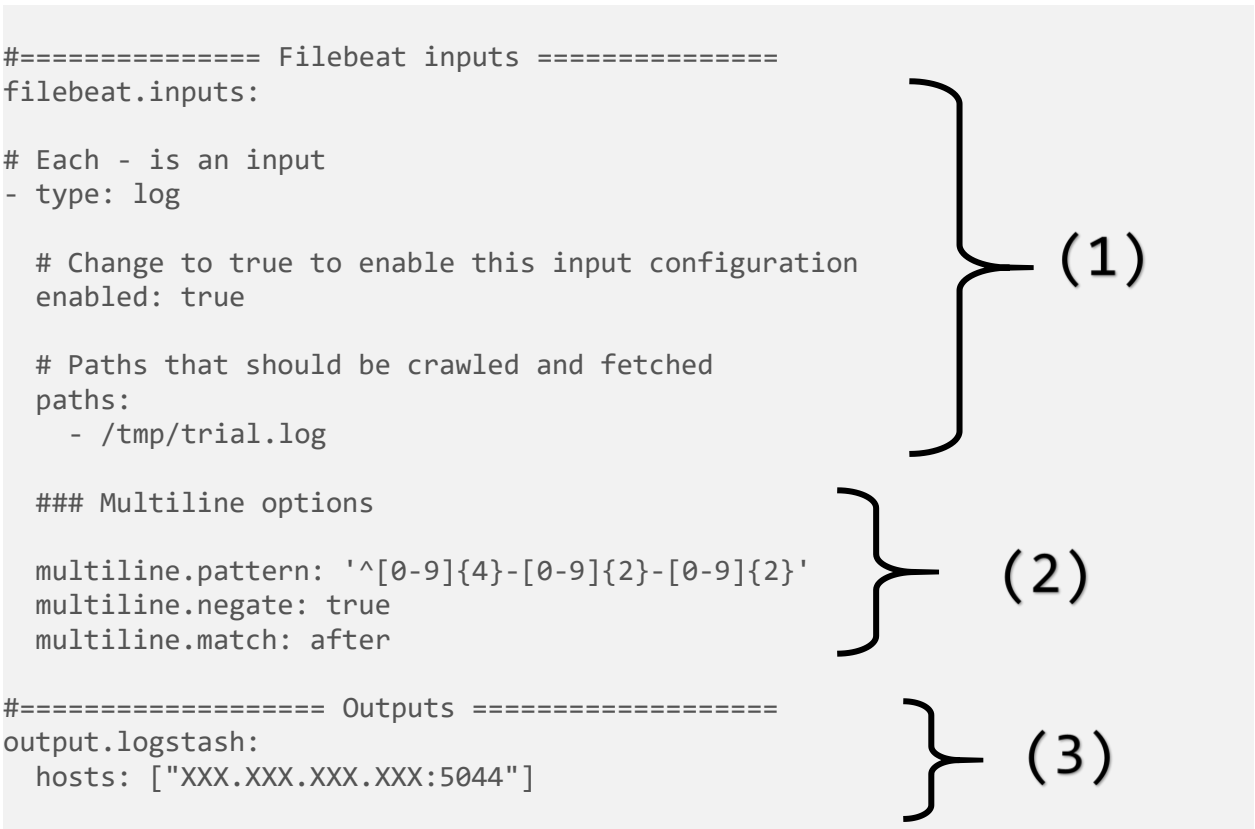

Figure 9: Filebeat.yml Configuration

1) Specify the input and its path
2) Treat all the lines that do not start with `YYYY-MM-DD` pattern as part of the last line that does in each block
3) Finally, forward the logs to the specified Logstash host IP's port `5044`

4.4. Log Enrichment

Once the CAS logs reach Logstash, the transformation can begin. Logstash is running on an Ubuntu host, and its configuration directory is located at `/etc/logstash.` Logstash allows for multiple pipelines to transform the data. Each of these will have a corresponding configuration file that defines it. The first pipeline's output would become the second pipeline's input and so on and so forth. All the pipelines used would be listed in the `pipeline.yml` file.

For simplicity reasons, instead of breaking the log transformation process into multiple phases, all the transformation tasks were combined into a single pipeline in this project. The contents of the `pipeline.yml` as follows:

```
# list of pipelines and corresponding config file paths
- pipeline.id: main
  path.config: "/etc/logstash/conf.d/01.conf"
```

Figure 10: Logstash Pipeline.yml Configuration

The next task was to create the `/etc/logstash/conf.d/01.conf` file that will shape the pipeline that the CAS logs will go through. As previously stated in Chapter 3, Logstash is made up of three parts – input, filters, and output - and this `/etc/logstash/conf.d/01.conf` configuration file would specify all those parts.

One of the biggest hurdles in the log transformation process was creating the custom Grok filter pattern. Unlike other logs, there is no pre-built pattern available for CAS logs from the Elastic company or the wider open-source community. To tackle this, it is important to remember what CAS logs are. CAS logs contain step by step user login information, but they are not ordered by users. CAS logs are ordered by time. Between the time a user starts and finishes their authentication, there could be multiple login steps belonging to other users captured in the logs. By looking at any random portion of CAS log from any given time, one cannot easily pinpoint where a user's login attempt begins and ends. As stated in previous chapters, this is precisely why CAS logs are not user friendly.

First step in creating the custom Grok pattern was to determine what sort of character sequences needed to be matched. The *trial data* was instrumental in figuring out all the steps involved in a single user's login process. As shown in Figure 3 earlier, each block of CAS log represents a different background actions during the login process. The header of each block starts

with the date and timestamp of the event. Beyond the header, there are seven remaining lines in every block. These seven lines begin with key words - `WHO`, `WHAT`, `ACTION`, `APPLICATION`, `WHEN`, `CLIENT IP ADDRESS` and `SERVER IP ADDRESS`. The names of these keys and their order remain consistent throughout the process in each block. Although the names of keys remain consistent, `WHAT` and `ACTION` keys have different values and those values' character sequences change from block to block. These inconsistencies in values would mean that the regular expression for each line would differ depending on the block the value is in.

As explained in Section 3.3, keys in each line in every block of CAS log was matched with the appropriate regular expression pattern [26] based on its value. Special attention was required for `WHAT` key because its values were changing in each block. Since the goal was to create a common pattern that would match line items in each and every block, `OR (|)` logic had to be used multiple times to account for different character sequences in `WHAT` key's values.

Further examination of the *trial data* revealed that there were many redundant details present. For example, the header of each block and the fifth key (`WHEN`) represented the same information – date and the time of the event. Hence, the `WHEN` key in each block could be ignored and the header was enough to derive the date and time details. Also, the fourth line `APPLICATION: CAS` was a constant throughout the process. This information was deemed insignificant and was marked to be filtered out from the final output. Figures 11 and 12 below show the custom Grok filter and the resulting output when that filter is applied, respectively.

```
(?m)%{NOTSPACE:date} %{TIME:time}.*
WHO: %{NOTSPACE:user_name}
WHAT: (\[|.*\[|\[result=|%{SPACE}*)%{DATA:what}(\(|\,|\-|\*| for).*
ACTION: %{NOTSPACE:action}
APPLICATION: %{WORD:application}
.*CLIENT IP ADDRESS: %{IP:source}
SERVER IP ADDRESS: %{IP:destination}
```

Figure 11: Custom Grok Filter Used in Logstash for Log Transformation

```
date: 2020-10-20
time: 04:07:32,710
user_name: jane.doe
what: UsernamePasswordCredential
action: AUTHENTICATION_SUCCESS
application: CAS
source: 10.10.10.10
destination: 20.20.20.20
```

Figure 12: Sample List of Key-Value Pairs after Log Transformation

Observing the filtered results from the *trial data* was vital in understanding what exactly would be indexed in Elasticsearch. This gave an opportunity to further beautify the logs by filtering out several pieces of metadata present in the logs that are irrelevant to this particular log analysis task. After many trial and errors, extra code was added in the Logstash filter section to make more adjustments to the data that would eventually be sent to Elasticsearch for indexing. Figure 13 shows the complete Logstash filter section created for this project.

```
input {

  beats {
    port => 5044
  }
}

filter {

  grok {
match =>  { "message" => "(?m)%{NOTSPACE:date} %{TIME:time}.*
WHO: %{NOTSPACE:user_name}
WHAT: (\[|.*\[|\[result=|%{SPACE}*)%{DATA:what}(\(|\,|\-|\*| for).*
ACTION: %{NOTSPACE:action}
APPLICATION: %{WORD:application}
.*CLIENT IP ADDRESS: %{IP:source}
SERVER IP ADDRESS: %{IP:destination}" }
    }

    mutate { replace => { "time" => "%{date} %{time}" } }

    date {
     match  => ["time", "yyyy-MM-dd HH:mm:ss,SSS"]
     target => "time"
    }

    if "beats_input_codec_plain_applied" in [tags] {
        mutate { remove_field => [ "tags" ] }
    }
    if "result=Service Access Granted" in [what] {
       mutate { update => { "what" => "SERVICE_ACCESS_GRANTED" }
       }
    }
    if "event=mfa-duo" in [what] {
       mutate { update => { "what" => "MFA_DUO" }
       }
    }
    if "event=success" in [what] {
       mutate { update => { "what" => "EVENT_SUCCESS" }
       }
    }
    if "SAML2_RESPONSE_CREATED" in [action] {
       mutate { update => { "what" => "ISSUER_URL" }
       }
    }
    if "SAML2_REQUEST_CREATED" in [action] {
       mutate { update => { "what" => "ISSUER_URL" }
       }
    }
    mutate { remove_field => ["date", "message", "application", "path", "host", "agent", "ecs",
"input", "log"] }

}

output {

  elasticsearch {
    hosts => ["localhost:9200"]
    index => "cas_logs"
  }
}
```

(1) (2) (3) (4)

Figure 13: Complete Logstash Filter

1) Listen on port `5044` for incoming logs via Filebeat
2) Transform the incoming CAS logs using the custom Grok filter
3) Further enhance the logs by removing and renaming keys and standardizing values
4) Send enhanced logs to an index called `cas_logs` in Elasticsearch for storage and analysis

Note that Elastic Search is running on the same host as Logstash and listens on port `9200`.

The index `cas_logs` had to be manually created in order to forward logs to it.

4.5. Log Storage and Indexing

The next phase is setting up Elastic Search so that it can receive the logs from Logstash. The default configurations done by Elastic Search itself during its installation are sufficient to get started. These configuration files reside under the `/etc/elasticsearch` directory.

As stated in the last section, an Elastic Search index had to be created before sending data to it. This can be done via the Kibana web interface that was set up in Section 4.2.3. Kibana has a built in Console section under the Dev Tools where users can run queries to interact with Elasticsearch. Even though mappings are optional in Elastic Search, after some trial and error, it was apparent that letting Elastic Search does the mappings automatically would lead to some erroneous results. Therefore, along with creating the `cas_logs` index, explicit mappings were done to the data.

To create an index with mappings, the HTTP method `put` is used followed by the name of the desired index. Figure 14 below shows the list of keys and what data types their values should be mapped to in Elasticsearch. This takes away any guessing Elasticsearch needs to do regarding the types of incoming data.

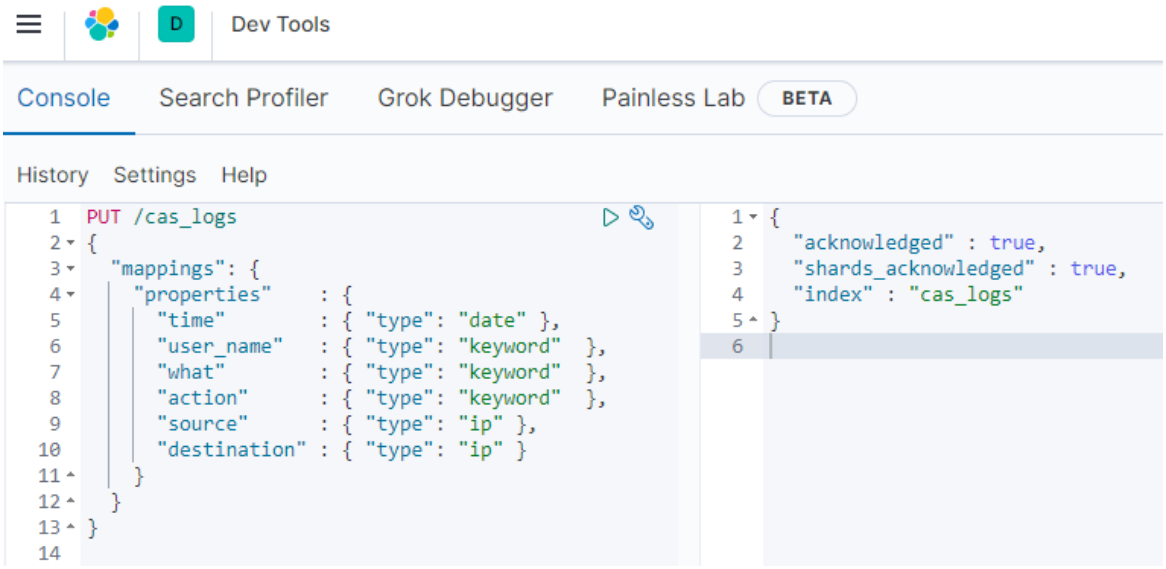

Figure 14: Elasticsearch Index Creation and Explicit Mapping

After the index creation, *trial data* was ingested by starting the services Elasticsearch, Kibana, Logstash, and Filebeat, respectively. Figure 15 below lists the commands used to start the services.

```
# sudo systemctl start elasticsearch
# sudo systemctl start kibana
# sudo systemctl start logstash

# sudo /etc/init.d/filebeat start
```

(1)

(2)

Figure 15: Elastic Stack Startup Commands

5. EVALUATION

After the initial experiment with *trial data* was completed, a new identical index called `logstash` was created in Elasticsearch and the actual *sample data* was ingested into it for analysis and evaluation. CAS logs generated over a span of fourteen weeks can now be queried and analyzed. There are multiple ways to query the data indexed in Elasticsearch. Users have the option of using Kibana built-in tools or using multiple other supported programming language clients to query the data.

5.1. Kibana Built-in Tools

Kibana Discover tool offers built-in filters and is a starting point to explore the indexed data. Kibana Discover queries are built on Kibana Query Language [39] (KQL). In order to explore the data via Discover, however, an index pattern has to be in place for that specific index in question. According to Elastic documentation, index patterns [40] allow users to combine different data sources together so their shared fields may be queried in Kibana. An index pattern can be set up by visiting the Kibana web interface that was set up in Section 4.2.3 and navigating to the Management section. In this case, an index pattern called `logsstash*` was created to match the index `logstash`.

As Figure 16 shows, *sample logs* collected over fourteen weeks resulted in 3,764,768 documents in Elasticsearch. This would correlate to 3,764,768 blocks of entries in the CAS logs. Of course, this does not equal to the number of user login attempts during that time because of the fact that each login attempt has multiple blocks. This Discover view allows for zeroing in on a particular timeline from the graph by selecting any area.

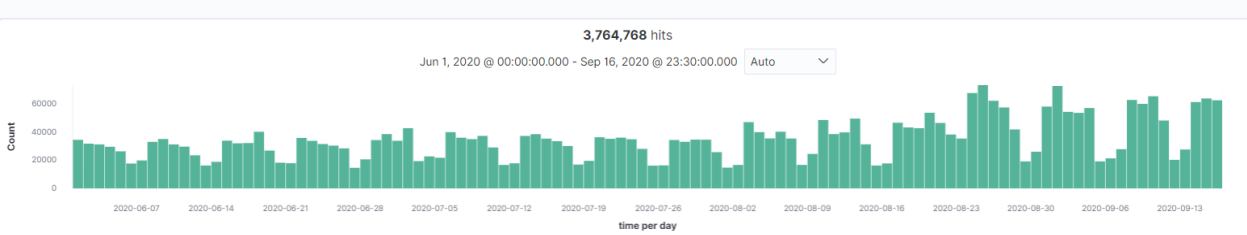

Figure 16: Indexed Sample Data Count in Elasticsearch

Figure 17 shows Kibana Discover view with structured and simplified CAS logs in key-value pairs format. Its search field is very user-oriented and capable of providing auto suggestions as queries are typed. Using Kibana's built-in filters and its automatic suggestions, simple queries can be performed with ease. Instead of going through multiple pages of CAS logs, all the necessary information is a few clicks away. In this example, the query is to find a specific user's failed username-password attempts. The search returns two instances.

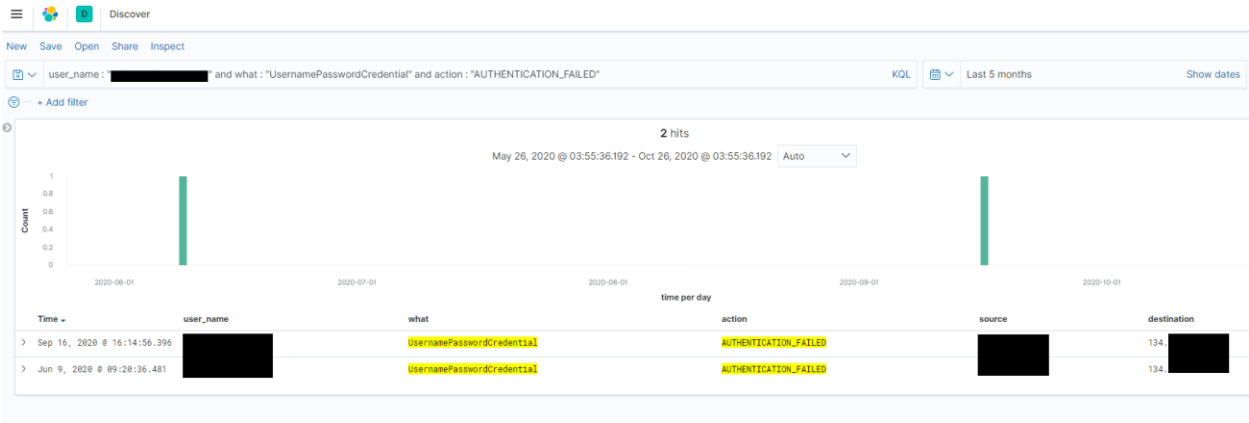

Figure 17: CAS Logs in Simplified Form

Kibana's built-in Console [41] offers a way to run complex queries on the indexed data using HTTP methods. The syntax of a query starts with the appropriate HTTP method and is followed by the index name and Elasticsearch command. Similar to Discover, Console tool also has the ability to suggest or auto-complete commands as they are typed.

This paper is by no means a comprehensive guide on all the Elasticsearch query possibilities. Search options are exhaustive and beyond the scope of the project. Elasticsearch

supports a wide variety of functions, including Boolean operators, ranges, wildcards, fuzzy queries, and so on [34]. For example, fuzzy searches will return results matching the search terms even if they have typos. Figure 18 shows a fuzzy search with typos in the `user_name` value. The query still returns the correct `user_name`.

Figure 18: Sample Fuzzy Query

5.2. External Python Program

In order to perform more complex analysis of CAS logs using Elasticsearch, it is important to have some domain specific knowledge about the data being dealt with in this project. Referring back to the study done on the *trial data* in Section 4.4, it was already established that each block of CAS logs corresponds to a background event during authentication. It was also observed that throughout the login process, the values of two keys `WHAT` and `ACTION` captured the context of each block. The values of each of those keys, of course, differed from block to block. Table 1 below documents the values of keys `WHAT` and `ACTION` in each block of CAS logs.

Table 1: CAS Log Events for a Single Successful User Authentication

BLOCK	WHAT	ACTION
1	Event	AUTHENTICATION_EVENT_TRIGGERED
2	**UsernamePasswordCredential**	**AUTHENTICATION_SUCCESS**
3	Service Access Granted	SERVICE_ACCESS_ENFORCEMENT_TRIGGERED
4	MFA-Duo	AUTHENTICATION_EVENT_TRIGGERED
5	MFA-Duo	AUTHENTICATION_EVENT_TRIGGERED
6	**DuoCredential**	**AUTHENTICATION_SUCCESS**
7	Event	AUTHENTICATION_EVENT_TRIGGERED
8	Service Access Granted	SERVICE_ACCESS_ENFORCEMENT_TRIGGERED
9	TGT	TICKET_GRANTING_TICKET_CREATED
10	Service Access Granted	SERVICE_ACCESS_ENFORCEMENT_TRIGGERED
11	ST	SERVICE_TICKET_CREATED
12	Service Access Granted	SERVICE_ACCESS_ENFORCEMENT_TRIGGERED
13	ST	SERVICE_TICKET_VALIDATED

A closer examination of the table revealed that out of all blocks, there are two that refer to credentials and have potentially more valuable information for further analysis. Rows 2 and 6 involve external user inputs (username-password and Duo credentials); the rest correspond to tasks performed by the CAS system itself. Row 2 represents the users entering their credentials and row 6 refers to users completing their multi-factor verification.

This insight into CAS logs could be used to develop more complex queries against data indexed in Elasticsearch that could help strengthen cybersecurity. For example, for the purpose of identifying suspicious activities related to login attempts, it is not necessary to take all the blocks of CAS logs into consideration. Focusing on just two blocks of logs with `WHAT` key values `UsernamePasswordCredential` and `DuoCredential` would be sufficient to analyze the outcome of user authentication. If both of those blocks also contain `ACTION` keys with values `AUTHENTICATION_SUCCESS`, then that would indicate the user successfully provided both

username-password and multi-factor. The rest of the blocks in CAS logs can be safely ignored because they are tied to CAS system's internal workings, not external user inputs.

How can this domain knowledge be applied to help cybersecurity? One approach would be to compare the number of times a user provides the correct username-password and the number of times the same user successfully negotiates multi-factor. The hypothesis is that if the difference is significantly higher, then a reasonable conclusion could be made - perhaps the user's password is compromised and bad actors are in possession of the first factor (password), but they do not have access to the second factor (for example: phone). Further analysis can be done by checking whether the source IP address of these unusually high failed multi-factor attempts are also abnormal for that user. By extracting these anomalies, compromised passwords related threats can be weeded out in order to bolster security.

In order to test this hypothesis, one can take advantage of Elasticsearch's aggregate function that puts results into separate buckets for comparison. Elasticsearch supports other programming language clients that can interact with the indexed data. Utilizing these capabilities, a protype Python program was developed that would create aggregations at multiple levels. The top-level aggregation bucket would be based on `user_name`. The second level aggregation would be for each `source` IP that the user attempted to login from. The third and final level would have two separate aggregation buckets; one for the number of times the user successfully entered the username-password and one for the number of times the user successfully negotiated multi-factor. Figure 19 visualizes the program's three levels of aggregation logic for easier understanding.

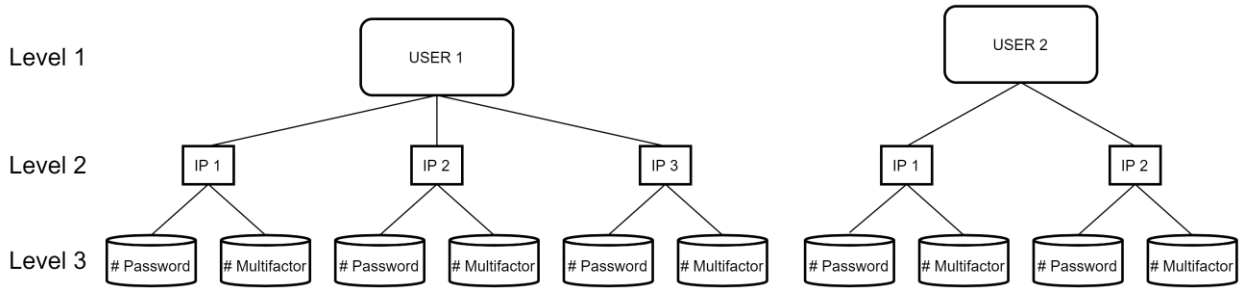

Figure 19: Visual Representation of Python Program with Aggregate Functions

Subsequently, the two aggregation buckets at the third level will be compared. In a certain predefined time period, if the difference in numbers between the two buckets happen to be more than an acceptable threshold for that specific `source` IP, further action can be taken regarding that user. Figure 20 documents the Python program's pseudocode.

```
for each “user_name” in documents indexed in Elasticsearch
      print (user_name)

      for each “source” IP
            print (source_ip)

            filter “action” by “AUTHENTICATION_SUCCESS”

                  count the number of “UsernamePasswordCredential”
                        set the password count to X

                  count the number of “DuoCredential”
                        set the multifactor count to Y

if (X-Y) == 0
      output “Looks normal”

elif (X-Y) > some_threshold
      output “Something is wrong!!!”
```

Figure 20: Pseudocode for the Python Program

Figure 21 shows a portion of the Python program's output when the test was run with the threshold number of ten, meaning if the difference between `UsernamePasswordCredential` and `DuoCredential` is greater than ten, a warning sign would be printed for that user, listing the

corresponding `source` IP. In this instance, the number of `source` IP user attempted to login from is just one. From that `source` IP, they successfully entered their username-password 331 times, whereas multi-factor was submitted only 320 times. Due to the prototype nature of the program, it currently lists only the first ten timestamps of both username- password and multi-factor entries.

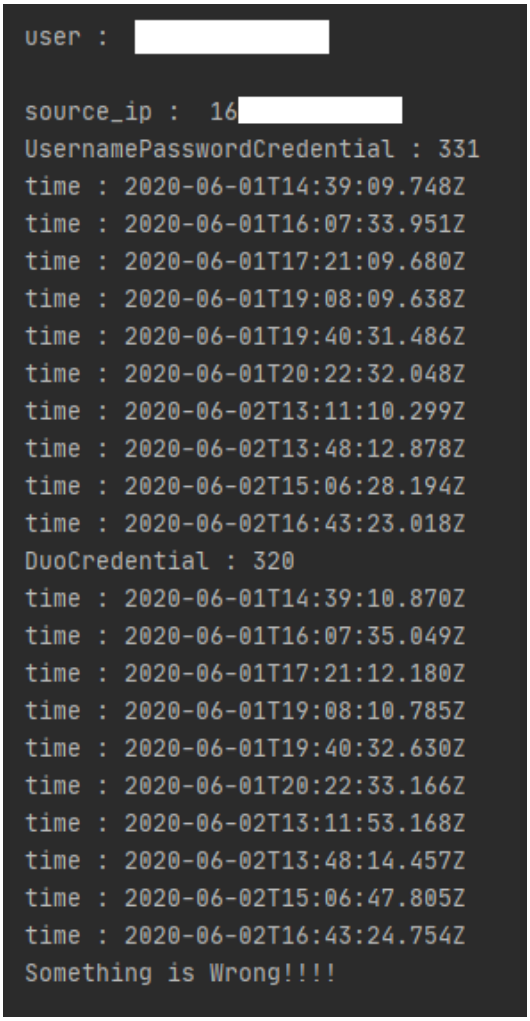

Figure 21: Sample Output of Python Program with the Threshold of Ten

Consider a hypothetical scenario where a user's password is compromised due to some data breach. Also assume that this is a shared credential among other accounts, including services provided at NDSU. If bad actors are not in possession of the user's second factor, such as a phone, the user's password alone cannot be used to exploit NDSU systems that are protected by multi-

factor. However, their attempts would be captured in CAS logs. Today's cyberattacks are highly automated and take place with little to no human intervention. So, it is possible for these bad actors to stay below the velocity limits set by the services and avoid lockdowns while the malicious attempts continue. Nevertheless, when Elastic Stack simplifies and analyzes CAS logs in near real time, it can alert to the significant difference between the number of successful password entries and failed multi-factor submissions.

It is imperative that the user gets notified about the compromised password as soon as possible. This could help the user avoid the same credential being used against any other services without multi-factor protection outside of NDSU. Therefore, this solution is not limited to protecting NDSU services, but could also potentially help NDSU community safeguard or mitigate attacks against their non-NDSU accounts and services. This fits into NDSU's broader context of protecting users and resources from cyberthreats. This prototype could be an additional asset to IT Security in their goal of enhancing cybersecurity.

6. CONCLUSION

6.1. Summary

In this paper, a pilot project of deploying Elastic Stack was proposed to process, transform, and analyze underutilized CAS logs in order to identify digital threats and bolster cybersecurity. The paper explained the types of cyberthreats faced today, and the pitfalls they can create.

The CAS system infrastructure and other elements of the problem domain were also discussed. Given the significant role authentication logs play in the cybersecurity field, it was critical to maximize the benefits gleaned from the wealth of information CAS logs contain. All the components of the proposed Elastic Stack solution were detailed along with how each of them would help fix the issues documented in the problem statement.

The paper also described why and how the custom Grok pattern was developed in order to implement the solution. Kibana visualization tool's effectiveness was documented to show how easy it was to consume CAS data after it was structured. Finally, the Elasticsearch component was evaluated for its effectiveness. An external Python program was developed to explore how transformed CAS logs along with Elasticsearch's aggregate function could potentially assist in mitigating malicious attacks. It was determined that Elasticsearch along with the proof-of-concept Python program have the potential to collect insights into user authentication patterns and identify threats.

6.2. Future Considerations

Even though it was shown that Elastic Stack is a potential tool to gather intelligence from CAS authentication logs, there are still ways to improve upon what was accomplished in this project. The Python based program was proven to effectively utilize indexed data and Elasticsearch aggregate function, but it still remains a prototype and requires human intervention to react to

potential threats. It can be further developed to automate the alerts and speed up the reaction time. A mechanism that can provide real-time security alerts generated by analyzing CAS logs using Elastic Stack would be a vital upgrade. Another area of focus could be taking advantage of Elasticsearch's Artificial Intelligence (AI) capabilities. A goal would be to build a smart detection system that would map a profile for each user from their historic login data and would alert if the patterns stray from what is considered to be normal behavior. Utilizing the rich visual capabilities of Kibana would be another way to help IT Security. Paired with the AI system, a dashboard set up to display approximate geo locations of users' login origin could help narrow down anomalies quicker.

If all these considerations were accomplished, the Elastic Stack solution can be packaged as a product and deployed at NDSU. This fully featured product could serve as an additional defensive tool in tackling cyberthreats, thus helping the NDSU community.

REFERENCES

[1] Cisco, “What Is Cybersecurity?,” Cisco. https://www.cisco.com/c/en/us/products/security/what-is-cybersecurity.html (accessed Aug. 14, 2020).

[2] A. Tugce, “How Log Monitoring Helps Cyber Security?,” Logsign, Aug. 28, 2019. https://blog.logsign.com/how-log-monitoring-helps-cyber-security/ (accessed Aug. 14, 2020).

[3] K. Kent and M. Souppaya, “Guide to Computer Security Log Management,” National Institute of Standards and Technology, Sep. 2006. Accessed: Aug. 18, 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf.

[4] New Jersey Institute of Technology, “What is Cryptojacking? How to prevent, detect, and recover from it,” NJIT. https://ist.njit.edu/what-cryptojacking-how-prevent-detect-and-recover-it (accessed Aug. 19, 2020).

[5] R. Kress, “Why humans are still security’s weakest link,” Accenture, May 08, 2019. https://www.accenture.com/us-en/blogs/security/humans-still-securitys-weakest-link (accessed Aug. 18, 2020).

[6] D. Mazurek, “CAS - CAS Protocol Specification,” Apereo.github, Dec. 01, 2017. https://apereo.github.io/cas/6.1.x/protocol/CAS-Protocol-Specification.html (accessed Aug. 24, 2020).

[7] Apereo Foundation, “CAS - Architecture,” Apereo.github. https://apereo.github.io/cas/6.2.x/planning/Architecture.html (accessed Aug. 24, 2020).

[8] M. Cobb, “Single sign-on (SSO) authentication can help prevent password fatigue,” Computerweekly, Jan. 28, 2010. https://www.computerweekly.com/tip/Single-sign-on-SSO-authentication-can-help-prevent-password-fatigue (accessed Aug. 24, 2020).

[9] Apereo Foundation, “About CAS,” Apereo. https://www.apereo.org/projects/cas (accessed Aug. 24, 2020).

[10] Apereo Foundation, “CAS Enterprise Single Sign-On,” Apereo.github. https://apereo.github.io/cas/5.3.x/index.html (accessed Aug. 25, 2020).

[11] Apereo Foundation, “CAS - CAS Protocol,” Apereo.github. https://apereo.github.io/cas/4.2.x/protocol/CAS-Protocol.html (accessed Aug. 25, 2020).

[12] National Institute of Standards and Technology, “Back to basics: Multi-factor authentication (MFA),” NIST, Dec. 09, 2019. https://www.nist.gov/itl/applied-cybersecurity/tig/back-basics-multi-factor-authentication (accessed Aug. 29, 2020).

[13] A. Greenberg, “So Hey You Should Stop Using Texts for Two-Factor Authentication,” Wired, Jun. 26, 2016. https://www.wired.com/2016/06/hey-stop-using-texts-two-factor-authentication/ (accessed Aug. 29, 2020).

[14] N. Carstensen, “Why is Log Management Important?,” graylog.org, Nov. 15, 2018. https://www.graylog.org/post/why-is-log-management-important (accessed Nov. 10, 2020).

[15] A. Chuvakin, K. Schmidt, and C. Phillips, “Chapter 15 - Tools for Log Analysis and Collection,” in Logging and Log Management, Amsterdam: Syngress Media Inc, 2013, pp. 243–266.

[16] M. Probert, “Grep, awk and sed -three very useful command-line utilities,” Nov. 2016. Accessed: Nov. 10, 2020. [Online]. Available: https://www-users.york.ac.uk/~mijp1/teaching/2nd_year_Comp_Lab/guides/grep_awk_sed.pdf.

[17] Splunk, “Documentation - Splunk Documentation,” docs.splunk.com. https://docs.splunk.com/Documentation (accessed Nov. 10, 2020).

[18] L. Bitincka, A. Ganapathi, and S. Zhang, “Experiences with Workload Management in Splunk,” Proceedings of the 2012 workshop on Management of big data systems - MBDS ’12, 2012, Accessed: Nov. 10, 2020. [Online].

[19] J. Stearley, S. Corwell, and K. Lord, “Bridging the Gaps: Joining Information Sources with Splunk.” Accessed: Nov. 10, 2020. [Online]. Available: https://static.usenix.org/events/slaml10/tech/full_papers/Stearley.pdf.

[20] Splunk, “Splexicon: Eventprocessing - Splunk Documentation,” docs.splunk.com. https://docs.splunk.com/Splexicon:Eventprocessing (accessed Nov. 10, 2020).

[21] Splunk, “Configure custom fields at search time - Splunk Documentation,” docs.splunk.com. https://docs.splunk.com/Documentation/Splunk/8.1.0/Knowledge/Createandmaintainsearch-timefieldextractionsthroughconfigurationfiles (accessed Nov. 10, 2020).

[22] Mozilla, “Regular Expressions,” MDN Web Docs, Aug. 17, 2020. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions (accessed Sep. 07, 2020).

[23] Elastic, “Beats: Data Shippers for Elasticsearch,” Elastic. https://www.elastic.co/beats/ (accessed Sep. 02, 2020).

[24] Elastic, “Logstash: Collect, Parse, Transform Logs,” Elastic. https://www.elastic.co/logstash (accessed Sep. 02, 2020).

[25] Elastic, “Grok filter plugin,” Elastic. https://www.elastic.co/guide/en/logstash/7.9/plugins-filters-grok.html (accessed Sep. 06, 2020).

[26] GitHub contributors, “Logstash--patterns-core,” GitHub, Jul. 29, 2020. https://github.com/logstash-plugins/logstash-patterns-core/blob/master/patterns/grok-patterns (accessed Sep. 06, 2020).

[27] Wikipedia contributors, “Elasticsearch,” Wikipedia, Jun. 18, 2020. https://en.wikipedia.org/wiki/Elasticsearch (accessed Sep. 10, 2020).

[28] D. Berman, “10 Elasticsearch Concepts You Need to Learn,” Logz, Nov. 10, 2019. https://logz.io/blog/10-elasticsearch-concepts/ (accessed Sep. 10, 2020).

[29] A. Brasetvik, “Elasticsearch as a NoSQL Database,” Elastic, Sep. 15, 2013. https://www.elastic.co/blog/found-elasticsearch-as-nosql (accessed Sep. 13, 2020).

[30] Elastic, “Dynamic field mapping,” Elastic. https://www.elastic.co/guide/en/elasticsearch/reference/current/dynamic-field-mapping.html (accessed Sep. 13, 2020).

[31] Microsoft, “Description of the database normalization basics,” docs.microsoft.com, May 21, 2020. https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description (accessed Sep. 13, 2020).

[32] Elastic, “REST APIs,” www.elastic.co. https://www.elastic.co/guide/en/elasticsearch/reference/current/rest-apis.html (accessed Sep. 18, 2020).

[33] Elastic, “HTTP | Elasticsearch Reference [7.10] | Elastic,” www.elastic.co. https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-http.html (accessed Sep. 18, 2020).

[34] Elastic, “Query DSL,” Elastic. https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html (accessed Sep. 20, 2020).

[35] Elastic, “Installing the Elastic Stack,” Elastic. https://www.elastic.co/guide/en/elastic-stack/current/installing-elastic-stack.html (accessed Jul. 19, 2020).

[36] Nginx, “What is a Reverse Proxy Server?,” Nginx. https://www.nginx.com/resources/glossary/reverse-proxy-server/ (accessed Aug. 30, 2020).

[37] J. Ellingwood and V. Kalsin, “How to Install Elasticsearch, Logstash, and Kibana (Elastic Stack) on Ubuntu 18.04,” DigitalOcean, Nov. 06, 2018. https://www.digitalocean.com/community/tutorials/how-to-install-elasticsearch-logstash-and-kibana-elastic-stack-on-ubuntu-18-04 (accessed Aug. 20, 2020).

[38] Elastic, “Manage multiline messages,” Elastic. https://www.elastic.co/guide/en/beats/filebeat/current/multiline-examples.html (accessed Sep. 10, 2020).

[39] Elastic, “Kibana Query Language,” Elastic. https://www.elastic.co/guide/en/kibana/current/kuery-query.html (accessed Sep. 28, 2020).

[40] Elastic, “Create an index pattern,” Elastic. https://www.elastic.co/guide/en/kibana/current/index-patterns.html (accessed Sep. 28, 2020).

[41] Elastic, “Console,” Elastic. https://www.elastic.co/guide/en/kibana/current/console-kibana.html (accessed Oct. 03, 2020).