TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Yasin Aydin 177781IVSB

SAFEGUARDING SENSITIVE DATA IN LOGS

Bachelor's Thesis

Supervisor: Kaido Kikkas

Ph.D. in Engineering

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia Teaduskond

Yasin Aydin 177781IVSB

TUNDLIKE LOGIANDMETE KAITSE

Bakalaureusetöö

Juhendaja: Kaido Kikkas

Ph.D. in Engineering

Tallinn 2021

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, the literature and the work of others have been referenced. This thesis has not been presented for examination anywhere else.

Author: Yasin Aydin

2021-05-17

Abstract

With the new developments of national and international data protection standards such as GDPR, the importance of personal data safety and its awareness has been increased. Massive leaks by technology giants happening in last few years revealed insufficient practices on protecting sensitive data. It is often observed that the vulnerability of having unsanitized data in logs is usually overlooked.

This thesis aims to provide clear picture on the problem of having sensitive data in logs, by using some well-known weaknesses and recent data leaks, to suggest solutions to different levels of this multi-dimensional problem, analyze existing solutions and their limitations, and ultimately suggest and develop a prototype software for sanitizing existing logs for sensitive data as a solution tool.

This thesis is written in English and is 62 pages long, including 7 chapters, 4 figures, and 2 tables.

Keywords: log, anonymization, sanitization, personal data, sensitive data, safeguarding

Annotatsioon

Uute riiklike ja rahvusvaheliste andmekaitsestandardite (nagu GDPR) arengu tulemusena on tõusnud isikuandmete kaitse tähtsus ning suurenenud on ka teadlikkus nendest. Viimase paari aasta jooksul toimunud massiivsed andmelekked suurfirmadest on toonud ilmsiks puudujäägid tundlike andmete kaitsmisel. Tihti selgub, et kontrollimata logiandmetest tekkinud haavatavustest on mööda vaadatud.

Käesoleva töö eesmärgiks on anda selge ülevaade tundlike logiandmete problemaatikast, kasutades tuntud nõrkusi ja hiljutisi andmelekkeid ning pakkudes sellele mitmemõõtmelisele probleemile välja eri tasanditel olevaid lahendusi. Töös analüüsitakse olemasolevaid lahendusi ja nende piiranguid ning töötatakse välja tarkvara prototüüp olemasolevates logides olevate andmete kontrolliks ja ohutuksmuutmiseks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 62 leheküljel, 7 peatükki, 4 joonist, 2 tabelit.

Märksõnad: logi, anonüümesitus, logipuhastus, isikuandmed, tundlik info, infolekete vältimine

List of abbreviations and terms

2FA Two Factor Authentication

AGPL GNU Affero General Public License

CCPA California Consumer Privacy Act

CI/CD Continuous integration and Continuous Deployment

CISA Cybersecurity and Infrastructure Security Agency

CNA CVE Numbering Authorities

COPPA Children's Online Privacy Protection Act

CVE Common Vulnerabilities and Exposures

CWE Common Weakness Enumeration

DBMS Database management system

E2E End-to-End

FOSS Free and Open Source software

GDPR General Data Protection Regulation

GPL General Public License

HIPAA Health Insurance Portability and Accountability Act

HTTPS Hypertext Transfer Protocol Secure

ICS Industrial Control Systems

IEC International Electrotechnical Commission

IETF Internet Engineering Task Force

ISO International Organization for Standardization

ISP Internet Service Provider

JMS Java Message Service

MAU Monthly Active Users

NIST National Institute of Standards and Technology

NPM Node Package Manager

OWASP Open Web Application Security Project

PA-DSS Payment Application Data Security Standard

PAN Primary Account Number or Primary Account Number

PCI-DSS Payment Card Industry Data Security Standard

PCI-SSC Payment Card Industry Security Standards Council

PII Personally Identifiable Information

PoC Proof of Concept

RFC Request for Comments

RPC Remote procedure call

SSL Secure Sockets Layer

TLS Transport Layer Security

VPN Virtual Private Network

Table of Contents

List of Figures

List of Tables

1 Introduction

Recently some major internet websites like Facebook and Twitter requested all of their users to change their passwords due to a recently discovered vulnerability they found. Around same time, there has been major data leaks concerning products such as Microsoft Bing and some major VPN providers. All of these have the same reason in common: they were all missing safety precautions in their logging systems. One of the biggest concerns about these incidents are how the personal data such as user IP addresses and passwords were unprotected and in cleartext.

Ever than before, countries and organizations start defining regulations regarding personal data more clearly and precisely. With data frameworks like GDPR[1] becoming more popular in Europe and worldwide, internet users, organizations and law start being more interested in personal data privacy and demand higher safety standards and more control. While many layers of the problem of transferring keeping sensitive data safely has been solved using technologies like HTTPS, SSL/TLS, encryption-at-rest and other tools and best practices, most of these solutions are targeting network, application or database layers. On the other hand, the safety of logs and logging systems and sanitization of sensitive data are in logs is usually missed.

This thesis will try to answer following questions:

- What are the legal frameworks that provide protecting sensitive data?
- Is the problem of sensitive data in logs valid and current?
- What are the current approaches to provide log data safety and are they adequate?

This work will also try to design an application that can be used for detecting and removing sensitive data on various logging systems. It is aimed to be an unified, multi-platform tool.

This thesis contains the following chapters:

Chapter 2 defines the initial motivation about this subject, defining related technical terms, existing logging systems and various legal requirements about the protection of sensitive data.

Chapter 3 addresses the exact problem as well as the cause and mitigations caused by it, by providing some vulnerabilities and recent leaks. It also lists other related studies on this matter and gives more explanation about the impact of this potential problem.

Chapter 4 analyses existing approaches on both safeguarding sensitive data and preventing them from ending up in logs. At the same time, this chapter lists and compares various existing solutions in programming languages or as a 3rd party application.

Chapter 5 proposes a unified sensitive data cleanup tool and provides details on determining its requirements. It then attempts to create a proof of concept software on this design.

Chapter 6 discusses further work regarding this issue. It suggests additional features to the created concept software, as well as improvements to existing legal frameworks and development lifecycles.

2 Background and Theoretical Information

This section aims to provide prior information regarding the subject, before addressing the problem this thesis stdates.

2.1 About the Thesis

2.1.1 Motivation for Choosing the Subject

To keep up with the latest changes and to be informed about the latest news, the author has been following multiple security related newsletters. One of these newsletters are managed and distributed by BleepingComputer[1], a security news and help website. Often, the author came across with data leaks from pioneering internet companies [2][3] (see Leaks section for more). It caught the author's attention that some of these leaks are caused by publicly available logs. The author decided to do a quick research and learn more about the frequency of these log leaks and standardization on preventing them: which did not find much results. Thus the author decided to work on this subject for the thesis.

2.1.2 Objectives and Goal

Primary objectives of this thesis is to focus on sensitive data problem in logs, define the problem clearly, research and validate the existence of the problem through real cases like log data leaks, search for existing solutions and approaches addressing this problem, and propose a prototype solution to this problem.

The main goal of this work is to increase security and anonymity of logging systems and logs created by these systems with the proposed prototype, provide more compatibility with the safety standards, reduce or nullify possible future damage caused by such leaks, and increase awareness of the problem.

[1] `https://www.bleepingcomputer.com`

2.1.3 Scope

As discussed in further sections of this thesis, preventing and removing sensitive data from logs has multiple reasons, caused by multiple layers of logging system and thus different solutions for each of these layers. These layers are caused by the nature of logging process flow, including the logger application, logging solutions, the medium logging is transferred, the storage where logs reside and accessing these logs. This work will mainly focus on removing the sensitive data from logs at rest, while also mentioning other stages, existing solutions for these stages and other theoretical data.

Mentioned main focus for the solution will be both be discussed theoretically and a working software prototype as a proof of concept will be provided.

This work will be focusing on computer logs, defined in the following definitions section. This includes logs generated by any computer application and stored in file systems, logging applications or cloud applications.

2.1.4 Methodology

To approach the problem, the author decided first to increase their theoretical background on this subject by researching and understanding more about what are logging systems, what are sensitive data and personal data, what are possible requirements for removing sensitive data from systems -such as legal frameworks. Results of this work can be found at the following section.

For defining the problem part, the author will try to demonstrate the existing problems such as recent log leaks, and with the help of existing CVEs and academic literature on this subject, will try to define the risk and the impact of this problem and propose a solution or multiple solutions. To suggest a theoretical solution and a practical concept application to this solution, in addition to the related academic works, the author will also be searching for existing solutions to the problem part, comparing them and analyzing their scope and effects.

Last section of this thesis is where the author proposes a general purpose software for log sanitization. This will include both theoretical discussions about its design and software development of a prototype software.

2.1.5 Research Methods

To approach and solve the problem in this thesis, multiple research methodologies have been used for different stages.

For problem finding, formal approach is selected. To address the problem, author prior research based on inductive fact finding. In the part where prevention methods for safeguarding logs are listed and where the prototype application was proposed, analog method was used: the author combined his knowledge as well as other findings from different areas in computer science to formulate. Based on these methods, the research problem has been established.

For research mode, induction is used in this work. Through induction, facts was gathered from CVEs, CWEs and leaks to generate a theory.

Multiple research strategies were used on theory testing and problem solving. Empirical and archival approach was used for data gathering, including various cases. In the proposal part, analytic approach was used to create a unified solution and improve the existing solutions.

2.2 Logs and Logging

2.2.1 Definition

Logging is a common programming practice to collect system runtime information for post-mortem analysis [4]. A **log** is an output of this action. Every log is an output generated by a software developer: this logging action and output can either be directly generated by a program, or indirectly by a dependency (such as 3rd party libraries, modules or programs).

2.2.2 Logging Mediums

Just like any data output, generated logs can be stored in various mediums. A sample list of log output and storage mediums could be defined as:

- File (local or cloud storage like Amazon S3[1])
- Database (i.e. Elasticsearch[2], Redis[3])
- Console output
- Proxy to another service

Logs can be directed to a console output or user terminal. Since terminal screen is not a permanent storage itself, this type of logging will not be included.

Some logs can be relayed to a different service instead of storing, through Inter-process communications (IPC) or network. Syslog Relay[5] and Java Message Service (JMS) via Remote procedure call (RPC) [6] could be given as example to such proxy logging.

Most common practices of storing logs are formatted large plain text files and logging systems. The scope of this thesis is permanent logging storage systems, thus these two storage types will be included in this work.

2.2.3 Log Management Software

Logging management software are 3rd party programs that provides log management features like log collection, storage, querying or analytics. These software could be on-premise/self hosted or cloud based. Some popular examples to log management programs are listed below.

With On-premise/self hosted option:

- LogStash[4]
- Sentry[5]
- Graylog[6]

Cloud-based only:

[1] https://aws.amazon.com/s3/
[2] https://www.elastic.co/elasticsearch
[3] https://redis.io/
[4] https://www.elastic.co/logstash
[5] https://www.sentry.com
[6] https://www.graylog.org

- AWS CloudTrail[1]
- NewRelic[2]
- Raygun[3]
- SolarWinds Papertrail[4]

2.2.4 Log File formats

Log files are plaintext files which are formatted and mostly are human-readable. Some examples to these formats include:

- CSV
- JSON
- XML [7]
- Common Log Format: Created by National Center for Supercomputing Applications - NCSA) and commonly used by web servers
- Extended Log Format (ELF): Created by W3C to store web server transaction data [8]
- Graylog Extended Log Format (GELF): A JSON-formatted compressible log format to replace syslog in Graylog's own systems [9]
- IIS Log FIle Format: Created by Microsoft for their IIS web servers and based on W3C's ELF [10]

2.2.5 Log Sources Based on Purpose

We might also try to categorize logs by their purpose or the type of application or service that generates these logs.

- Web server logs (Example: ELF, IIS ELF)

[1] `https://aws.amazon.com/cloudtrail`
[2] `https://newrelic.com`
[3] `https://raygun.com`
[4] `https://www.papertrail.com`

- Database transaction logs (VLFs for MSSQL)
- Event logs (i.e. syslog, GELF)
- Application debug logs, garbage col, debugging, compiling, runtime

2.3 Compliance

This section defines various related technical standards that are relevant to the problem, its impact, or its solution in any way.

2.3.1 Data Privacy Frameworks

A **Data Privacy Framework** is a documented conceptual structure that can help businesses and governments protect sensitive data like payments, personal information, and intellectual property. The framework specifies how to define sensitive data, how to analyze risks affecting the data, and how to implement controls to secure it [11].

There are various data privacy frameworks such as international data security standards and international and local data protection laws [12].

2.3.2 Local Legislative Regulations

Local laws and acts are type of data frameworks that is an example of legislative regulations. For the purpose of defining and regulating sensitive data many countries established their own local data privacy laws and frameworks:

- Turkey: Personal Data Protection Law (KVKK)[1]
- Germany: Federal Data Protection Act (BDSH)[2]
- Australia: The Privacy Act[3]
- Canada: Personal Information Protection and Electronic Documents Act (PIPEDA)[4]

[1] `https://www.kvkk.gov.tr/Icerik/6649/Personal-Data-Protection-Law`
[2] `https://www.gesetze-im-internet.de/englisch_bdsg/`
[3] `https://www.oaic.gov.au/privacy/the-privacy-act/`
[4] `https://laws-lois.justice.gc.ca/ENG/ACTS/P-8.6/index.html`

- Brazil: General Personal Data Protection Law (LGPD)[1]
- Estonia: Personal Data Protection Act[2]

While some countries might have only one national or federal data privacy laws, there are other countries which have multiple laws on this subject. For example, United States has multiple federal and local data protection legislations, such as:

- HIPAA - Health Insurance Portability and Accountability Act[3]
- COPPA - Children's Online Privacy Protection Act [4]
- CCPA - California Consumer Privacy Act[5]
- NIST Data Privacy Network[6]

2.3.3 International Laws and Agreements

GDPR is a well-known example to international data privacy frameworks. Published by the European Parliament and the European Council, GDPR provides protection of natural persons with regard to the processing of personal data and on the free movement of such data [1]. GDPR is a wide concept that defines what data is, how it should be stored, accessed and deleted. Related to this thesis, it also includes the requirement of safety measurements to protect user data and mitigate possible risks.

Privacy Shield[7] is another international privacy framework, which is created by U.S. Department of Commerce, European Union and Swiss Administration. It includes two programs, EU-US Privacy Shield Framework and Swiss-U.S. Privacy Shield Framework, both to provide data protection regulation for the countries. This program provides compatibility with GPDR between European Union and Switzerland and United States.

[1] https://lgpd-brazil.info/
[2] https://www.riigiteataja.ee/en/eli/523012019001/consolide
[3] https://www.hhs.gov/hipaa/index.html
[4] https://www.ftc.gov/ogc/coppa1.htm
[5] https://oag.ca.gov/privacy/ccpa
[6] https://www.nist.gov/privacy-framework
[7] https://www.privacyshield.gov/welcome

2.3.4 Industry-Specific and Independent Regulations

In this sections there are regulations listed which are created and organized by international companies or organizations.

ISO/IEC 27001 Information Security Management[1] is a standard published together by International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). It specifies various standards and regulations regarding security management, including information security risks.

PCI-DSS and **PA-DSS** are two standards created and administrated by the Payment Card Industry Security Standards Council (PCI-SSC). Originally formed by world's leading credit card vendors such as Visa Inc, MasterCard and American Express, it provides standardization on credit card payments. PCI-DSS and PA-DSS provides safety regulations on payment systems. These standards defines sensitive data such as personal information and payment information and have very strict requirements on how this data should be stored and accessed.

2.4 Sensitive and Personal Data

For many, sensitive data, personal data, personally identifiable information might sound similar or even equal terms. There are no internationally standards or universally accepted definition on these terms, especially in terms of stating what defines as sensitive or personal data. Instead, these terms are defined in the scope they are used: legal frameworks and other legislation, international standardization institutes and other companies, each defining these terms according to their own scope. This section aims to provide clear definitions and distinctions.

2.4.1 Sensitive Data

Sensitive data is a wide, umbrella term that might include any type of data with a sensitive nature, including but not limited to: personal, private, confidential, classified, secret, trade secret data, which does not belong to or not released to public at the time.

In the scope of this thesis, sensitive data will be addressing to any type of data that be-

[1] `https://www.iso.org/standard/54534.html`

longs to individuals, organizations or computer systems that are not publicly available and should be kept secret and secure, and processed with security precautions such as encryption. An easier and more clear distinction can be made as any information that leakage of it could cause privacy or legal issues.

2.4.2 Personal Data and Personally Identifiable Information

Personal data will be a bigger part of the scope of sensitive information discussed in this thesis. Just like sensitive data and other terminologies mentioned in this section, definition of what personal data is depends on the context that is defined.

In most contexts, when personal data is mentioned, it usually refers to the definition in GDPR. In **GDPR**, personal data is defined as [1]:

> any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person

In United States, personal data is usually referred to as Personally Identifiable Information or **PII**. Different laws have different definitions on what states as personal data or personally identifiable information. This term is usually defined within the context of the regulation that it refers to, in most cases either a federal law or state law.

The main difference between personal data and personally identifiable information lays on the identifying part. Personal data usually refers to a sensitive information that is unique to a person, such as identification numbers or credit cards information. Personally identifiable information also includes non-personal data such as metadata, which can be used in combination with personal data to identify an individual. For example, just a name and surname might refer to any number of persons and not be considered as personal data, but in combination of date and place of birth it might be enough information to identify an individual. In fact, a study in 1990 found out that 87% of the population in United States can be uniquely identified by only gender, postal code and full date of birth information

[13]. These identifiers which are not directly unique identifiers but can be used as such in certain or sufficient combinations are called **quasi-identifiers**.

A non-exhaustive list of unique personal identifiers could be compiled as:

- National or other unique personal identification numbers, passport numbers, social security numbers
- Passwords, password hashes, PIN codes
- Financial and payment information such as credit card number, transaction details
- Physical address information such as home or work

And some example quasi-identifiers which can be used in combination to identify a person are:

- IP address
- Gender and sexual orientation
- Zip code, city, country
- Healthcare information
- Digital metadata, analytics, application telemetry

2.4.3 Anonymity and Anonymization

Anonymity is the state of data or person that is not uniquely identifiable. **Anonymization** is removing identifying information from a data. Depending on the requirement or context, it might suggest removing personal data only, or also metadata. **Metadata** is a reference that defines or represents to a data. While anonymyzing a data, scrubbing metadata should also be considered. The opposite of the anonymization operation is called **de-anonymization** or **re-identification**.

Anonymyzing sensitive data can be done by [14]:

- Removing the sensitive data completely

- Partially or fully masking sensitive data
- Splitting the data into separate tables (compartmentalize)

One common way of measuring the anonymity of a data is k-anonymity. **k-anonymity** is an indicator to measure the anonymity of a data, first introduced in 1998 [15]. It became popular in 2018 when it was used along to create leaked personal information checking website haveibeenpwned.com[1]

[1] `https://haveibeenpwned.com`

3 Problem and Validation

3.1 Problem Statement

Logs are just another type of data produced by either computer systems or its users. Just like any data that is being transmitted or stored, safety and security of log data should also be provided. While this is very common practice for other types of data and data storage such as DBMS, log data is often overlooked. Another problem with logs is that it is usually raw data, with no modifications or safety precautions (i.e. password hashing, user data sanitizing). Thus, a potential attacker accessing this log data could impose a bigger, hidden problem. Logs are one of the common reasons of data leaks by companies and systems.

This section aims to demonstrate the importance, relevance and the impact of this security problem. Theoretical validation and relevance is done by researching related vulnerabilities. Practical validation is provided by some number of recent data leaks, all caused by unsanitized or unsecured logging systems. The impact of this vulnerability is discussed further.

3.2 Related CVEs and CWEs

Common Vulnerabilities and Exposures (CVE)[1] is a public database for publicly disclosed cybersecurity vulnerabilities, which is run by U.S. Homeland Security[2] and The MITRE Corporation[3] [16]. Common Weakness Enumeration (CWE)[4] is a category reference system for computer software weakness and vulnerabilities, which is also maintained by these same organizations.

CVE and CWE systems are world's leading weakness and vulnerability directory authorities. This is why this section will be focusing on weakness and vulnerability databases by MITRE (CVEs and CWEs) only.

The MITRE Corporation and U.S. Homeland Security are not the only organizations that

[1] `https://cve.mitre.org`
[2] `https://www.dhs.gov/science-and-technology/cybersecurity-programs`
[3] `https://www.mitre.org`
[4] `https://cwe.mitre.org`

can contribute or manage this directory. MITRE also provides and organizes CVE Numbering Authorities (CNA), which authorizes organizations around the world to assign CVE IDs to products and services managed by them [17]. Organization is managed by Root and Top-Level Root CNA roles, consisting of MITRE Corporation and Cyber security and Infrastructure Security Agency (CISA) Industrial Control Systems (ICS). As of April 12, 2021, CNA program has 163 organizations and 27 countries participating, which does not include Estonia to date. List of participating countries and organizations can be found at CVE IDs website[1].

The MITRE Corporation also has a CWE Compatibility and Effectiveness Program[2] for companies with security services and products. As of April 2021, 58 products and companies are recognized by this program [18].

This section aims to find, list and explain existing weaknesses, vulnerabilities and exposures regarding the safety of sensitive data in computer generated logs.

3.2.1 CWE-532: Insertion of Sensitive Information into Log File

Submitted anonymously on 2006, this CWE is the most referenced CWEs of MITRE on the security of sensitive information in logs. It describes a weakness where sensitive information might cause an attack or possible expose. This weakness seems to be the most relevant CWE for the problem scope of this thesis. Table below shows summary information for CWE-532.

[1] `https://cve.mitre.org/cve/request_id.html`
[2] `https://cwe.mitre.org/compatible/program.html`

Table 1. Information about CWE-532: Insertion of Sensitive Information into Log File[19].

Description	Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information.
Phase of Introduction	Incorrect design related to an architectural security tactic
Consequence	Confidentiality: Provides attackers with an additional, less-protected path to acquiring the information
Likelihood Of Exploit	Medium
Submission	2006-07-19, Anonymous (Under NDA)

Connections

This CWE is also related to the following other weaknesses:

- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor, 2006[1]
- CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory, 2006[2]

The CWEs below were previously defined as independent weaknesses but then were deprecated because their abstraction was found too low-level and joined with CVE-532 instead [20, 21, 22]:

- CWE-533: Information Exposure Through Server Log Files, 2006 [20]
- CWE-534: Information Exposure Through Debug Log Files, 2006 [21]
- CWE-542: Information Exposure Through Cleanup Log Files, 2006 [22]

Mitigation

[1] `https://cwe.mitre.org/data/definitions/200.html`
[2] `https://cwe.mitre.org/data/definitions/538.html`

Another importance of this CWE is the similarity of the suggested mitigation by MITRE to suggested solution defined in the related section. Potential mitigations of this CWE is described as table below:

Table 2. Potential Mitigations of CWE-532.

Phase	**Possible Mitigation**
Architecture and Design, Implementation	Consider seriously the sensitivity of the information written into log files. Do not write secrets into the log files.
Distribution	Remove debug log files before deploying the application into production.
Operation	Protect log files against unauthorized read/write.
Implementation	Adjust configurations appropriately when software is transitioned from a debug state to production.

The only mitigation not discussed here is the removal of existing sensitive data from the logs, which constitutes the purpose of this thesis.

Summary

Observing CWE-532 has been reported in 2006 and added to various categories approximately every few years until 2019 tells that this weakness is a widely used and referenced possible weakness.

3.2.2 CWE-117: Improper Output Neutralization for Logs

CWE-117 describes weakness when data from an untrusted source such as an attacker is written to the log due to lack of sanitation or neutralization of output [23]. It might look like it is related to the subject of the thesis, sensitive data in log files, but that's not the case. CWE-117 describes a case where the attacker causes undesired effect by changing what is logged.

3.2.3 CWE-312: Cleartext Storage of Sensitive Information

CWE-312 is a generic cleartext sensitive information storage. It is not limited to logs; it also includes other programmatic flows and artifacts such as functions, cookies, and other server responses. Because of its wide applicability, this CWE is a part of many CVEs and categories such as OWASP Top Ten Category [24].

3.2.4 CWE Categories

In Mitre terminology, a (CWE) category is a CWE record that is not a weakness itself, but a set that contains other weaknesses or categories, to help with grouping weaknesses in structure. Categories also help with auditing and making standard compliance of existing software easier.

The CWE categories that are related to the CWEs mentioned above are as follows:

- CWE-199: Information Management Errors, 2006[1]
- CWE-731: OWASP Top Ten 2004 Category A10 - Insecure Configuration Management, 2008[2]
- CWE-857: The CERT Oracle Secure Coding Standard for Java (2011) Chapter 14 - Input Output (FIO), 2011[3]
- CWE-963: SFP Secondary Cluster: Exposed Data, 2014[4]
- CWE-1009: Audit, 2017[5]
- CWE-1036: OWASP Top Ten 2017 Category A10 - Insufficient Logging & Monitoring, 2013[6]
- CWE-1147: SEI CERT Oracle Secure Coding Standard for Java - Guidelines 13. Input Output (FIO), 2018[7]
- CWE-1210: Audit / Logging Errors, 2019[8]

[1] https://cwe.mitre.org/data/definitions/199.html
[2] https://cwe.mitre.org/data/definitions/731.html
[3] https://cwe.mitre.org/data/definitions/857.html
[4] https://cwe.mitre.org/data/definitions/963.html
[5] https://cwe.mitre.org/data/definitions/1009.html
[6] https://cwe.mitre.org/data/definitions/1036.html
[7] https://cwe.mitre.org/data/definitions/1147.html
[8] https://cwe.mitre.org/data/definitions/1210.html

3.2.5 CVEs

The weaknesses (CWE) described above in this section are already defined as potential vulnerabilities (CVEs). Some of these observed vulnerabilities are defined as follows:

- CVE-2017-9615: Password exposure in Cognito Software Moneyworks 8.0.3 related to verbose logging[1]
- CVE-2018-1999036: SSH private key password exposure in build log in Jenkins SSH Agent Plugin 1.15 and earlier versions[2]
- CAPEC-215: Using fuzzing tools to send malformed messages then to observe sensitive information on logs and error messages[3]
- CVE-2019-3830: Information Exposure in ceilometer-agent prints sensitive configuration data to log files without DEBUG logging being activated[4]
- CVE-2007-4786: Cisco Adaptive Security Appliance (ASA) composes cleartext passwords[5]

It could be easily seen that since the logging can be generated by any programming language or computer software, known vulnerabilities vary from programming languages like Java to CI/CD tools like Jenkins, showing the width of its scope.

3.3 Recent Leaks, Breaches and Announcements

This section includes some recent leaks and other similar hacking attacks, which are all partially or fully caused by sensitive data in logs, in the purpose of trying to exhibit the relevance to the thesis and consequence of having sensitive data in logs.

[1] `https://nvd.nist.gov/vuln/detail/CVE-2017-9615`
[2] `https://nvd.nist.gov/vuln/detail/CVE-2018-1999036`
[3] `http://capec.mitre.org/data/definitions/215.html`
[4] `https://nvd.nist.gov/vuln/detail/CVE-2019-3830`
[5] `https://nvd.nist.gov/vuln/detail/CVE-2007-4786#vulnCurrentDescriptionTitle`

3.3.1 2018 Twitter Password Change Announcement

In May 2018, Twitter[1], one of the biggest social networks and microblogging services as of today, announced a potential threat in their blog [25]. In the blog post, Company's Chief Technology Officer Parag Agrawal stated that they identified a bug that stored user passwords in clear text format in an internal log. They stated that normally they mask the user passwords using bcrypt before storing them, but this bug caused storing those passwords before the hashing, thus unencrypted. Although Twitter claimed that they did not detect any breach or misuse of this data, they requested all their users to change their password, just in case.

According to BusinessOfApps website[2], in the first quarter of 2018, Twitter had 336 million Monthly Active Users (MAU) worldwide [26]. Since Twitter suggested all of their users to change their passwords and enable 2FA, it's safe to assume that all of these users were under risk.

This was not the first time a risk related to sensitive data in Twitter being stored in logs, in cleartext, In June 2016, LeakedSource announced that they had access to around 32 million users' information such as username, email and cleartext passwords [27], but it then was claimed by BankInfoSecurity website and then verified by LeakedSource that the leak data was not coming from Twitter itself but its customers [27][28].

3.3.2 2019 Facebook-Instagram Password Change Announcement

A potential risk similar to Twitter's passwords stored in logs was revealed by Facebook[3] in 2019. On March 2019, on their official news page, Facebook announced that they have discovered passwords of *some* users were stored in readable format on their data storage systems [29]. Including the information from the update came a month later, the total number of users affected were hundreds of millions of Facebook Lite users, tens of millions of other Facebook users and millions of Instagram users, of their total 2.5 billion Monthly Active Users (MAU) [30].

Facebook claimed that although they use cryptographic hashing functions such as scrypt to encrypt passwords one way before storing them on their databases, they still had same

[1] `https://twitter.com`
[2] `https://www.businessofapps.com/`
[3] `https://facebook.com`

passwords stored as human readable cleartext format on their logs.

3.3.3 2020 Microsoft Bing Search Logs Leak

On September 2020, a security team of the mobile news website WizCase[1] led by Ata Hakcil has found a massive data leak belonging to Microsoft's Bing Search[2] mobile app, which is available both in AppStore[3] and Google Play Store[4] [2]. The leak was caused by unsecured logging data stored in ElasticSearch, similar to the Hong Kong VPNs Hack mentioned in this section. Size of the leak is over 6.6 terabytes, including search records from users of more than 70 countries. The data revealed to be including personal and sensitive information including searched terms, GPS coordinates of the user up to 500 meters precision, device and operating system info and various IDs assigned to users by Microsoft, including their Microsoft IDs.

Figure 1. A sample from the Bing leak showing sensitive fields such as coordinates, device ID and search info.

[1] https://www.wizcase.com
[2] https://www.bing.com
[3] https://www.apple.com/app-store
[4] https://play.google.com/store

3.3.4 2021 Hong Kong based VPN Companies Log Leaks

Just recently, in March 2021, a research team led by Noam Rotem of the VPN comparison and news website vpnMentor[1] found Personally Identifiable Information (PII) data belonging to the users of multiple VPN companies [31]. According to the report, the affected VPN companies are UFO VPN, FAST VPN, Free VPN, Super VPN, Flash VPN, Secure VPN, and Rabbit VPN, which are all based in Hong Kong and connected by a common app developer. Total number of users claimed by these VPN providers are 20 million. The leak consists of over 1 billion log files, sizing more than 1.2 terabytes.

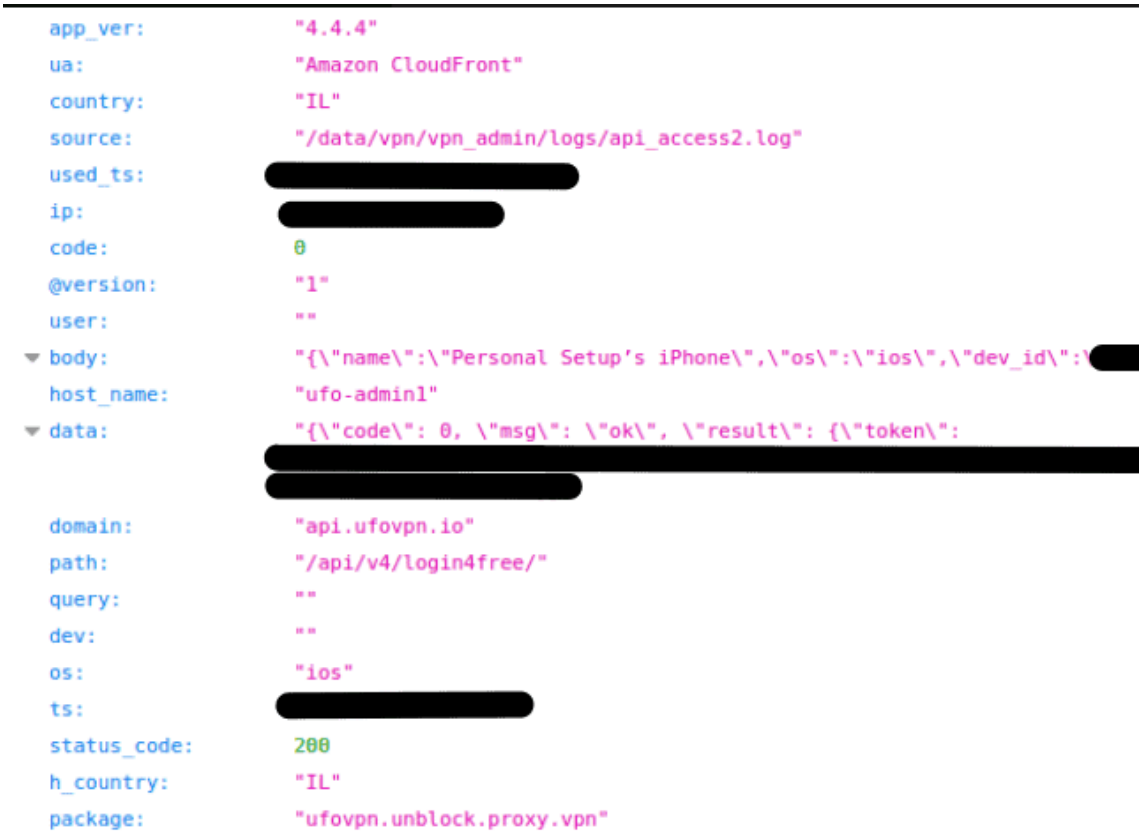

Figure 2. Activity log for UFO VPN app showing source as logs and leaked personal data.

According to the screenshots vpnMentor shared, some of the leaked logs seem to be stored at ElasticSearch. Two of the most common usages of ElasticSearch are to provide a full-text search engine for databases, and, as a part of ElasticStack (or formerly known as ELK Stack[2] - ElasticSearch-LogStash-Kibana), storing and processing logs. It was claimed that all these VPN companies were using the same ElasticSearch instance to

[1] https://www.vpnmentor.com
[2] https://www.elastic.co/elastic-stack

store their logs.

The logs included emails and passwords of users in cleartext, password changes including the old and new passwords, connection information and web activities through the VPN services such as location, IP address, user agent headers and website traffic. In addition to the risk and damage caused by personal and sensitive user information, because of this security incident, legal problems might also occur for such countries where a user is connected to a website that is illegal on their country. One other potential legal problem surfaced with the leak is that some of the VPN companies claimed that they do not log user traffic, which was found untrue.

3.4 Chapter Summary

Technical details and the example leaks provided in this section clearly states the existence of the problem and it's impact. Although the weakness about storing sensitive data in logs has been defined at least since 2006, there are still new leaks happening today by big corporations. Vulnerabilities and damages caused by this weakness not only affects the applications and the companies but also the users of these companies and applications.

Another revelation CVEs and CWEs show us is the multidimensionality of leakage problem. Sensitive data in logging can be caused by in any level of the application flow. Thus, solving and preventing leaking of any sensitive data in logs needs to be accomplished in multiple different places. Following chapter focuses on these layers.

4 Preventive Measurements and Existing Solutions

The main problem defined in this thesis is the detection and removal of sensitive data in logs. However, there are no single solutions to this problem, because leaking sensitive data in logs can be caused by multiple number of reasons. In an application or a digital transaction, logging event might occur on any part of this whole workflow. For example, a user's interaction with a web based application consists of the user's browser, user's and application's internet connections, proxy servers, application server and database server. Each of these components might log sensitive data. That's why we can discuss the vulnerabilities and best practices to avoid sensitive data leaking for each of these layers.

4.1 Preventing Unnecessary Sensitive Data

First defense against a weakness is not to have it in the first place. According to this best practice, any part of digital communication: from internet and network to applications, should not handle, process, transport or store any sensitive data unless they are required to do so. Any sensitive information received or processed should be destroyed and not stored, if not needed any further.

4.2 Protect Sensitive Data During Transport

Two of the most common best practices to safeguard sensitive data on transport are SSL/TLS, hashing, and using POST.

Starting from 2018, Google Chrome started marking websites using unsecure HTTP protocol as 'unsafe' [32]. As of April 2021, between 78% and 98% of the websites visited using Google Chrome browser are using secure transport via SSL/TLS. However, there are often still websites using HTTP. HTTPS provides end-to-end encryption between parties, ie web browser and web server; protecting any intermediate routers from reading encrypted data. Importance of this practice becomes much more crucial when a sensitive data is concerned. Two very likely reasons why a web application might be using HTTP are either being a prototype/non-production system or thinking that setting up HTTPS takes extra time. However, With the self-signing certificate authorities like Let's Encrypt[1]

[1] `https://letsencrypt.org`

becoming more popular, SSL/TLS configurations became easier.

Secondly, any sensitive data that can be hashed should be hashed as early as possible, ideally before leaving the client. Most commonly used field for this are password fields. One common risk is that any possible loggers between the client's browser and part of web application's code handling the client input could log these fields in plaintext. Facebook's [29] and Twitter's [25] recently found vulnerabilities mentioned above are caused by lack of this practice.

Finally, any sensitive data that needs to be transferred between the client and server should be located in body part of an HTTP POST request (as defined in IETF RFC 7231 [33]), not the URL path. Besides from POST body, the other way of transferring parameters in an HTTP request is query strings, which are also located in the URL. The problem with this anti-pattern is, all HTTP requests are very often logged by multiple parties: internet provider (ISP) systems, proxy servers and web servers. If there are any sensitive data in the URL such as query strings, this data will also be in the log. Also HTTPS connections does not protect from this vulnerability, since HTTPS encrypt the request body, not the URL part.

Another precaution could be taken against the risk of leaking sensitive data on HTTP connections is to strip out any sensitive data before redirecting. For example, Ruby on Rails provides `config.filter_redirect` config[1] to allow users to remove a given string or regular expression from the URL before redirecting.

4.3 Isolate Sensitive Data

According to this best practice, in an application dealing with sensitive data, only the parts of the application (such as classes, modules and units) should access to the sensitive data. No other parts of the application and no other interconnected services that does not require this sensitive data should even receive it. This practice reduces attack surface and probability.

For example, when designing a system with passwords, only that service in whole application should have read and write access to passwords. The passwords should be stored

[1] `https://guides.rubyonrails.org/action_controller_overview.html#redirects-filtering`

in databases should be only accessible from within this service. Any other component or service should instead ask this when doing password operations like checking or changing it.

One other advantage of isolating components accessing sensitive data is control. When all the actions regarding these sensitive data is logged, any access could be audited and any unwanted access could be discovered.

4.4 Explicitly Define What to Log

When logging on application, the sensitive data inside the log should be sanitized. Most common way of doing this is explicitly specifying the exact data to be logged. An anti-pattern to this is when logging whole data sets and objects like database results and incoming/outgoing HTTP body, which can include sensitive data such as passwords.

Another way to sanitize logs is to use filters with the logging. Some programming languages, frameworks and loggers has this functionality.

For example, in Winston[1], a popular logger for Node.js and JavaScript has a formatter functionality[2]. This feature is originally created to format the output format of a log. However, this feature can also be used to filter out sensitive data. Example usage is shown in Appendix Code 6.

In some languages and frameworks, a more ready-to-use implementation of this feature already exists. For example, in Ruby on Rails[3], a strongly-opinionated web application framework for Ruby[4], `config.filter_parameters` parameter provides filtering user-specified fields before logging[5]. Rails automatically adds `:password` field to the default options and other fields such as credit card number or password salt can be added. Some packages like `logstop`[6] and `blouson`[7] extends this functionality of Ruby to provide detection of some predefined fields. This method in general, however, is also not fool-proof since every single field to be excluded needs to be defined explicitly. With growing appli-

[1] `https://github.com/winstonjs/winston`
[2] `https://github.com/winstonjs/winston#formats`
[3] `https://rubyonrails.org`
[4] `https://www.ruby-lang.org`
[5] `https://guides.rubyonrails.org/action_controller_overview.html#filters`
[6] `https://github.com/ankane/logstop`
[7] `https://github.com/cookpad/blouson`

cation size, developers might forget adding some new sensitive fields.

An example to a more comprehensive package for log sanitization on application-level is `log-sanitizer`[1]. This Java library intercepts before the logging action to detect any sensitive data, using built-in detectors for payment card numbers (PAN), IBAN numbers and some common password hashing formats, as well as user defined fields.

More examples to other log formatters are:

- `pino`[2], a logger for Node.js has `redact` feature to redact specified fields before logging
- `node-bunyan`[3], a logger for Node.js, supports serializer functions
- `og`[4] for Elixir programming language is a collection of logger helpers, including formatting
- `Serilog.Sanitizer`[5] and `Serilog.Enrichers.Sensitive`[6] provide sanitization for Serilog logger on .NET
- `loguru`[7] for Python supports formatters

4.5 Checking Application Code For Logging Leaks

One best practice is to make sure the application does not shipped with code that logs sensitive data.

A manual way to achieve this is through pull requests (as in GitHub terminology) or merge requests (as in GitLab terminology). In an ideal development environment, every pull request should be reviewed by at least one other party. This review could also include extra caution on logging functions. One other thing to look out for when reviewing code is using correct logging levels for each environment: i.e. not using DEBUG level logging in production.

[1] `https://github.com/mjeffrey/log-sanitizer`
[2] `https://getpino.io`
[3] `https://github.com/trentm/node-bunyan`
[4] `https://hexdocs.pm/og`
[5] `https://github.com/waxtell/Serilog.Sanitizer`
[6] `https://github.com/sandermvanvliet/Serilog.Enrichers.Sensitive`
[7] `https://github.com/Delgan/loguru`

These checks could also be executed automatically, usually with the CI/CD pipeline. Developers can write a small program that checks for and matches known sensitive data fields and values in a changed code, and automate checks by this application on CI/CD process. There also exist 3rd party tools that could be integrated to CI/CD tools, such as Detectify[1].

4.6 Sanitize During and After Logging

If it is unavoidable to prevent sensitive data being logged or the data that is logged cannot be controlled sensitive information could still be sanitized. Many logging systems provide various solutions or workarounds to this problem. There are various possible stages in a logging flow where sensitive data can be scrubbed or masked from:

- On the logger side, logging driver or software can be configured to mask or drop certain fields
- A relay server can be located between the log source and logging service to intercept logging traffic
- Logging server or service may provide data input configuration to filter and change log data after it's received and before it's written
- While querying existing logs, logging applications may be able to filter results and hide certain fields or values

Below the author has searched for and listed some popular logging systems with their anti-sensitive data measurements.

NewRelic[2], an online log management and event analytics platform, provides `drop` command in NewRelic's GraphQL client NerdGraph, to remove specified fields [34].

Application monitoring and error tracking service **Sentry.io**[3] provides various solutions to safeguard sensitive data in logs in their application. On the log source machine, Sentry SDK can be configured to scrub sensitive data before sending [35] and Sentry's Server-Side Data Scrubbing feature can be used to to scrub when the data arrives to Sentry

[1] `https://detectify.com`
[2] `https://newrelic.com`
[3] `https://sentry.io`

servers [36]. To control the data that is already sent to Sentry servers, Sentry provides Advanced Data Scrubbing feature [37]. Sentry also has a product called Relay[1] which is installed separately, runs between the logging application and Sentry servers, and can scrub personally identifiable information before sending them.

Splunk[2] is a data and IT security monitoring and analysis tool. For log management, Splunk has various features to sanitize data. `SEDCMD` or regex settings allow a user to define keywords to replace strings in logs before sent to Splunk cloud [38]. When searching through logs, `scrub` can be used to filter out sensitive data before displaying the query result [39]. And when using Stream Processor Pipeline, `replace` function can be used to manually match and mask strings of selected fields [40].

4.7 Checking Configuration

One best practice is to make sure the application does not shipped with code that logs sensitive data.

A manual way to achieve this is through pull requests (as in GitHub terminology) or merge requests (as in GitLab terminology). In an ideal development environment, every pull request should be reviewed by at least one other party. This review could also include extra caution on logging functions. One other thing to look out for when reviewing code is using correct logging levels for each environment: i.e. not using DEBUG level logging in production.

These checks could also be executed automatically, usually with the CI/CD pipeline. Developers can write a small program that checks for and matches known sensitive data fields and values in a changed code, and automate checks by this application on CI/CD process. There also exist 3rd party tools that could be integrated to CI/CD tools, such as Detectify[3].

[1] `https://docs.sentry.io/product/relay/`
[2] `https://www.splunk.com`
[3] `https://detectify.com`

5 Prototype Application for Proposed Solution

The main problem defined in this thesis is the detection and removal of sensitive data in logs. Hence, the main solution will be focused on this. The proposed solution will be targeting the logs, reading and analyzing them , detecting any sensitive data and removing if they exist.

As discussed in previous chapters, the author failed to find any general-purpose sensitive data cleaning tool from logs. For this reason the author decided to design and develop a prototype application for this goal. Purpose of this prototype application is to define requirements and properties of an ideal log sanitization application, and to provide a working application, following some of these ideal specifications.

5.1 Alternative Tools and Their Comparison

Before writing a proof of concept application, the author searched for any existing application or library that provides a feature removing sensitive data from logs. Below are listed some examples that are worth mentioning by the author.

5.1.1 NCSA FLAIM Framework

In 2006, Adam Slagell, Kiran Lakkaraju, Katherine Luo wrote the paper FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs[41] that suggests a framework for log and network traffic sanitization. At the same year, a tool of the same[1] has been introduced. This paper and framework is very comprehensive in terms of defining risks. However, the latest version of this tool has been released in 2008 and this project is unreachable at the moment.

5.1.2 Amazon Macie

Amazon Macie[2] is a cloud-based sanitization service on Amazon Web Services (AWS). This sensitive data detection and removal tool connects to Amazon S3[3] and AWS Cloud-

[1] `https://github.com/fulinux/flaim`
[2] `https://aws.amazon.com/macie`
[3] `https://aws.amazon.com/s3`

Trail[1], collects metadata information and using it's own lists (Theme[2], Regex[3], Support Vector Machine (SVM) classifiers[4]) scans Amazon S3 buckets various types of data such as logs, source codes, office productivity documents for sensitive data such as banking information, unique personal identifiers, birth date as well as security attack keywords and password hashes.

This product is very comprehensive, but it has it's caveats. First to mention is that it is a paid, closed-source and cloud-only product, which are all limiting it's usage. Secondly, this tool is built for scanning files located in a S3 bucket, limiting using other log storage systems.

5.1.3 Microsoft logsanitizer

The author found logsanitizer project[5] On GitHub while searching for alternatives. This tool is similar to author's proposed prototype: it reads logs from files, detects each lines, filters based on predefined rules and rewrites log file. However this project is not maintained since 2016. Also, author's solution provides more extensibility like supporting different logging systems and auto-detecting sensitive fields.

5.1.4 scrubadub

Scrubadub[6] is a generic PII cleanup tool. It scans free texts and finds sensitive fields such as names, email addresses, usernames and passwords. In comparison with suggested prototype, the missing features are not being specialized for logging systems, not supporting non-file logs (such as LogStash/ElasticSearch or CloudTrail), and not supporting custom fields.

[1] `https://aws.amazon.com/cloudtrail`

[2] `https://docs.aws.amazon.com/macie/latest/userguide/macie-classify-objects-theme.html`

[3] `https://docs.aws.amazon.com/macie/latest/userguide/macie-classify-objects-regex.html`

[4] `https://docs.aws.amazon.com/macie/latest/userguide/macie-classify-objects-classifier.html`

[5] `https://github.com/fulinux/flaim`

[6] `https://scrubadub.readthedocs.io`

5.1.5 git filter-branch

Git version control system has a feature called `git filter-branch` which can be used to search for and remove sensitive data from git history [42]. This tool is only for git repositories, and works by first searching for a filename in all git history and then deleting it if wanted, erasing the file itself from all of the previous commits.

5.1.6 loganon

loganon[1] is a project on GitHub that was created first in 2015. However, the project has not been maintained since 2018. It is a generic log anonymyzer, just like the author's solution. However this tool supports only logs from Microsoft Exchange server, Dovecot mail server, Fortigate products and Infoblox, hence has limited functionality.

5.1.7 Spectx

Spectx[2] is a log parsing and analysis tool. It has filter option which can be used to query sensitive data by specifying manually. However this tool does not have any feature to detect sensitive data automatically or rewriting logs. Yet, Spectx still can be used to identify sensitive data in logs with custom queries.

5.1.8 Database Cleaners

mlogcensor[3] is a tool to scan MongoDB logfiles for sensitive data.

DataDefender[4] is a generic sensitive data discovery and anonymization tool that supports various database management systems such as MariaDB, MySQL, MSSQL, PostgreSQL and DB2. However, this solution is limited to databases only.

[1] `https://github.com/sys4/loganon`
[2] `https://www.spectx.com`
[3] `https://github.com/johnlpage/mlogcensor`
[4] `https://github.com/armenak/DataDefender`

5.2 Target Audience

The target audience of this application is any person or company possessing logging systems or log files, which include but not limited to:

- IT, network, system and infrastructure administration personnel
- Developers
- IT managers and other related managers
- Security consultants, researchers and bounty hunters

Any organization who also does fit the targets defined above but is obligated to comply with any local or international regulations like local laws, GDPR or PCI-DSS is also encouraged to use this application. Another target audience for this tool is any application or company that process or store any sensitive information.

5.3 Product Specifications and Limitations

This section defines the workflow and product requirements. Application flow of prototype tool could be defined as follows:

- Start program
- Specify log file or service to analyze
- Configure and test log access
- Partially analyze log and detect sensitive data
- Start sensitive data removal
- Exit program

To provide this functionality, specifications for proof of concept application are defined as such, to support:

- Multiple log formats and services (i.e. Nginx, Apache and Systemd)

- Very big log files
- Streaming operations
- Analyzing the log structure and map its fields
- Detecting sensitive fields and values
- Future extensions (i.e. plugins)

The purpose of creating this application is to provide a proof of concept for the proposed solution. Thus, the scope will be limited. In the current version, the application will support only specified Nginx log files; however it will also provide a way to add support for other log sources. It will also include a limited number of sensitive fields such as username, password and IP address, and will also provide an extensible system for future additions. Developed program will only run in dry-run mode, meaning after scanning the log file and detecting sensitive fields and values, it will not modify the file by removing or masking sensitive fields. This feature was taken out of the scope of this subject and could be easily implemented in the future.

5.4 Technical Requirements

Proposed technical requirements of the prototype tool are:

- Cross-platform: Supports multiple operating systems; but at least Unix/Linux support is a must, since over 70% of public servers on the internet are using Linux[43]
- Multi-architecture: Should at least support 32-bit (x86) and 64-bit (x64) CPU architectures
- FOSS: Free and Open Source software, with a permissive license
- Community support: Used 3rd party dependencies should be well-used and well-supported by their communities. This would provide increased stability and safety.
- Type-safe programming: The program should have at least compile-type type safety via strong typing, static typing, or both.
- Automated testing: Should have at least unit tests for core functionality. Ideally, 100% code coverage for units plus some happy path Integration/E2E (End-to-End) tests.

5.5 Technology Selection

The author is currently working as a Software Engineer developing full-stack web applications. Amongst the technology stack author has, Node.js[1], JavaScript, TypeScript and Jest are some of the main programming environments the author has the most experience.

JavaScript is a compile-time, dynamically typed programming language created by Brendan Eich in 1995, which is based on EcmaScript[2]. It is one of the most used programming languages in the world [44]. JavaScript was originally created to run on web browsers: most of the JavaScript engines that compiles and runs the code on run-time are created for web browsers, such as Google's V8[3] (used by Chromium-based browsers), Mozilla's SpiderMonkey[4] (used by Firefox and other Mozilla Application Framework products[5]) and WebKit[6] (used on Apple platforms).

Node.js is a single-threaded runtime environment created for running JavaScript on server-side. It uses Google's V8 for runtime engine, libuv[7] for asynchronous event loop and CommonJS[8] API for module ecosystem. Node.js's embedded package manager NPM[9] has the highest number of publicly available modules than any other programming language, with more than 1.5 million packages [45]. It has binaries for macOS, Windows (x86 and x64) and Linux (x86, x64, ARM, System z, Power LE).

TypeScript[10] is a superset of JavaScript, created by Microsoft in 2012. It provides some important features and programming paradigms that JavaScript is missing, including static typing (compile type) and namespaces. It transpiles to JavaScript using transcompilers like it's own TypeScript Checker/Compiler (tsc) or Babel[11], which makes it being able to run on both client-side or on server-side (including Node.js).

Jest[12] is a testing framework for JavaScript, created by Facebook in 2014. It has a wide

[1] `https://nodejs.org`
[2] `https://262.ecma-international.org`
[3] `https://v8.dev`
[4] `https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey`
[5] `https://mozdev.org`
[6] `https://webkit.org`
[7] `https://libuv.org`
[8] `http://www.commonjs.org`
[9] `https://www.npmjs.com`
[10] `https://www.typescriptlang.org`
[11] `https://babeljs.io`
[12] `https://jestjs.io`

support of JavaScript-based technologies such as Babel, React.js, Vue.js, TypeScript and Node.js. Jest is a all-inclusive test suite: it has it's own assertion library, test runner, code coverage, function and timer mocks and snapshot support.

To develop the prototype application, Node.js, TypeScript and Jest is chosen by the author. JavaScript and Node.js provides very agile development and runtime pace. TypeScript is also selected to provide type safety on top of JavaScript, which allows safer and more secure programs. Node.js has very wide processor architecture support (see the definition above). In addition, because of its very small memory footprint and runtime overhead, it's well adopted to many serverless and other cloud provisioning services such as AWS Lambda[1], Google Cloud Functions[2] and Cloudflare Workers[3] which allows Node.js to run on cloud on a lower provision level to parse cloud-based logs faster. Node.js/JavaScript/TypeScript ecosystem also supports asynchronous programming out of the box, which is ideal for data streams and Event-drive programming, and becomes very useful on reading and writing multiple or very big log files.

5.6 Source and Licensing

The author has chosen to publish this application as Free and Open Source Software (FOSS) by publishing the source code and a permissive software license. The source code is hosted freely at GitHub on `github.com/yasinaydinnet/sanitize-logs`. A `README` file is provided to show how to run the application, how to develop the application and how to run automated tests. The issues and discussions features of GitHub were enabled for this repository to allow future users of the application to discuss features and possible bugs.

There are multiple licensing types and existing standards for free and open source software. Some commonly used permissive licenses include MIT License[4], Apache License[5], General Public License (GPL)[6] and Creative Commons[7]. The author chose to allow any personal or commercial copy and usage of the application, as long as they are also

[1] `https://docs.aws.amazon.com/lambda/latest/dg/lambda-nodejs.html`
[2] `https://cloud.google.com/functions/docs/quickstart-nodejs`
[3] `https://workers.cloudflare.com/node`
[4] `https://mit-license.org/`
[5] `https://www.apache.org/licenses`
[6] `https://www.gnu.org/licenses/gpl-3.0.html`
[7] `https://creativecommons.org/licenses`

open source. GPL licenses fits these requirements. As the license of the application, the author has chosen GNU Affero General Public License (AGPL) version 3, which, in addition to GPL license, requires any application that is calling this application over the network also be open-source [46]. To help with license comparison and selection, `https://choosealicense.com` website was also used.

5.7 Packaging and Usage

The PoC application was developed with Node.js. The author has published the source code as publicly accessible at NPM.js[1] located at `https://www.npmjs.com/package/sanitize-logs`. This allows developers to include `sanitize-logs` (the PoC application) inside their applications using Node.js package managers such as NPM or Yarn.

The author aimed to create the application in an executable or to be easy to run. In NPM, two of the most common methods of making an application usable globally are `global installation` and `npx`. The author implemented both of these methods inside application. This is done by defining `"bin": "./bin/app.js"` parameter in `package.json` file (see Appendix Code 3 for full file).

When installing a package using `npm install`, a `--global` parameter could be provided to install the specified package as global module, which, if the app is configured as such. To install and run the application, following commands could be run:

```
npm install --global sanitize-logs
sanitize-logs
```

`npx`[2] is an NPM command that allows users to run a configured Node.js application locally or globally, without installing it. `npx` is provided by NPM versions 7.0 and above. To run the application, following command could be run:

```
npx sanitize-logs
```

[1] `https://www.npmjs.com/`
[2] `https://docs.npmjs.com/cli/v7/commands/npx`

5.8 Application Usage and Parameters

There are two possible program modes that could be implemented: interactive and unattended. Interactive mode provides a user interface (either command-like or graphical) and an interactive interface that the user can interact with the application on each step, mainly to provide application parameters. In unattended mode, the application parameter are provided by command line parameters while calling the application. This feature of unattended mode allows it to be integrated better with automated tools. For this reason, the author has chosen to include only an unattended command-line interface for proof of concept. When the application is run without any parameters, the program returns an error, warning the user.

To detect program arguments in Node.js, `commander.js`[1] was used. In the code, `src/lib/args.ts` file creates a commander instance, defines which arguments the program allows, reads defined arguments from the current instance and allows access to these parameters. The author defined the commander instance as variable in global scope to provide caching of the values. At the moment, following parameters are allowed in the application:

- `-h`, `--help`: Help screen to list available fields. Included in commander.js.
- `-t`, `--type`: Log type (ie Nginx, Apache, ElasticSearch). For PoC, only **Nginx** log is implemented.
- `-f`, `--file`: File path for file logs. It works both with relative paths and absolute paths. The file should be both readable (to scan) and writable (to update). Prior to scanning, the application checks these permissions.
- `-d`, `--debug`: Enables debug mode for verbose logging.
- `--testrun`: Runs application with sample Nginx log data.

An example usage of the application, defining an Nginx Access Log file and enabling debug output could be called as:

```
sanitize-logs --file nginx-log-file --type nginx --debug
```

[1] `https://github.com/tj/commander.js`

or with parameter shorthands:

```
sanitize-logs -f nginx-log-file -t nginx -d
```

5.9 Application Workflow

The application runs through these 4 stages:

1. Loading or connecting to log source (file or service)
2. Reading log
3. Parsing log data
4. Detecting sensitive data from parsed data item

This section provides insight to the implementation details of each stage.

5.9.1 Drivers for Log Engines

The develop application is designed to support multiple log engines (sources) such as Nginx, Apache or ElasticSearch. To allow managing multiple systems, the author has implemented **Driver Pattern**. `src/drivers` folder in source code is where each of these drivers are defined. Each driver file includes information on how to connect to these log sources, how to parse the logs into keys (defined as **fields**) and values and how to map these fields defined in the logs with the sensitive fields defined in the application. To accomplish these tasks, the author found many existing 3rd party libraries in NPM ecosystem[1]. For example, the author has used `@sematext/logagent`[47]; this module is originally created to collect logs from various sources and send them to the cloud-based log management service Sematext Logsene[2]. As defined in the Typescript type definition file `src/types.ts`, a driver exists of following properties:

```
interface Driver {
  name: string,
```

[1] `https://www.npmjs.com`
[2] `https://sematext.com/logsene`

```
  parseLine: Function, # Parses logs and maps fields
  sensitiveFields: SensitiveFields, # Provides sensitive fields
  detectSensitiveFields: Function # Match sensitive fields
}
```

Figure 3. PoC application Driver interface.

Accessing the logs could be a product specific protocol or a shared method. Accessing file logs is an example to a shared method. To define these shared methods, the author created `src/actions/file.ts` and `src/lib/file.ts` files, which define methods to check file access and write permissions, iterates through file and accomplishes main program features by calling the specified log driver.

5.9.2 Fields

The author also defined possible sensitive fields as **fields**. These fields are defined in the `src/config/compositeFields.ts` and `src/config/fieldTypes.ts` files. The author has defined two types of fields. Plain fields or just fields are simple definitions of sensitive fields that directly matches to only one field in a log. On the other hand, **composite fields** are fields that are encoded in various formats (i.e. JSON or query string) and include multiple fields and values. An example to composite fields could be URLs: in the query string part of the URLs there could be multiple keys and values. The definitions in `src/config/compositeFields.ts` file provides how to parse these composite fields into simpler fields and `findFieldTypesFromStrings` method in `src/lib/fields.ts` file matches these fields into defined sensitive fields. As for algorithmic complexity, these composite fields has quadratic complexity (`O(N^2)` in Big-O Notation), which might slow down log processing in larger files.

As defined in the types file, a field has following properties. An example field could be found at Appendix Code 1.

```
# src/lib/fields.ts
interface FieldType {
  label: String,
  sensitivity: Number,
```

```
  safe_values_regex?: FieldSafeValues,
  match_values?: Array<string>
}
```

Figure 4. PoC application fieldType interface.

5.9.3 Sensitivity and Safe Values

The author designed the application to allow users to specify a **sensitivity level** when matching and removing sensitive information from logs. From the least sensitive to most, the author defined the sensitivity fields as:

- 0. Public
- 1. PII, such as name, surname, city
- 2. Identifying, such as email, IP address and ID code
- 3. Secret, such as passwords and application secrets

In the application author developed a feature to provide safe values for a field to skip while scanning the logs. For example, private IP address blocks such as `10.0.0.0\/8` or usernames such as `root` or `admin`. These safe values could be defined in `safe_values_regex` property of a `FieldType` in a string or regular expression format.

5.10 Other Development Details

The folder structure of the application can be found at Appendix Code 2.

In the root folder of the application, there are also various files defining extra functionality. Of these, `editorconfig` file provides configuration for EditorConfig[1], a coding-style engine for multi-platform support. `.tool-versions` file provides support for `asdf-vm`[2] version manager.

TypeScript configurations are defined in `tsconfig.json` file. The author decided to use latest Node.js version to date (version 16.x) and EcmaScript 2019 as TypeScript tar-

[1] `https://editorconfig.org`
[2] `https://asdf-vm.com`

get. Using a newer version allowed the author to use newer JavaScript features such as `Object.fromEntries()` method. Before using and publishing, the source code in TypeScript is compiled to JavaScript using `tsc` compiler into `dist` folder. This folder is ignored in project source using `.gitignore` file.

5.11 Testing

The program provides two different testing options: manual and automated.

For manual testing, the author developed a test run feature. When the application is run with `--testrun` parameter, the application loads the sample Nginx access log located `test/fixtures` folder, scans the file for sensitive fields and display the results on the screen. An example output could be seen in Appendix Code 4.

For automated tests, the author used Jest to create unit tests. Test files are located next to their modules in `MODULENAME.test.ts` naming style. When the project source is downloaded or cloned and installed, `npm test` command could be run to run all automated tests. Jest also includes a code coverage utility. To run the code coverage for the application, `npm run coverage` command could be used. At the moment, the application has 100% code coverage, as could be seen at Appendix Code 5.

5.12 Result Analysis and Conclusion

In the end, the author created and published a working proof of concept application to detect sensitive fields and values in an Nginx log file. As defined in the limitations section of this chapter, the PoC application provides a very limited set of features. It currently allows only Nginx log files, detecting limited number of sensitive fields and runs read only. However, the application was designed in a way to allow adding support for other log types (engines) and field types easily and safely. For this application to become a universal tool, a decent number of these 3rd party support should exist. Application being published as permissive license to public can allow anyone to participate in the project and extend feature set.

Suggestions for possible future extensions and improvements are discussed in the future works section.

Both sample log file and automated unit tests display that the developed functionality is working according to defined specifications.

In the end, the application successfully displayed the practical copy of the desired solution.

6 Future Work

The prototype software was created to be a proof of concept. With more resources, this application could be improved with the features describe in this section. With enough improvements and features, this tool could be used by company IT teams and developers. These suggestions include limitations defined previously in this thesis.

6.1 Extending Log and Field Support

Plugin system could be extended in a way that it could support wider choice of features, such as different logging applications and data formats. More ready-to-use plugins could be written by the author, NPM could be used for hosting these plugins as public packages. With new plugins, other communication methods and protocols could also be supported, i.e. IPC, RPC, gRPC, ProtoBuff. Also, some of the existing drivers like `@sematext/logagent` could be re-written, for less 3rd party dependency, improved speed and improved security.

Sensitive data detection could be improved in many ways. In the future, this application could have more predefined rules for more fields. Also, various Machine Learning methods could be applied for improved successful detection. For example, `k-Anonymity` [15] could be used for calculating fields. Another way to improve field detection is to use 3rd party tools such as VirusTotal's YARA[1]. At the same time, more unit tests could be written for the existing rules to reduce possible false negatives and false positives. Another future work for sensitive fields is to define safe values for languages other than English, for international support.

6.2 Runtime

To increase the application performance, multiple threads could be added, using Node.js's Cluster API[2]. This API provides forking the process as many as number of threads available to the operating system, but does not provide memory sharing. Thus to implement such feature, log operations like scanning the log should be split into sub-threads and managed by so.

[1] `https://github.com/virustotal/yara`
[2] `https://nodejs.org/api/cluster.html`

To improve developer experience and platform support, Docker[1] support could be added, along with docker-compose.

6.3 Interface and Configuration

A graphical user interface (GUI) could be created for graphical configuration and monitoring. For this purpose, web-based desktop application engines such as Electron.js[2] should provide faster development, since the application is already written in Node.js, as was Electron.js. This graphical interface could also be written using front-end frameworks.

The application interface, configuration and output could be improved for more compatible and seamless CI/CD integration. Extra plugins could also be provided or written for commonly used CI/CD and monitoring tools like Jenkins[3], TeamCity[4], and SolarWinds[5].

In addition to a GUI, an interactive command-line interface (CLI) could also be developed. For the unattended CLI, feature for using config files to provide program parameters could be developed.

An external database (i.e. SQLite) could be used to store past scans, scan results and program parameters.

[1] `https://www.docker.com`
[2] `https://www.electronjs.org`
[3] `https://www.jenkins.io`
[4] `https://www.jetbrains.com/teamcity`
[5] `https://www.solarwinds.com`

7 Summary

First purpose of this thesis was to analyze sensitive data problem in logs. It was found that this weakness has been defined in as early as 2006 (CWE-532[19]). Regulations and standards addressing the safety of personal data date back to 1974 (Privacy Act), with more recent ones still being developed such as GDPR (2018[1]). Yet, the author found out this vulnerability still being an issue in 2021. The author studied various recent leaks and revealed the validity and importance of it, all being caused by unsanitized logs. It is proven that keeping sensitive data in logs still exists and overlooked.

This work also analyzed various existing solutions, approaches and products for this problem. The author analyzed and compiled a list of stages for solving and preventing this issue. It was found that these existing solutions are mostly language specific, insufficient, and partial. They are unstandardized and they provide only partial solution. Also, only few tools featured sensitive data removal from logs, and they support only a limited number of logging systems. Author also found few incomplete and old open source tools, and failed to see a recent, widely used solution. This study showed that a generic application framework for sensitive data removal is still missing in practice.

Combining existing and proposed solutions and the feedback from existing products, the author proposed an inclusive prototype application. Its requirements were set to be compatible, platform / language / logger agnostic, fast and configurable. A PoC (proof of concept) application has been developed to demonstrate and validate it. This application could be an starting point and inspiration to any related future works.

References

[1] Council of European Union. *Council regulation (EU) no 2016/679*. https://eur-lex.europa.eu/eli/reg/2016/679/oj. 2016.

[2] Chase Williams. *Data Leak: Unsecured Server Exposed Bing Mobile App Data*. https://www.wizcase.com/blog/bing-leak-research/. 2021. (Visited on 2021-04-21).

[3] Rui Zhou et al. "MobiLogLeak: A Preliminary Study on Data Leakage Caused by Poor Logging Practices". In: *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2020, pp. 577–581. DOI: 10.1109/SANER48275.2020.9054831.

[4] Jieming Zhu et al. "Learning to Log: Helping Developers Make Informed Logging Decisions". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. 2015, pp. 415–425. DOI: 10.1109/ICSE.2015.60.

[5] R. Gerhards. *The Syslog Protocol*. RFC 5424. RFC Editor2, 2009-03. URL: http://www.rfc-editor.org/rfc/rfc5424.txt.

[6] Qusay H. Mahmoud. *Getting Started with Java Message Service (JMS)*. Oracle. 2004. URL: https://www.oracle.com/technical-resources/articles/java/intro-java-message-service.html (visited on 2021-04-21).

[7] *Turn Your Log Files into Searchable Data Using Regex and the XML Classes*. Microsoft. 2011. URL: https://docs.microsoft.com/en-us/previous-versions/dotnet/articles/ms972965(v=msdn.10) (visited on 2021-04-21).

[8] Phillip M. Hallam-Baker and Brian Behlendorf. *Extended Log File Format*. W3C. 1996. URL: https://www.w3.org/TR/WD-logfile.html.

[9] *GELF*. Graylog, Inc. URL: https://docs.graylog.org/en/4.0/pages/gelf.html (visited on 2021-04-21).

[10] *IIS Log File Formats*. Microsoft. 2017. URL: https://docs.microsoft.com/en-us/previous-versions/iis/6.0-sdk/ms525807(v=vs.90) (visited on 2021-04-21).

[11] *Personally Identifiable Information (PII)*. Imperva. 2019. URL: https://www.imperva.com/learn/data-security/personally-identifiable-information-pii (visited on 2021-04-21).

[12] *Data Privacy*. Imperva. 2020. URL: https://www.imperva.com/learn/data-security/data-privacy/ (visited on 2021-04-21).

[13] Philippe Golle. "Revisiting the Uniqueness of Simple Demographics in the US Population". In: *Proceedings of the 5th ACM Workshop on Privacy in Electronic Society*. WPES '06. Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 77–80. ISBN: 1595935568. DOI: 10.1145/1179601.1179615. URL: https://doi.org/10.1145/1179601.1179615.

[14] Joe Crobak. *Seven Best Practices for Keeping Sensitive Data Out of Logs*. 2018. URL: https://medium.com/@joecrobak/seven-best-practices-for-keeping-sensitive-data-out-of-logs-3d7bbd12904 (visited on 2021-04-21).

[15] Pierangela Samarati and Latanya Sweeney. *Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression*. Tech. rep. 1998.

[16] The MITRE Corporation. *CVE*. https://cve.mitre.org. (Visited on 2021-04-21).

[17] The MITRE Corporation. *CVE Numbering Authorities*. https://cve.mitre.org/cve/cna.html. 2021. (Visited on 2021-04-21).

[18] The MITRE Corporation. *CWE-Compatible Products and Services*. https://cwe.mitre.org/compatible/compatible.html. 2020. (Visited on 2021-04-21).

[19] The MITRE Corporation. *CWE-532: Insertion of Sensitive Information into Log File*. https://cwe.mitre.org/data/definitions/532.html. 2021. (Visited on 2021-04-21).

[20] The MITRE Corporation. *CWE-533: Information Exposure Through Server Log Files*. https://cwe.mitre.org/data/definitions/533.html. 2021. (Visited on 2021-04-21).

[21] The MITRE Corporation. *CWE-534: Information Exposure Through Debug Log Files*. https://cwe.mitre.org/data/definitions/534.html. 2021. (Visited on 2021-04-21).

[22] The MITRE Corporation. *CWE-542: Information Exposure Through Cleanup Log Files*. https://cwe.mitre.org/data/definitions/542.html. 2021. (Visited on 2021-04-21).

[23] The MITRE Corporation. *CWE-117: Improper Output Neutralization for Logs*. https://cwe.mitre.org/data/definitions/117.html. 2021. (Visited on 2021-04-21).

[24] The MITRE Corporation. *CWE-312: Cleartext Storage of Sensitive Information*. https://cwe.mitre.org/data/definitions/312.html. 2021. (Visited on 2021-04-21).

[25] Parag Agrawal. *Keeping your account secure*. https://blog.twitter.com/official/en_us/topics/company/2018/keeping-your-account-secure.html. 2018. (Visited on 2021-04-21).

[26] Mansoor Iqbal. *Twitter Revenue and Usage Statistics (2020)*. https://www.businessofapps.com/data/twitter-statistics/. 2021. (Visited on 2021-04-21).

[27] LeakedSource. *LeakedSource Analysis of Twitter.com Leak*. https://archive.is/MWMIN. 2016. (Visited on 2021-04-21).

[28] Marianne Kolbasuk McGee. *32.8 Million Twitter Credentials May Have Been Leaked*. https://www.bankinfosecurity.com/33-million-twitter-credentials-may-have-been-leaked-a-9187. 2016. (Visited on 2021-04-21).

[29] Pedro Canahuati. *Keeping Passwords Secure*. https://about.fb.com/news/2019/03/keeping-passwords-secure/. 2019. (Visited on 2021-04-21).

[30] Facebook, Inc. *Facebook Reports Fourth Quarter and Full Year 2019 Results*. https://investor.fb.com/investor-news/press-release-details/2020/Facebook-Reports-Fourth-Quarter-and-Full-Year-2019-Results/default.aspx. 2020. (Visited on 2021-04-21).

[31] vpnMentor. *Report: No-Log VPNs Reveal Users' Personal Data and Logs*. https://www.vpnmentor.com/blog/report-free-vpns-leak/. 2021. (Visited on 2021-04-21).

[32] *A secure web is here to stay*. Chromium Blog. 2018. URL: https://blog.chromium.org/2018/02/a-secure-web-is-here-to-stay.html (visited on 2021-04-21).

[33] *RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. Internet Engineering Task Force (IETF). 2014. URL: https://tools.ietf.org/html/rfc7231#section-4.3.3 (visited on 2021-04-21).

[34] *Drop data using NerdGraph - Example drop rules*. Sentry.io. URL: https://docs.newrelic.com/docs/accounts/accounts/data-management/drop-data-using-nerdgraph/%5C#example-rules (visited on 2021-04-21).

[35] *Scrubbing Sensitive Data - JavaScript SDK*. Sentry.io. URL: https://docs.sentry.io/platforms/javascript/data-management/sensitive-data/ (visited on 2021-04-21).

[36] *Server-Side Data Scrubbing*. Sentry.io. URL: https://docs.sentry.io/product/data-management-settings/server-side-scrubbing/ (visited on 2021-04-21).

[37] *Advanced Data Scrubbing*. Sentry.io. URL: https://docs.sentry.io/product/data-management-settings/advanced-datascrubbing/ (visited on 2021-04-21).

[38] *Anonymize data*. Splunk. 2021. URL: https://docs.splunk.com/Documentation/Splunk/8.1.3/Data/Anonymizedata#Anonymize_data_with_a_sed_script (visited on 2021-04-21).

[39] *scrub - Search Reference*. Splunk. 2020. URL: https://docs.splunk.com/Documentation/Splunk/8.0.1/SearchReference/Scrub (visited on 2021-04-21).

[40] *Masking sensitive data in the Splunk Data Stream Processor*. Splunk. 2020. URL: https://docs.splunk.com/Documentation/DSP/1.2.0/User/Masking (visited on 2021-04-21).

[41] Adam Slagell, Kiran Lakkaraju, and Katherine Luo. "FLAIM: A Multi-Level Anonymization Framework for Computer and Network Logs". In: *Proceedings of the 20th Conference on Large Installation System Administration*. LISA '06. Washington, DC: USENIX Association, 2006, p. 6.

[42] *Removing sensitive data from a repository*. GitHub Docs. URL: https://docs.github.com/en/github/authenticating-to-github/removing-sensitive-data-from-a-repository (visited on 2021-04-21).

[43] *Usage statistics of Unix for websites*. W3C. 2020. URL: https://w3techs.com/technologies/details/os-unix (visited on 2021-04-21).

[44] *TIOBE Index for April 2021*. TIOBE Software BV. 2021. URL: https://www.tiobe.com/tiobe-index (visited on 2021-04-21).

[45] *Module Counts*. URL: http://www.modulecounts.com/) (visited on 2021-04-21).

[46] *Why the Affero GPL*. GNU. 2015. URL: https://www.gnu.org/licenses/why-affero-gpl.html (visited on 2021-04-21).

[47] *@sematext/logagent NPM homepage*. URL: https://www.npmjs.com/package/@sematext/logagent (visited on 2021-04-21).

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Yasin Aydin (date of birth 29.03.1986)

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Safeguarding Sensitive Data in Logs" , supervised by Kaido Kikkas
   (a) to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   (b) to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

17.05.2021

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Proof of Concept Application Code Snippets

```
export const fieldTypes: FieldTypes = {
  username: {
    label: "Username",
    sensitivity: 2,
    match_values: ["user", "user_name"],
    safe_values_regex: [/admin.*/, "root", "user"],
  },
}
```

Code 1. Example sensitive field definition as fieldType.

```
bin/app.js     # For running application globally
src/           # Program code
  actions/     # Shared log access types
  config/      # Field definitions
  drivers/     # Log engine definitions
  lib/         # Helper utilities
  app.ts       # Main entry point
  types.ts     # Type definitions
test/fixtures/ # Sample log files
```

Code 2. PoC application folder structure.

```
# bin/app.js
#!/usr/bin/env node
require('../dist/app.js')
```

Code 3. bin/app.js.

```
$ sanitize-logs --testrun --debug
logsanitizer@1.0.20
[debug] Debug mode enabled
[debug] Started in unattended mode
[info]  Source file is: .../sanitize-logs/test/fixtures/nginx-accesslogs.txt
[debug] Source type is: file
[debug] Checking log type... nginx
[debug] Detecting driver... logAgent
[debug] Checking file permissions... OK
[debug] Reading file contents... OK
[debug] Analyzing log file and determining fields... OK
Scan is done.
Lines scanned: 30
Sensitive values found: 7
```

```
Unique sensitive values for each field is below:
Field name  Sensitive Data Type  Field Sensitivity   Sensitive Value
----------  -------------------  ------------------  ---------------
path        Password             3: Secret           MyPassw0rd
referer     Password             3: Secret           MyPassw0rd
client_ip   IP Address           2: Unique Personal  1.2.3.4
```

Code 4. Sample log scan with –testrun parameter.

```
$ npm run coverage
> sanitize-logs@1.0.0 coverage
> jest --coverage
 PASS  src/lib/fields.test.ts
 PASS  src/drivers/logagent.test.ts
----------------------|---------|----------|---------|---------|-
File                  | % Stmts | % Branch | % Funcs | % Lines |
----------------------|---------|----------|---------|---------|-
All files             |     100 |      100 |     100 |     100 |
 config               |     100 |      100 |     100 |     100 |
  compositeFields.ts  |     100 |      100 |     100 |     100 |
  fieldTypes.ts       |     100 |      100 |     100 |     100 |
 drivers              |     100 |      100 |     100 |     100 |
  logagent.ts         |     100 |      100 |     100 |     100 |
 lib                  |     100 |      100 |     100 |     100 |
  fields.ts           |     100 |      100 |     100 |     100 |
----------------------|---------|----------|---------|---------|-
Test Suites: 2 passed, 2 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        2.969 s, estimated 3 s
Ran all test suites.
```

Code 5. Unit test code coverage.

Appendix 3 – Other Used Code and Snippets

```
const { createLogger, format, transports } = require('winston');
const { sanitize } = require('./some-sanitizer');
const formatter = format.printf(({ level, message }) => {
  return `${level}: ${sanitize(message)}`;
});
const logger = createLogger({
  format: formatter,
  transports: [new transports.Console()]
});
logger.info("user_logged_in: ${sensitiveUserData}");
```

Code 6. Example for using Winston formatter for log sanitization.