Analysis of software vulnerabilities through historical data

MAGNUS TÖRNQUIST
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Analysis of software vulnerabilities through historical data

Magnus Törnquist
magnusvmt@gmail.com

Department of Electrical and Information Technology
Lund University

Supervisor: Martin Hell

Assistant Supervisor: Jonathan Sönnerup

Examiner: Thomas Johansson

June 29, 2017

Printed in Sweden
Tryckeriet i E-huset, Lund

Popular science summary

Lately there has been increasing media coverage of cyber crime, especially in relation to the elections in France and the United States. Every day information is being stolen from governments, businesses and private citizens. Information that can be sold, used for blackmail or for other nefarious purposes. Commonly this information is obtained through exploiting vulnerabilities in software. A vulnerability is essentially a bug in the code and they are very hard to avoid, especially in large complex programs. Having vulnerabilities in software is inevitable and software is everywhere: in every computer, router, webcam, mobile device and even in some coffeemakers. As long as these devices are connected an intruder has a wide variety of options on how to attack a network and the fast growth of Internet of Things (IoT) has lead to a huge amount of new devices on networks all over the world. This reality means that larger organizations have to spend a lot of time making sure all their software is updated and keeping track of potential breaches. This also means that it is very important for the software developer to maintain their code and patch any discovered vulnerabilities quickly. So how does an organization, the developer of an IoT product or a regular user choose which software to use if they are concerned about software security and is there a way to help them do it? That is what this thesis explores.

The general idea is to to find a way to measure the security levels of software. This would make comparisons between software easier for end-users. In order to accomplish this we gather and analyze a large amount of software data to gain an understanding of which metrics are important and why. This data includes all historical vulnerabilities for the software, their severity and any public exploits that might exist to make use of these vulnerabilities. It also includes other data about the software such as popularity and how often the developers release new version of the software. A total of 44 software products were chosen to include a wide variety of software, both in type (what the software was designed to do) and popularity. Since this is a large amount of software and the data gathering process is slow, the analysis is divided into two parts. One part is a broad comparison between the software based on data that has been automatically collected, the other part is a detailed comparison between five software using data that has been manually gathered. We also look at previous research on the subject and give an overview of interesting ideas and concepts.

The results of this analysis are used to present suggestions for how these metrics could currently be implemented. But more importantly the results are used to draw conclusions and suggest new ways in which this problem can be approached in the future to make further progress in the field of software security metrics. One of the conclusions made is that machine learning could be used in the future to solve some of the issues that presented themselves during analysis, however a machine learning strategy requires huge amounts of data that is not easily available yet.

Abstract

Software security has become an increasingly hot topic of debate during the last few years of cyberattacks, especially now that we are entering the era of Internet of Things. How does the developer of a product decide which software to include from a security perspective and is it possible to create a tool for software comparison that the developer could use for this purpose? The aim of this thesis is to investigate which metrics are available for measuring the overall level of security in software and suggest ways in which these metrics can be used. This study is done partly by reviewing previous research on software security metrics and partly by analyzing metrics in different categories such as general metrics about the software, metrics based on historical data and more detailed metrics about the vulnerabilities in the software. A small survey is also performed to gather the opinions about some of these metrics from potential end-users of a scoring system. Ideas for scoring systems that can use these metrics are suggested, however no weights for these metrics are determined. The conclusion is that under current circumstances creating a good automated scoring system is difficult due to a lack of data, however there are exciting opportunities for continued research and ideas for new approaches are presented.

Acknowledgements

First and foremost I would like to offer my sincerest gratitude to my supervisor Martin Hell and assistant supervisor Jonathan Sönnerup at the Department of Electrical and Information Technology, Lund University, for allowing me to work on this thesis and for aiding me throughout both planning and execution.

I would also like to thank Prof. Krzysztof Nowicki at the Department of Statistics, Lund University, for his advice and for an interesting discussion on the limits of statistics.

Furthermore, I would like to extend my appreciation to each and every one of the respondents to the survey, your participation was very valuable.

Last but not least, I would like to thank my family and friends for the continued support and encouragement they have given me during the entirety of my education.

Contents

List of Figures

List of Tables

Abbreviations

ALM	Alhazmi Malaiya Logistic
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DDoS	Distributed Denial of Service
IoT	Internet of Things
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
OWASP	Open Web Application Security Project
PVL	Product Value Loss
VDM	Vulnerability Discovery Model
VM	Virtual Machine
VRSS	Vulnerability Rating and Scoring System
VTEM	Vulnerability Timeline and Exposure Metrics
WIVSS	Weighted Impact Vulnerability Scoring System

Chapter 1

Introduction

Internet of Things (IoT) has been growing at a very fast pace, predictions show that 26 billion IoT devices will exist in 2020 [1]. Together these devices, that are usually online at all times, have access to a huge amount of bandwidth and computing resources which means that when compromised even a small proportion of these devices can create a powerful botnet. Botnets are currently a booming business where the operators lease out botnet services to anyone willing to pay. On top of that military organizations have been expanding their focus on cyberwarfare, nation states are with increasing frequency using illegally obtained data for geopolitical strategy, financial institutions are being heavily targeted and millions of private citizens have their personal information and identities stolen every year. With all this in mind there is plenty of motivation to explore the field of software security.

The most common approach for attacks on these devices is through their software. The software can be compromised remotely either because of misconfiguration (default passwords, incorrect privileges etc.) or through exploiting vulnerabilities which are essentially bugs in the code of the software. One vulnerability that, with good reason, received widespread media attention was Heartbleed [2]. Heartbleed was a vulnerability that allowed an attacker to send malformed packets to a server running OpenSSL and the software would respond by leaking data from memory, data that could be highly sensitive like private encryption keys. Vulnerabilities like this are unavoidable in software and what is important is how the developer of the software handles them.

If we look at what we know about software vulnerabilities from the perspective of a user or a developer of IoT products, how are they to know which software developer handles vulnerabilities in the best possible way? With other words, how do they choose the best software from a security perspective? That is the fundamental question that this thesis is built upon. The following two sections of this chapter will further detail the objectives of the thesis and present an outline of the report structure.

1.1 Problem description and research motivation

The idea behind this thesis is to analyze software and attempt to find a way to compare software from a security perspective which could help developers choose a software to use in their project. The thesis has the following three objectives:

1. Examine which metrics are available for measuring software security and evaluate their value.
2. Study available scoring systems and other methods of measuring software security.
3. Develop and suggest scoring systems based on what has been learned.

The scoring systems should give a user the ability to compare a set of software and help the user choose the best from a security perspective.

1.2 Outline

The report is organized as follows:

- *Chapter 1 Introduction:* Introduces the reader to the problem, specifies the objectives for the thesis and contains an outline of the report.
- *Chapter 2 Background:* Contains information about what a vulnerability is, where they are found and different methods of measuring them. This chapter also has information on what statistical methods and tools were used during analysis.
- *Chapter 3 Previous metric research:* Provides an overview of previous academic works on the subject of software security metrics.
- *Chapter 4 Method:* Presents an outline of the analysis process. It explains how the software was chosen, how the data was gathered, what variables are focused on and how they are defined.
- *Chapter 5 Results:* Compilation of analysis results for each of the four parts of analysis. The results are presented with graphs and comments.
- *Chapter 6 Practical application of the results:* Provides ideas for using the analysis results and other gathered data in a practical way.
- *Chapter 7 Conclusions and future research:* Summarizes the findings, concludes the thesis and contains ideas for where future work can be focused.

Chapter 2

Background

The (in)security of a software is analyzed through the collection of vulnerabilities the software has had during its lifespan, therefore understanding vulnerabilities is paramount for the task at hand. This chapter covers how a vulnerability is defined, where vulnerabilities are found and introduces different metrics used to measure them. It also includes the statistical methods and tools that were used during the analysis.

2.1 Vulnerabilities and weaknesses

The Wikipedia article on vulnerability(computing) [3] claims that a vulnerability is a weakness. However in the context of this thesis a vulnerability will be defined as a specific bug in a specific software which can be abused in an unintended manner to potentially cause a negative impact to the user of the software. The reason for this distinction is that a weakness is used as a broader term that describes design or implementation errors that can lead to a vulnerability. While the thesis will be fully focused on vulnerabilities this section describes the most common way of enumerating both.

2.1.1 CVE

Common Vulnerabilities and Exposures (CVE) is a dictionary of publicly known information security vulnerabilities and exposures run by the MITRE Corporation [4]. It is sponsored by the U.S. Department of Homeland Security and is now the industry standard for identifying vulnerabilities with what they call CVE Identifiers (eg. CVE-1999-0400). These identifiers are given to a vulnerability by a CVE Numbering Authority (CNA) together with a description of the vulnerability. Many of the larger software vendors act as their own CNA and therefore add their own vulnerabilities to the dictionary [5]. By their own disclaimer [6] CVE is not a vulnerability database. They do however feed into the U.S. National Vulnerability Database (NVD) [7] that adds some additional analysis like metrics. In this report a reader will see wordings like ”a CVE” or ”the CVEs”, which are implicative referrals to the vulnerability entries in the CVE dictionary.

2.1.2 CWE

Common Weakness Enumeration (CWE) has the same function as CVE but it enumerates weaknesses [8] instead. According to their FAQ [9] the difference between weaknesses and vulnerabilities is that a software weakness is an error that can lead to a vulnerability. Therefore the CWE identifies the underlying problem rather than a specific software vulnerability (eg. CWE-89: SQL injection). This is why the definitions made earlier are important, while ”Missing Authorization ” is a weakness ”Missing Authorization in software X” is a vulnerability.

2.2 Information sources

A vulnerability can be published almost anywhere, depending on who discovers it. This means that there is no go-to place for all vulnerabilities and a manual search of the web is recommended when trying to find a vulnerability in a specific software.

2.2.1 Databases

Apart from the already mentioned U.S. National Vulnerability Database [7] three other big databases are Vulndb [10], CVE Details [11] and IBM xForce Exchange [12].

Vulndb is a vulnerability database that Risk Based Security provides to its paying customers. It maps all vulnerabilities with CVE identifiers and more. According to vulndbs statistics they reported 14185 vulnerabilities in 2015, over 6000 more vulnerabilities than NVD/CVE [13]. They use their own version of Common Vulnerability Security System (CVSS) metrics (more on this in Section 2.3.1), as well as Vulnerability Timeline and Exposure Metrics (VTEM). ”VTEM uses vulnerability timeline data such as discovery, disclosure and patch availability dates to measure the vendor’s overall responsiveness to correcting vulnerabilities” [14] VTEM was launched in 2015 and the specifics of VTEM would be of great relevance to this thesis, unfortunately they are not public.

CVE Details works in the same vein as NVD, it uses the CVE dictionary and adds some detail and analysis about the vulnerabilities. It also provides RSS feeds which makes it easier to stay on top of vulnerabilities for a chosen set of software.

IBM xForce Exchange was created in 2015 to consolidate threat intelligence from multiple sources and make it easily accessible both through a web interface and through a RESTful API.

In 2010 Messacci and Nguyen analyzed and compared vulnerability databases by focusing on Mozilla Firefox [15]. Their work shows that many sources did not have information about exploit publishing date and their conclusions support the claim that using any single source for vulnerabilities does not give good coverage.

2.2.2 Mailing lists

Mailing lists are a great source for continuous updates on new vulnerabilities and discussions around them. Seclists [16] has a comprehensive list of security mailing lists. Two good ones for vulnerability updates are Bugtraq [17] and Full Disclosure [18]. Founded in 1993, Bugtraq is one of the oldest active mailing lists for vulnerabilities. Vulnerabilities disclosed here must not always have received a CVE and is therefore a good source for 0-day (zero-day) vulnerabilities. A 0-day vulnerability is a vulnerability that has not been disclosed in public previously. Full Disclosure was restarted in 2014 after the original owner did not want to run it anymore. Even though it is not as popular as Bugtraq some 0-days have been published there in the past and it is a good place to discuss an exploit further in dept.

2.2.3 Other sources

Other possible sources for vulnerabilities are Github repositories and Twitter. Some Twitter accounts run bots that automatically post vulnerabilities, these can be valuable to follow. If the potentially vulnerable software has a repository on Github looking at the bug and discussion section there might be very useful.

2.3 Vulnerability metrics

The National Institute of Standards and Technology (NIST), the institute behind CVSS, has developed a standard to ”assist in the development, selection, and implementation of measures to be used at the information system and program levels” [19]. It lists the following four factors that must be considered when developing a security metric:

- Measures must yield quantifiable information (percentages, averages, and numbers)
- Data that supports the measures needs to be readily obtainable
- Only repeatable information security processes should be considered for measurement
- Measures must be useful for tracking performance and directing resources

This standard described quantitative metrics where a score is set on the basis of a rigid mathematical process. It is important to distinguish these from qualitative rating systems that are often used on vulnerabilities. According to George Jelen of the International Systems Security Engineering Association good metrics are those that are SMART: Specific, Measurable, Attainable, Repeatable and Time-dependent [20].

2.3.1 CVSS version 2

Common Vulnerability Scoring System (CVSS) is an open and free industry standard for measuring the severity of vulnerabilities in software. Version 2 of CVSS [21] was released in June 2007 and is a very common security score used for vulnerabilities. It was replaced later by version 3 which is described in Section 2.3.2.

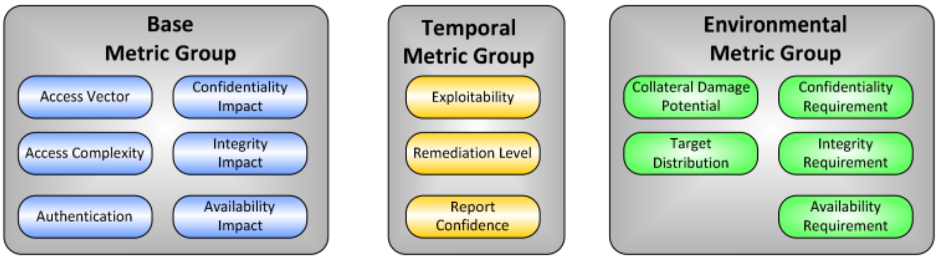


Figure 2.1: CVSS v2 subgroups and metrics from the CVSS v2 user guide [21]. Copyright FIRST, used with permission.

CVSS calculates three different scores: a base score, a temporal score and an environmental score as seen in Figure 2.1. The base score is calculated based on qualities the vulnerability has that do not change over time or in different environments, this is the only mandatory score. The temporal score is based on characteristics which change over time, for example if an exploit or a fix exists. This score is described in Section 2.3.3. The environmental score changes depending on the environment and how serious the impact of an attack on the vulnerable system would be.

The CVSS v2 base score is a score between 0 and 10 rounded up to the nearest decimal. According to the specification it is calculated as follows:

$$\begin{aligned} BaseScore = \mathrm{round\_to\_1\_decimal}(((0.6 \cdot Impact) + (0.4 \cdot Exploitability) \\ - 1.5) \cdot \mathrm{f}(Impact)) \end{aligned} \tag{2.1}$$

$$Exploitability = 20 \cdot AccessVector \cdot AccessComplexity \cdot Authentication \tag{2.2}$$

$$Impact = 10.41(1-(1-ConfImpact)(1-IntegImpact)(1-AvailImpact)) \tag{2.3}$$

$$\mathrm{f}(Impact) = \begin{cases} 0, & \text{if } Impact = 0 \\ 1.176, & \text{otherwise} \end{cases} \tag{2.4}$$

2.3.1.1 Access Vector

Access Vector can assume three different values depending on the vulnerability: Local(0.395), Adjacent Network(0.646) and Network(1.0). It is set to Network if the vulnerability is possible to exploit remotely over the internet (OSI layer 3), to Adjacent Network if the exploit requires an attacker to be closer to the target (can not be performed through a router), to Local if the vulnerability is not bound to the network stack and requires the ability to read/write to the system.

2.3.1.2 Access Complexity

Access Complexity can assume three values: High(0.35), Medium(0.61) and Low(0.71). It is set to Low if an attacker can expect the exploit to be successful on most systems without any special preparation. It is set to Medium if there are some additional requirements for access, such as an uncommon configuration. It is set to High if the exploit requires the attacker to do reconnaissance and tailor the exploit after the target.

2.3.1.3 Authentication

Authentication is a metric that measures what privileges an attacker needs before exploiting the vulnerability. It can assume three values: None(0.704), Single(0.56) and Multiple(0.45). None if there are no requirements for the attacker to authenticate. Single if the attacker must authenticate once to proceed with the exploitation. Multiple if the attacker needs to authenticate multiple times, even if using the same credentials.

2.3.1.4 Confidentiality

Confidentiality and the two remaining sub-metrics all deal with the impact a vulnerability can have on the system. Confidentiality measures the importance of the data that can be stolen by an attacker by exploiting the vulnerability. It can assume three values: Complete(0.660), Partial(0.275) and None(0.0). Complete if the attacker gains access to all the data in the impacted component or if the stolen data is of critical importance (like a root password or encryption keys). None if the attacker does not gain access to any restricted data. Partial if the attacker gains access to some restricted data but it is constrained in some way and not critical to the impacted component.

2.3.1.5 Integrity

This metric measures the integrity impact of an attack using the vulnerability. Integrity in this case means how trustworthy is the data on the impacted device after an attack. Can it be trusted to be the same as before the attack? The Integrity metric can assume three values: Complete(0.660), Partial(0.275) and None(0.0). Complete if the attacker has gained the ability to modify all the data on the impacted component or if the data that the attacker is able to modify present a clear danger to the component. Partial if the data the attacker can modify does not have a serious impact on the component. None if the attacker can not modify any data on the impacted component.

2.3.1.6 Availability

Availability measures the impact an attack has on the availability of the impacted component, in other words, if the resources of the impacted components can be accessed. A loss of availability can be sustained (while the attack is ongoing) or persistent (the availability loss persist after the attack). The availability metric

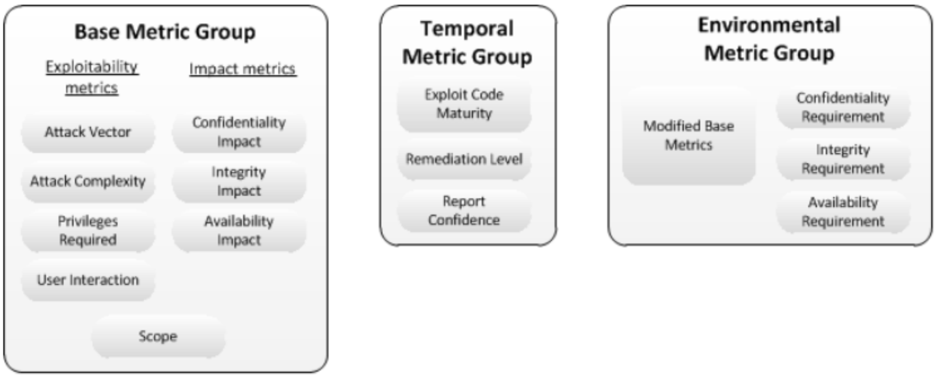


Figure 2.2: CVSS v3 subgroups and metrics from the CVSS v3 user guide [22]. Copyright FIRST, used with permission.

can assume three values: Complete(0.660), Partial(0.275) and None(0.0). This is set to Complete if the vulnerability is possible to exploit in such a way that the attacker gains full control over the availability of the impacted component or if the control the attacker gains can restrict resources that are of critical importance. It is Partial, if exploiting the vulnerability impacts performance but does not allow an attacker to completely cut off access to the component even if the exploit is used multiple times. None, if the vulnerability does not impact the availability of the component.

2.3.2 CVSS version 3

The newest version of CVSS, version 3, was released in June 2015 [22]. It looks rather similar to version 2 as can be seen in Figure 2.2.

One of the big changes from the previous version is that version 2 always measured impact on the host platform while version 3 measures the impact on what they call the ”impacted component”. The impacted component can be any software, hardware, or networking component. This is represented by a new metric called Scope. Scope will be explained in greater detail in Section 2.3.2.5 Other changes are rather minor like name changes of metrics and values. The CVSS v3 base score is described below for comparison.

The CVSS v3 base score is a score between 0 and 10 rounded up to the nearest decimal. According to the specification it is calculated as follows:

$$BaseScore = \begin{cases} Impact + Exploitability, & \text{if Scope=Unchanged} \\ 1.08(Impact + Exploitability), & \text{if Scope=Changed} \end{cases} \tag{2.5}$$

$$\begin{aligned} Exploitability = 8.22 \cdot AttackVector \cdot AttackComplexity \\ \cdot PrivilegesRequired \cdot UserInteraction \end{aligned} \tag{2.6}$$

$$Impact = \begin{cases} 6.42ISC_{base}, & \text{if Scope=Unchanged} \\ 7.52(ISC_{base} - 0.029) - 3.25(ISC_{base} - 0.02)^{15}, & \text{if Scope=Changed} \end{cases} \tag{2.7}$$

$$ISC_{base} = 1 - [(1 - Confidentiality)(1 - Integrity)(1 - Availability)] \tag{2.8}$$

Table 16 (page 20) in the specification [22] contains all possible numeric values for these sub-metrics. Short explanations of them follow here:

2.3.2.1 Attack Vector

Attack Vector can assume four different values depending on the vulnerability: Network(0.85), Adjacent(0.62), Local(0.55) and Physical(0.2). CVSS v3 added the new value called Physical which the attack vector is set to if the vulnerability can only be exploited with physical access to the vulnerable system.

2.3.2.2 Attack Complexity

Attack Complexity can assume two values: High(0.44) and Low(0.77). In CVSS v3 the value Medium was removed while High and Low are still defined in the same way as in version 2.

2.3.2.3 Privileges Required

Privileges Required the new version of the Authentication sub-metric in CVSS version 2. It is a metric for what privileges an attacker needs before exploiting the vulnerability. It can assume three values: None(0.85), Low(0.68 if Scope = Changed, 0.62 otherwise) and High(0.50 if Scope = Changed, 0.27 otherwise). None if the attacker does not need any privileges at all. Low if the attacker needs ”basic user” privileges. High if the attacker needs administrative privileges (root).

2.3.2.4 User Interaction

User interaction is a new addition in CVSS v3. It can only assume two values: None(0.85) and Required(0.62). None if the vulnerability can be exploited by an attacker without needing a user or user process of the targeted system to interact in any way. It is set to Required if the attacker needs a user or a user process to participate in the attack, for example by executing something.

2.3.2.5 Scope

Scope can assume two values: Changed or Unchanged. If the impacted component is not governed by the same authorization scope as the vulnerable component, a scope change has occurred. The authorization scope are the privileges to computing resources (CPU, memory etc.) that are granted to the vulnerable component by an authorization authority (e.g. an application, operating system or sandbox). For example, a scope change occurs if a vulnerability in a virtual machine (VM) compromises the underlying operating system (OS) because the VM and the OS are seperate authorization authorities (one grants privileges to the VM users and the other to the host system where the VM is running). Put simply, a scope change occurs when breaking out of a sandbox environment. Since sandbox environments are very common today (for example in modern web browsers) it makes the addition of Scope the most important change from CVSS version 2 to version 3.

2.3.2.6 The impact metrics

The three remaining sub-metrics: Confidentiality, Integrity and Availability all deal with the impact a vulnerability can have on the system. The only difference between CVSS v2 and v3 is the name and numeric values that these impact metrics can be set to. They can all assume the same three values: High(0.56), Low(0.22) and None(0.0). They directly correspond to the values called Complete, Partial and None in CVSS version 2.

2.3.2.7 CVSS in IoT software

CVSS has received criticism for being deceiving when applied on software used in IoT. Since it was designed with classical computing software in mind, the score does not take the potential for ”real” damage to property or even human life into account [23].

2.3.3 CVSS temporal metrics

As mentioned in Section 2.3.1 CVSS includes an optional score called Temporal Score that can change with time. Even though it is seldom included in vulnerability databases and security advisories it is an interesting score to consider when comparing software security. According to the CVSS v2 and v3 documentation [21] [22] it consists of three metrics: Exploitability, Remediation Level and Report Confidence. The Temporal Score ranges from 0 to 10 but cannot go above the base score, it is calculated like this:

$$\begin{split} TemporalScore = \text{round\_to\_1\_decimal}(BaseScore \cdot Exploitability \\ \cdot RemediationLevel \cdot ReportConfidence) \end{split} \tag{2.9}$$

The numeric values these sub-metrics can take on differs a little between CVSS v2 and v3, the version 2 numeric values have been used in this thesis. The version 3 numerics can be found in the CVSS v3 specification. Short explanations of the metrics follow below:

2.3.3.1 Exploitability

Exploitability measures how difficult a vulnerability can be exploited using the exploits available at the time of measuring. It can assume five different values:

Unproven(0.85), Proof-of-Concept(0.9), Functional(0.95), High(1.00) and Not Defined(1.00). It is set to Unproven if no exploit code is available for example when an exploit is theoretical. It is set to Proof-of-Concept if an exploit is only a proof-of-concept. A proof-of-concept is an exploit not functional in all situations and may require substantial modification by a skilled attacker. It is set to Functional if exploit code is available. The exploit should work for most vulnerable components. It is set to High if the vulnerability is exploitable automatically (for example by a worm), or if no exploit is required and details on how to exploit the software manually is widely available. It is set to Not Defined if the metric is to be ignored.

2.3.3.2 Remediation Level

Remediation Level measures if and how difficult it is to fix a vulnerability. It can also assume five different values: Official Fix(0.87), Temporary Fix(0.90), Workaround(0.95), Unavailable(1.00) and Not Defined(1.00). It is set to Official Fix if a complete vendor solution is available. Either the vendor has issued an official patch, or an upgrade is available. It is set to Temporary Fix if there is an official temporary fix available, for example a temporary hotfix, tool, or workaround. It is set to Workaround if there is an unofficial, non-vendor solution available. It is set to Unavailable if there is no solution available. It is set to Not Defined if the metric is to be ignored.

2.3.3.3 Report Confidence

Report Confidence measures the likelihood that the measured vulnerability actually exists and works as reported by evaluating the source of the vulnerability report. This metric can assume four values: Unconfirmed(0.90), Uncorroborated(0.95), Confirmed(1.00) and Not Defined(1.00). It is set to Unconfirmed if there is a single unconfirmed source or multiple conflicting reports. There is little confidence in the validity of the reports. An example is a rumor that surfaces from the hacker underground. It is set to Uncorroborated if there are multiple non-official sources. These reports should not be too conflicting but can vary on the smaller technical details. It is set to Confirmed if the vulnerability has been acknowledged by the vendor or if the vulnerability has been confirmed by external sources such as a functional exploit or a proof-of-concept. It is set to Not Defined if the metric is to be ignored.

2.3.4 CWSS

Common Weakness Scoring System (CWSS) is similar to CVSS but measures software weaknesses instead of vulnerabilities. It gives a score from 0 to 100 based on a set of sub-metrics [8]. Since this thesis is focused on vulnerability metrics it will not go into detail about scoring systems for software weaknesses.

2.3.5 CERT-CC severity metric

Previous to 2012-03-27 the CERT Coordination Center used a self-developed vulnerability metric [24]. This metric is on a non-linear scale from 0 to 180 and measures the severeness of a vulnerability by the following seven factors:

- Is information about the vulnerability widely available or known?
- Is the vulnerability being exploited?
- Is the Internet Infrastructure at risk because of this vulnerability?
- How many systems on the Internet are at risk because of this vulnerability?
- What is the impact of exploiting the vulnerability?
- How easy is it to exploit the vulnerability?
- What are the preconditions required to exploit the vulnerability?

These factors affect the final score with different weights.

2.3.6 SANS vulnerability analysis scale

The SANS vulnerability analysis scale is a now defunct vulnerability scale that was developed and used by the SANS Institute. It divided vulnerabilities into four priorities: Critical, High, Moderate and Low. Elisa Bertino et al. describe the scale in a book from 2009 [25] and provide the following list of some key factors used in the scale:

- The diffusion of the affected product.
- Whether the vulnerability affected a server or client system.
- Whether the vulnerability is related to default configurations/installations.
- The IT assets affected (e.g. databases, e-commerce servers).
- The network infrastructure affected (DNS, routers, firewalls).
- The public availability of exploit code.
- The difficulty in exploiting the vulnerability.

2.3.7 Qualitative rating systems

Some larger software vendors have their own qualitative rating systems where they categorize the severity of vulnerability based on certain criteria. Some examples are Microsoft [26] and Redhat [27] whose rating systems are very similar. They divide vulnerabilities into four priorities:

- Critical: A vulnerability that can be exploited without user interaction and can therefore be exploited automatically by worms.
- High: A vulnerability that can easily compromise the confidentiality, integrity, or availability of resources.
- Moderate: A vulnerability that could compromise the confidentiality, integrity, or availability of resources but is mitigated by some factor such as authentication or unlikely configurations.

- Low: This rating is given to all other issues that have a security impact. These vulnerabilities are either very unlikely to be exploited or have minimal impact.

2.4 Statistical tools and methods

This section contains the statistical tools and methods that are used or mentioned in this report.

2.4.1 Linear regression

Linear regression is an approach in statistics to model the relationship between a dependent variable and one or more independent variables. In simple terms it is used to determine the equation of the line that best fits the given data points. All linear regression is done in R [28] which is a programming language for statistical computing.

2.4.2 None-linear regression

Non-linear regression is a form of regression analysis that is used when the observational data are best modeled by a nonlinear function. All non-linear regression is performed through an iterative process using a trial version of IBM SPSS Statistics [29].

2.4.3 Chi-Squared test

Chi-Squared (χ^2) is a commonly used goodness-of-fit test. A goodness-of-fit test is a test that measures how well a theoretical model fits the observed values, for example how well a regression model fits the data. The equation for the test is the following:

$$\chi^2 = \frac{1}{d} \sum_{k=1}^{n} \frac{(O_k - E_k)^2}{E_k} \tag{2.10}$$

It is a normalized sum of the squared differences between the observed and theoretical values (O and E respectively). This sum together with the number of degrees of freedom is used to look up the probability value (P-value) of the test which helps determine its statistical significance.

2.4.4 Coefficient of determination

Similar to the Chi-Squared test, the coefficient of determination (r^2) is a goodness-of-fit measurement of how many data points fall within the results of a regression model. It is the percentage of points the curve formed by regression analysis passes through. The coefficient of determination is the Pearson correlation coefficient (r) squared. This correlation coefficient is calculated in the following way:

$$r = \frac{n(\sum x_i y_i) - (\sum x_i)(\sum y_i)}{\sqrt{[n \sum x_i^2 - (\sum x_i)^2][n \sum y_i^2 - (\sum y_i)^2]}} \tag{2.11}$$

Where x and y are the two datasets that are being compared for n number of data points.

Chapter 3

Previous metric research

There has been a fair amount of research on the topic of software security metrics with a wide variety of different approaches. This chapter provides an overview of multiple interesting ideas and concepts that were found on the subject. These metrics will not be implemented, however ideas from Section 3.7 on predictive algorithms will used later in this thesis.

3.1 WIVSS

Weighted Impact Vulnerability Scoring System (WIVSS) was proposed by Spanos et al. in 2013 [30]. It was proposed to improve on CVSS v2 in terms of accuracy and to have a more balanced score distribution. Some important differences between WIVSS and CVSS v2 are:

- The impact score is between 0 and 7 instead of 0 and 10.
- Impact Score = Confidentiality + Integrity + Availability
- Confidentiality has the heaviest weight on the impact score. Confidentiality > Integrity > Availability.
- When the confidentiality metric is set to "None" then the other impact metrics also are set to "None".
- The 33 possible sums of the three impact metrics must be different.
- Partial Impact = 0.5 * Complete Impact (in CVSS v3 these are called Low and High)

3.2 PVL

Potential Value Loss (PVL) is a method of rating vulnerabilities developed by Yang & Wang in 2012 [31]. It uses the following seven indicators to measure severity.

- Loss of confidentiality of network node containing the vulnerability
- Loss of integrity of network node containing the vulnerability

- Loss of availability of network node containing the vulnerability
- Asset value of network node containing the vulnerability
- The level that asset values of network node depends on its confidentiality
- The level that asset values of network node depends on its integrity
- The level that asset values of network node depends on its availability

These indicators were selected based on the criteria that they should be effective, usable and accurate. The paper demonstrates PVL by using the method to rate published vulnerabilities for IP Multimedia Subsystem (IMS).

3.3 VRSS

Vulnerability Rating and Scoring System (VRSS) was proposed by Liu & Zhang in 2011 [32]. Their idea was to create a hybrid scoring system by merging qualitative and quantitative scoring methods. The system combines the qualitative metrics used in databases such as ISS X-Force and Vupen Security with the quantitative measuring in CVSS. Both of these databases had similar rating systems to the qualitative metrics mentioned in the previous section. The authors reasoning behind this was that analysis showed that the qualitative methods had greater consistency than CVSS. VRSS divides vulnerabilities into three priorities: High, Medium and Low. The purpose of the system was to replace the many different vendor-specific metrics that were in use at the time.

In 2012 Liu et al. further developed VRSS in order to improve the accuracy and have a bigger diversity in scoring [33]. The improvements enable the vulnerability type (based on CWE) to help prioritizing vulnerabilities. Using the vulnerability type together with a analytic hierarchy process on the basis of VRSS they analyzed 11,395 CVE vulnerabilities. The analysis of the results supports the claim that it has better score distribution than CVSS and the previous version of VRSS.

3.4 OWASP risk rating

Open Web Application Security Project (OWASP) is a non-profit organization with an online community that provides free information in the field of web application security. They have developed their own risk rating system [34]. They define risk as $risk = likelihood \cdot impact$. Likelihood is the average of eight metrics, four of them pertaining to the ”Threat agent” (the attacker) and four to the vulnerability. Each of these metrics can have a score between 0 and 9. In the context of this thesis we are only interested in the proposed four metrics that are about the vulnerability itself. This because the threat agent and the impact varies between targets. The four metrics that are related to the vulnerability and their suggested scale:

- Ease of discovery
  How easy is it for this group of threat agents to discover this vulnerability? Practically impossible (1), difficult (3), easy (7), automated tools available (9)

- Ease of exploit
  How easy is it for this group of threat agents to actually exploit this vulnerability? Theoretical (1), difficult (3), easy (5), automated tools available (9)
- Awareness
  How well known is this vulnerability to this group of threat agents? Unknown (1), hidden (4), obvious (6), public knowledge (9)
- Intrusion detection
  How likely is an exploit to be detected? Active detection in application (1), logged and reviewed (3), logged without review (8), not logged (9)

3.5 Policy security score

This metric, suggested by Muhammad Abedin et al. in 2006 attempts to measure the quality of protection of a security policy [35]. The score ranges from 10 to 0, 10 being the most secure. It uses several factors to determine this score:

- EF: Exposure Factor
- TRF: Traffic Rate Factor
- EVM: Existing Vulnerability Measure
- HVM: Historical Vulnerability Measure
- AHVM: Aggregated Historical Vulnerability Measure
- TVM: Total Vulnerability Measure.
- PSS: Policy Security Score

Figure 1 on page 1 in the referenced paper [35] is a good flow-diagram of how the Policy security score is calculated. The factors that are most interesting in the context of this thesis are EVM and HVM. EVM(A) measures all vulnerabilities that exist on system A.

$$EVM(A) = \alpha_1 \cdot \ln(\sum_{v_i \in EV_S(A)} e^{SS(v_i)}) + \alpha_2 \cdot \ln(\sum_{v_j \in EV_U(A)} e^{SS(v_j)}) \tag{3.1}$$

SS(v) is the severity score (CVSS) of vulnerability v. The set EV_U contains vulnerabilities that have no patch available and EV_S contains the vulnerabilities that have patches available but they have not been applied. The alphas are weights that can be adjusted accordingly.

The historical vulnerability measure is especially interesting in the context of this thesis. Operating under the assumption that older vulnerabilities are analyzed and patched with time it is used as an exponential decay of the severity score.

$$HVM(S) = \ln(\sum_{X \in \{H,M,L\}} w_X \cdot \sum_{v_i \in HV_X(S)} SS(v_i) \cdot e^{-\beta \cdot Age(v_i)}) \tag{3.2}$$

The sets H,M,L are the priority class of the vulnerability (High,Medium,Low) and w is the weight for that specific priority. β controls the speed of the decay.

3.6 K-zero day safety

K-zero day safety is a network security metric proposed by Wang et al. in 2014 [36]. A network security metric measures the security level of a particular network asset unlike the previously discussed metrics that focus on specific vulnerabilities. The authors state that one of the biggest issues with network security is the inability to measure the unknown. The unknown in this case is if an attacker uses non-public or previously undiscovered vulnerabilities. These are referred to as 0-day vulnerabilities because they give no time for the software vendor to fix the issue. K-zero day safety uses attack graphs to measure the number of 0-day vulnerabilities required to compromise a network asset.

See Figure 3 on page 4 in the referenced paper [36] for an example of an attack graph where each triple indicates an exploit (<vulnerability, source host, destination host>) and each pair indicates a condition <condition, host>. It is a directed graph with edges pointing from pre-conditions to their corresponding exploits and from those exploits edges point to post-conditions. In the referenced Figure there are four different "paths" an attacker can take to the final target (root,2), all require at least 3 vulnerabilities. For example if the pre-conditions <http,1> (HTTP service on host 1) and <0,1> (connectivity between host 0 and 1) are fulfilled an attacker on host 0 can used a 0-day exploit for the HTTP server which gives him user privileges on host 1 (<user,1>). Those privileges are used with a second 0-day to exploit the secure shell service to gain user privileges on host 2. In order to gain root access on the measured asset (host 2) the attacker will need a third 0-day. Unpatched software can of course change the score drastically depending on where it is positioned in the path to the asset. When analyzing networks in this manner it becomes easier to find the weak links.

3.7 Predictive algorithms

On a software level, vulnerability discovery risk can be measured by estimating the number of residual vulnerabilities as shown by the following equation:

$$RVD = VD - KVD$$

VD stands for Vulnerability Density, KVD stands for Known Vulnerability Density. This means that the software security is measured by the difference between the number of known vulnerabilities and the total number of vulnerabilities that will ever exist for that software. The main problem is estimating the total number of future vulnerabilities. One approach to this problem is to use vulnerabilities discovery models (VDMs). Some VDMs suggested include Alhazmi et al. [37] where they proposed a model now called Alhazmi Malaiya Logistic (ALM), Rescorla [38] who used an exponential model and Anderson [39] who based a discovery model on thermodynamics. Joh et al. [40] attempted to use Weibull distribution as a model but it did not always provide a good fit. These models use different variables and algorithms but generally a VDM is either time-based or effort-based. The effort-based models take into account the number of users by using market share data while the time-based use calendar time only.

According to [41] the AML model seems to be the best fitting but it assumed a symmetry in vulnerability discovery around the peak that is not always there. Further research was made by for example Joh et al. [42] who tried to capture asymmetric behavior by modeling the skewness of the distribution.

The following equation is the ALM model:

$$\Omega_{AML}(t) = \frac{B}{BC^{-ABt} + 1}$$

Note that $\Omega(t)$ approaches B as the time t approaches infinity. This means that B represents the total number of vulnerabilities that will be found in the software. A and C determine the slope during the different phases of the curve. During fitting these 3 parameters can be estimated through non-linear regression analysis. These parameters are then used when performing a goodness-of-fit test like the χ^2 test (Section 2.4.3). One method to find a good model is to try different models and use a goodness-of-fit test to evaluate their success. This method was used in 2016 by Awad A Younis Mussa [43] where he compared ALM and a folded VDM model and found that the folded VDM model performed better, but as the author himself remarks in his conclusions, the folded VDM model needs further testing to develop guidelines on how and when it should be used.

3.8 Other work on security metrics

In 1985 Mosleh et al. [44] proposed a bayesian probabilistic approach for risk analysis. The authors model the potential loss as a family of normal distributions. More recently, Xie et al. showed how bayesian networks can be applied on attack graphs [45].

Another approach is fuzzy logic. Fuzzy logic has been widely used in risk analysis and is also being used to prioritize vulnerabilities [46]. In 2013 Huang et al. [47] applied a fuzzy delphi method on the CVSS score of a vulnerability to filter out the base metric and the temporal metric. Then they used a fuzzy analytical hierarchy process to obtain fuzzy membership values. These values are analyzed with a synthetic decision making model to show to which degree the vulnerability affected the security of the system. This method shows how different weight of the evaluation criteria in CVSS affect the vulnerability priority.

Chapter 4

Method

This chapter presents the method used during the software and vulnerability analysis. The first section is an outline of the analysis process which gives an oversight of how the analysis results will be presented. The second section explains how software was chosen for analysis. The following sections provide information on what variables were considered during analysis and will explain the data gathering process. The last section explains the reasoning behind the survey and how it was conducted.

4.1 Analysis outline

The analysis is divided into four parts, the first two are focused on analysis of software and vulnerability data. The first part is a broad general comparison between several software categories based on data that can and has been automatically collected. The second part is a deeper look into one of the software categories and the software contained within it. During this deeper look the analysis will focus on the data points defined in this chapter. The third part is an analysis of the public exploit sources and the fourth part is completely focused on the survey, more about these parts in Sections 4.5 & 4.9 respectively. Interesting results of this analysis will be presented in Chapter 5 and used to investigate options for a software security metric in Chapter 6.

4.2 Software selection

In order to have a wide array of software available during analysis, software was chosen based on popularity and on what application the software has. The following nine categories of software were chosen:

- Video playback libraries and software
- HTTP-Servers
- Encryption libraries and software
- Script languages
- Firewall software

- Virtualization software
- FTP-Servers
- Communication (Chat software)
- SQL-Servers

Each of these categories contain 4-6 various software of different popularity levels for a total of 44 software from different developers. The exact software chosen are listed in Appendix A. During this report these categories will be referred back to in shortened form such as ”HTTP” or ”Encryption”.

4.3 Vulnerability data

This section specifies what vulnerability data is collected and what definitions are used.

4.3.1 Historical data

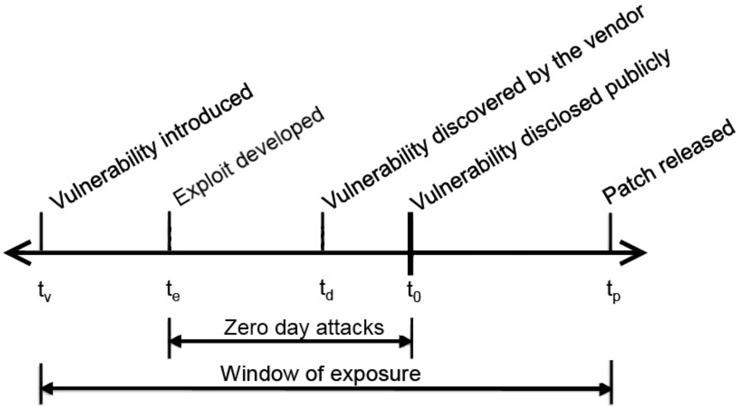


Figure 4.1: Vulnerability life timeline

Five interesting points in time during the vulnerability life cycle have been identified: Time of Introduction, Time of Exploitation, Time of Discovery, Time of Disclosure and Time of Patching as shown in Figure 4.1. These points in time are not necessarily in that order, for example the vulnerability can be discovered by the vendor and therefore no exploit is released while the software is vulnerable as shown in Figure 4.2.

We define the Time of Introduction (t_v) as the point in time that the vulnerability was introduced in the software. t_v is almost always unknown and will not be used in the analysis. The four remaining points in time have all been previously used in analysis of vulnerabilities, for example in a paper by Shahzad et al. [48] and in a large-scale analysis by Frei et al. [49].

We define the Time of Exploitation (t_e) as the oldest confirmed date at which an exploit or proof of concept for exploiting the vulnerability was released.

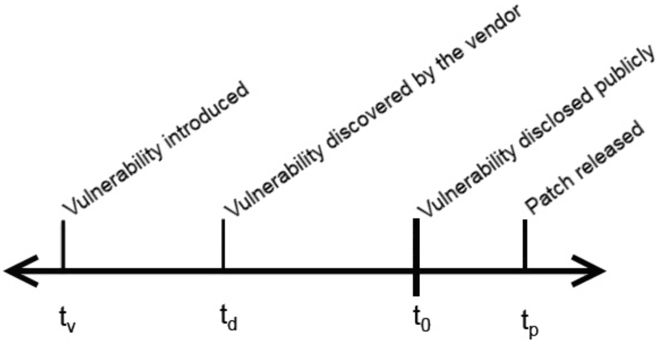

Vulnerability introduced
Vulnerability discovered by the vendor
Vulnerability disclosed publicly
Patch released
t_v
t_d
t_0
t_p

Figure 4.2: Alternative vulnerability life timeline

We define the Time of Discovery (t_d) as the date the vendor claims to have discovered the vulnerability. Often this is not reported but in some cases it is included at the time of disclosure.

We define the Time of Disclosure (t_0) as the oldest date at which the vendor publicly acknowledges the vulnerability. For a lot of the vulnerabilities in our database this will be close to the date in the CVEs published field, the reason for this being that these vulnerabilities are publicly acknowledged by the vendor after the vulnerability has been patched but before the CVE is made public.

We define the Time of Patching (t_p) as the date at which the software vendor released either a software fix or a suggested solution to the specific vulnerability. Third-party solutions are therefore excluded. If an official fix is available the date the fix was released will be used as the Time of Patching even if alternative fixes existed previously.

We compare the exploit-, discovery- and patch-time with the disclosure date and analyze the distribution of these points in time in an attempt to quantify the stages of the vulnerability life-cycle and discover trends.

4.3.2 CVSS

CVSS v2 is included in all CVE data gathered from NVD, CVSS v3 is included in all CVEs published since the end of 2015. As previously described in Section 2.3.2 the main difference between version 2 and 3 is ”Scope”. For software where the scope remains unchanged when exploited, CVSS v2 is sufficient. In the interest of consistency and backward compatibility CVSS version 2 will be used during the analysis.

4.3.3 Temporal data

Temporal metrics is an optional part of CVSS (see Section 2.3.3) and it is often not included in vulnerability databases. If data about temporal metrics can be found for the examined CVEs it will be collected and used together with the other

CVSS data to calculate temporal scores for each CVE.

4.4 Vulnerability types

As described in Section 4.3.1, the defined points in time can be in different orders (excluding vulnerability introduction). Depending on the order of these points we divide the vulnerabilities into different types. For example if $t_e < t_p$ the vulnerability was exploitable with no patch available from the vendor to protect the user and if $t_e < t_0$ it is a zero-day exploit.

Disclosure:

- Type 1: When $t_e < t_0$. The exploit was published before the vulnerability was disclosed. We call this a malicious discovery.
- Type 2: When $t_e >= t_0$ The exploit was published together with or after a public disclosure about the vulnerability has been made.

Patching:

- Type 1: When $t_p <= t_0$. The vulnerability was patched before a public disclosure was made. This means that the vulnerability was handled internally before it became public and could cause harm on a larger scale. The vast majority of the vulnerabilities collected belong to this category.
- Type 2: When $t_0 < t_p$. The vulnerability was patched at least one day after a public disclosure was made.
- Type 3: When $t_p = null$. No patch date or patch found for this vulnerability.

4.5 Exploits

Since the exploit publishing date is one of the important points in time identified in Section 4.3.1 data about public exploits will be be gathered and an analysis of that data will be performed as to determine the vulnerability coverage of our exploit sources and to find any correlation between the publishing dates of public exploits and vulnerabilities.

4.6 Data sources

The task is to collect as many of the dates above for the vulnerabilities in our database, the difficulty in this is that no single source provides the dates. Finding the points in time requires background research on each vulnerability. Therefore we want to use sources with extensive references. By restricting ourselves to vulnerabilities that have a CVE-identifier we can use NVD as our only source of vulnerabilities. NVD includes a list of external references with each CVE that will be the source of dates to correlate with the points in time we are interested in. As a source for public exploits and exploit dates we use exploit-db [50] and the references on NVD. For the temporal data a mix of data from IBMs X-force

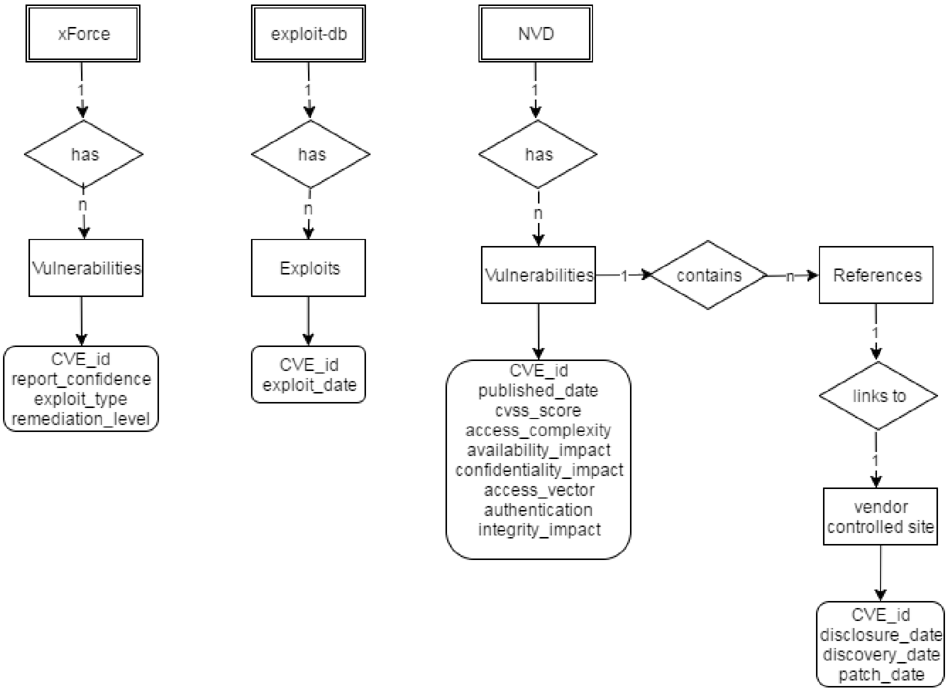


Figure 4.3: Diagram of the collected data and its sources.

Exchange [12] and manual reference research was used. Figure 4.3 shows a relation diagram over the sources that were used and what data was collected from them. The CVE identifiers were cross-referenced and added to the same database. The ”vendor controlled site” in the diagram refers to any official channel the software vendor uses to disclose vulnerabilities and announce patches, for example the OpenSSL security advisory [51].

4.7 Other software data

This section lists some other data gathered about the vulnerabilities that may prove to be interesting and relevant in the analysis.

4.7.1 Users and popularity

Being precise when measuring the popularity of a specific software is difficult since vendors rarely publicize their user data. With the ultimate goal of automation in mind the metric that will be used in its place is search engine hits. Since search engine hits usually have a high variance it requires collection of data during an extended period of time. Therefore data from both Google and Bing have been collected once every week during two months for the chosen set of software. This data has been used to evaluate the potential of using search engine hits as a metric and to theorize about potential applications of it.

4.7.2 Developers

The number of developers that worked on the software. This data is not always available but if the project is open source it is often possible find.

4.7.3 Language

What programming language the software was written in. This data is almost always available.

4.7.4 Documentation

It is no simple task to quantify what makes a good documentation since they often target different user groups with different skills. However, usability is important for all of them. Three heuristics have been identified that are important when judging the usability of software documentation:

1. Search and navigation. A user should be able to find the required topic using some form of search function, or by browsing to it.
2. Orientation. A users should be able to know where they are in the documentation relative to the whole.
3. Minimalist writing. The documentation should avoid irrelevant information.

Using these heuristics as a foundation together with personal judgment a score from 0 (no documentation) to 3 (excellent documentation) is chosen.

4.7.5 Release activity

How many releases were made per day during a certain time period, measured in days/release. This data has to be manually gathered since releases are published on different sites for different software.

4.8 Data gathering process

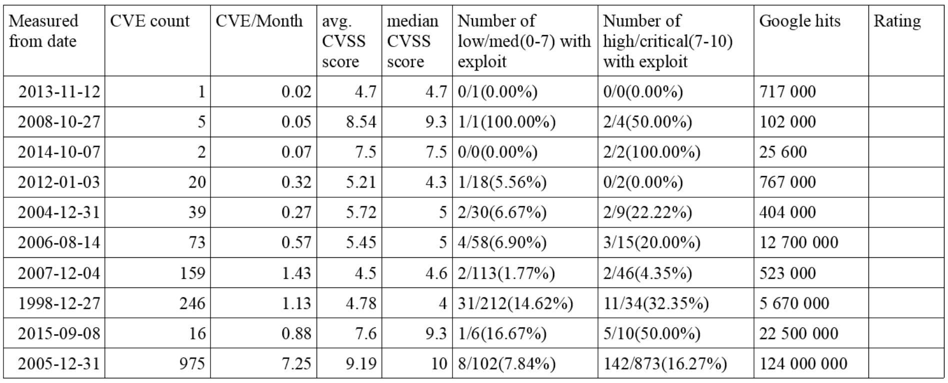
The CVE data is available on NVD in a compressed XML format. This data was gathered using a Python script that automatically downloaded, extracted and inserted the data into a mongoDB database. Having the data stored in a database simplifies the process of retrieving CVE data for a specific software. For example a query retrieving all CVEs where "OpenSSL" is listed among the products could look like this:

```
db.cves.find({"products": {"$in":[/OpenSSL/]}}).pretty()
```

The data available is publish date, modified date, CVSS score, references and a summary of the vulnerability. NVD does not include any specific vulnerability data like the points in time defined in Section 4.3.1, exploit data or patch data. This data was gathered manually and inserted into a spreadsheet. The temporal metrics on X-force Exchange were gathered using a script calling their API that allows up to 5000 free searches per month. The public exploit data was gathered from an offline version of exploit-db that is available on Github [52].

4.9 Survey

The importance (priority) of the different metrics that have been identified has to be decided, not only to determine which metrics to include in a potential scoring system but also to decide their weights if they are included. In order to accomplish this a small survey has been performed amongst professionals in the software security industry. To be specific, the survey was directed at "anyone with an interest in software security and anyone who actively tries to pick secure software."



Measured from date	CVE count	CVE/Month	avg. CVSS score	median CVSS score	Number of low/med(0-7) with exploit	Number of high/critical(7-10) with exploit	Google hits	Rating
2013-11-12	1	0.02	4.7	4.7	0/1(0.00%)	0/0(0.00%)	717 000	
2008-10-27	5	0.05	8.54	9.3	1/1(100.00%)	2/4(50.00%)	102 000	
2014-10-07	2	0.07	7.5	7.5	0/0(0.00%)	2/2(100.00%)	25 600	
2012-01-03	20	0.32	5.21	4.3	1/18(5.56%)	0/2(0.00%)	767 000	
2004-12-31	39	0.27	5.72	5	2/30(6.67%)	2/9(22.22%)	404 000	
2006-08-14	73	0.57	5.45	5	4/58(6.90%)	3/15(20.00%)	12 700 000	
2007-12-04	159	1.43	4.5	4.6	2/113(1.77%)	2/46(4.35%)	523 000	
1998-12-27	246	1.13	4.78	4	31/212(14.62%)	11/34(32.35%)	5 670 000	
2015-09-08	16	0.88	7.6	9.3	1/6(16.67%)	5/10(50.00%)	22 500 000	
2005-12-31	975	7.25	9.19	10	8/102(7.84%)	142/873(16.27%)	124 000 000	

Figure 4.4: Table used during the survey.

Figure 4.4 shows the table used during the survey. Participants are asked to rate the unnamed software from 1 to 10 where 1 is “I would definitely include this software in a product” and 10 is “I would never include this software in a product”. The participants are also asked to comment on which of the metrics they found most useful, least useful and if they felt any metric was missing.

The metrics included in the survey were chosen based of the following criteria:

- Relevancy. The purpose of the survey is to determine the relevancy of the metrics, therefore metrics that the author considered relevant were chosen.
- Automation possibility. All metrics in the table can and have been collected automatically.
- Easy to understand. Since the survey has been sent to people that likely do not possess knowledge about the inner workings of CVSS, the chosen metrics have to be easy to explain and understand.

The final results of the survey will be analyzed partly through regression analysis and partly through review of answers to the free text questions. The regression analysis is performed to find the correlation between the metrics and the ratings and the review is done to find common threads among the answers. The regression analysis combined with the comments received on the survey serves as the basis for deciding what weights the metrics will have in a scoring system.

Chapter 5

Results

The results of the analysis are presented in this chapter. The first two sections are focused on vulnerability analysis, both broad and detailed. The third section reviews exploit-db as a source for public exploits and the fourth is a compilation of the survey results. Graphs and comments on these results are included.

5.1 Vulnerability analysis of multiple categories

Using only data collected automatically with scripts we extend the amount of software analyzed. Nine categories of software have been chosen and the software within those categories have been picked to represent a wide array of popularity. Listed in Appendix A are the categories and the keywords (most often vendor:product_name) used in our search for vulnerabilities for specific software.

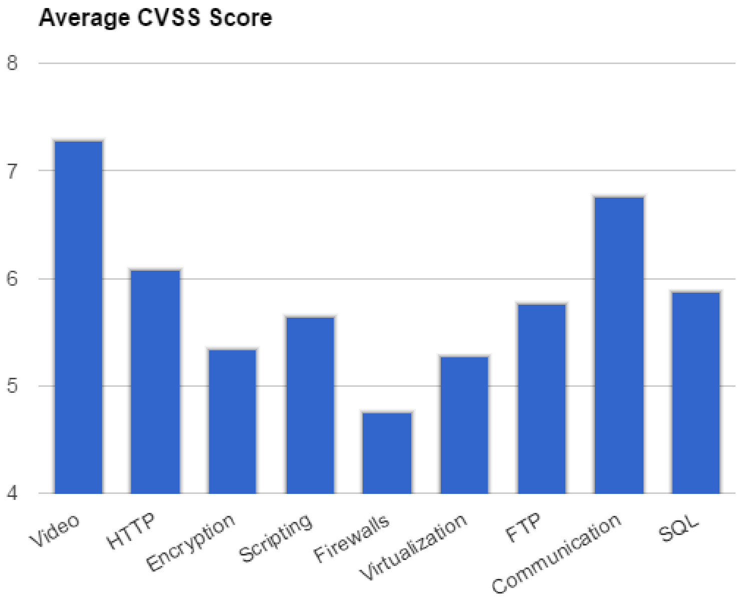


Figure 5.1: Average CVSS Score per category

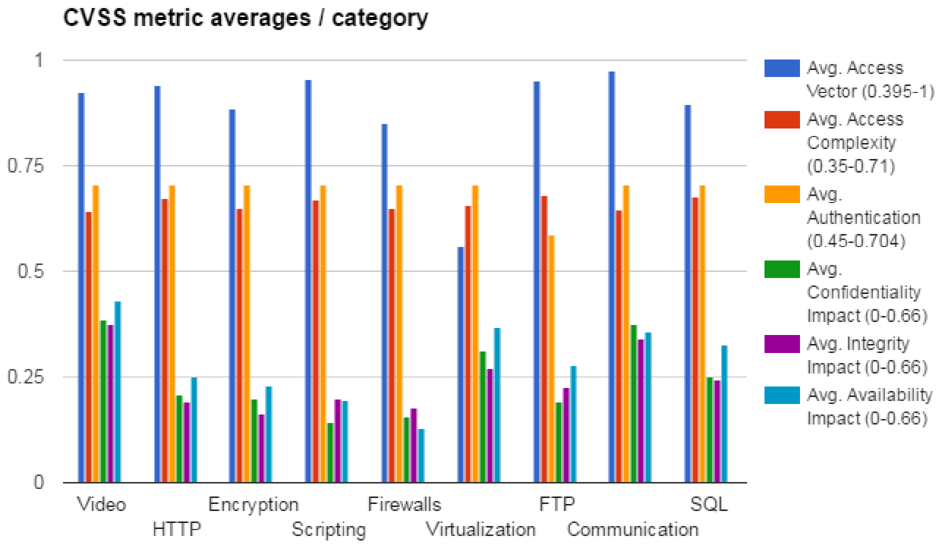


Figure 5.2: The average CVSS sub-metric values per category

Figure 5.1 compares these categories to each other based on average CVSS score. The Figure shows that there is a large difference between these categories, for example software examined in the ”Firewall” category have vulnerabilities with an average CVSS score of 4.754 while the ”Video” software averaged 7.276. Two theories that could explain this difference are:

1. Software developers for categories like ”Firewalls” or ”Encryption” have a higher focus on security compared to developers of ”Video” software (for obvious reasons).
2. Video players have a larger user base and are therefore a more viable option for exploitation in most cases. For example, if a user is running a video player that is vulnerable to code execution through a maliciously formed video file an attacker will most likely try to trick the user into executing the attackers code from the inside the network and bypass the firewall instead of attacking the firewall software directly.

Figure 5.2 breaks down the CVSS score into the specific CVSS metrics and shows that the security-focused categories are overall much better at handling the Confidentiality, Integrity and Availability-impact of a vulnerability. This leads us to the third theory which is that Windows-versions may be poisoning the data-well. CVEs normally do not specify the underlying operating system which means that for software like VLC the versions for the different operating systems are mixed under the same software name while in the firewalls category the software picked was all UNIX-based and do not have Windows-versions. A comparison of the impact metrics between the Windows and Linux versions of the same software would be interesting, unfortunately even though NVD tries to add details like operating system the CVEs they provide are still lacking in this respect and data collection would most likely have to be done manually or with another source to get a somewhat trustworthy result.

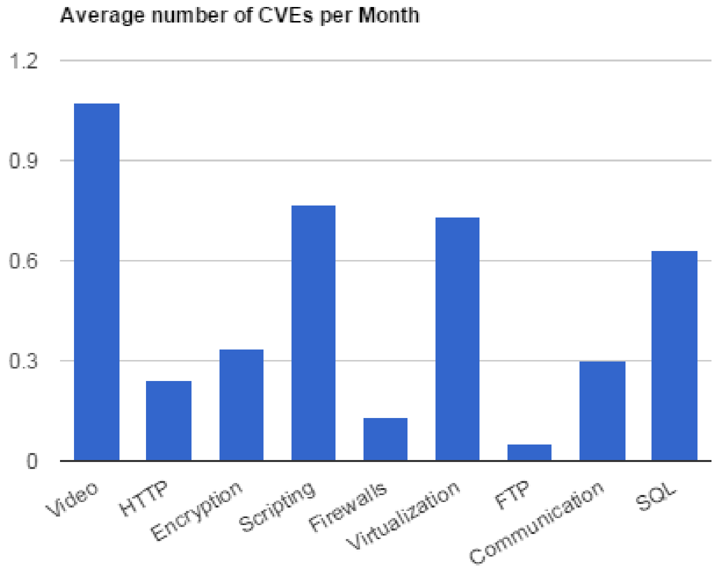


Figure 5.3: Average number of CVEs per Month for each category

Figure 5.3 is a graph that shows the average number of CVEs that are published each month (30,44 days) in the different categories, starting with the date of the first published CVE for that software. In conjunction with the previous graph depicting the average CVSS scores (Figure 5.2) it shows that software in some categories such as FTP-Servers, HTTP-Servers and communication (chat) have a lower discovery frequency of vulnerabilities compared to other categories but that these vulnerabilities still have a rather high average CVSS score. This is because the impact (confidentiality, integrity and availability) of vulnerabilities in these categories is high.

5.2 Vulnerability analysis of a single category

This section focuses on deeper analysis into the software in the ”Encryption”-category and their CVEs. Currently the data analyzed in this section is restricted to five programs and their vulnerabilities stretching from 2014-01-01 to 2016-09-27. This restriction was necessary due to the time consuming nature of researching the individual CVEs. In cases where data is available outside this range of dates it will still be included.

Table 5.1 above shows the connection between popularity and security issues. The data was collected once every week for 10 weeks for the five chosen software by simply searching for the software names on the respective search engines and retrieving the number of hits. It is not a big sample size but they do seem to correlate. See Figure 5.4 and 5.5. A metric sometimes used in Vulnerability Discovery

Table 5.1: Search hits based on an average over 10 weeks

	CVEs/Month	Google hits	Bing hits
OpenSSL	0,813	16922222	1153333
GnuTLS	0,272	484555	128777
LibreSSL	0,407	113333	136000
OpenSSH-p	0,043	2370000	343000
WolfSSL	0,187	97500	52300

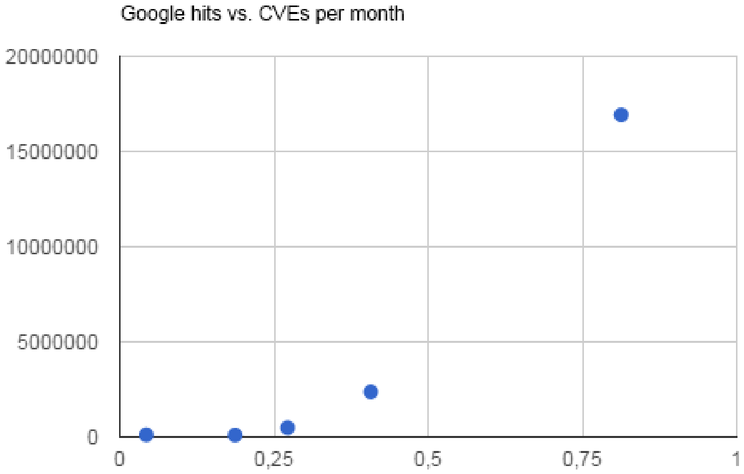


Figure 5.4: Correlation between CVE/Month and the number of Google hits

Models (see Section 3.7) is ”market share”, the average number of Google-hits can be used as a rough way of estimating that without access to any official numbers.

Tables 5.2 and 5.3 show how many dates of each category that were collected for the chosen software. Non-internal refers to being in patching category 2 as defined in Chapter 4. When looking at 5.3 a reader may assume that the three non-internal vulnerabilities for OpenSSL are the same vulnerabilities for which exploit-dates are available, however this is not the case, none of them are.

Table 5.2: Information about the gathered dates.

	# of CVEs	Have Discovered date	Have patch date
OpenSSL	89	72	88
GnuTLS	14	2	13
LibreSSL	3	1	1
OpenSSH-p	15	3	11
WolfSSL	4	1	4

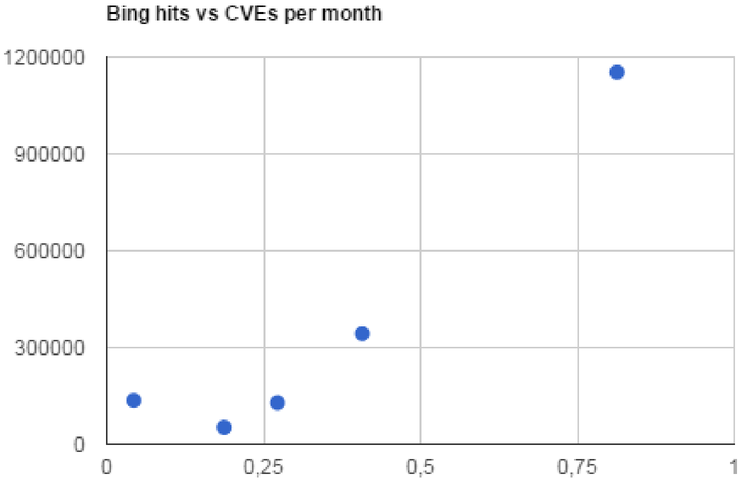


Figure 5.5: Correlation between CVE/Month and the number of Bing hits

Table 5.3: More information about how many of each date were gathered.

	# of CVEs	Disclosure date	Exploit date	Non-internal
OpenSSL	89	89	3	3
GnuTLS	14	6	1	4
LibreSSL	3	1	0	0
OpenSSH-p	15	8	1	6
WolfSSL	4	4	0	0

Table 5.4: Statistics gathered from open source repositories

	# of contributors	Days/Release
OpenSSL	220	24.45
GnuTLS	14	17.1
LibreSSL	26	18.2
OpenSSH-p	14	83.75
WolfSSL	32	24.42

Some statistics gathered from the open source repositories can be seen in Table 5.4. OpenSSH only had a repository for the portable version (OpenSSH-p) and that is the repository we used for the data.

These statistics show that OpenSSL has by far the most contributors to the source code which is unsurprising given that it is also the by far the most popular software of the five. In theory more contributors could increase the risk of new vulnerabilities being introduced due to human error but it also means that the code is more heavily scrutinized. The numbers also show that the portable version of OpenSSH have more than three times as many days between releases as the other software. There can be multiple reasons behind the slow release pace of OpenSSH, for example a low amount critical bugs combined with a low amount of developers. It is also an 18 year old piece of software that has been refined through the years and might not require as much maintenance. There is no obvious correlation between this data and the number of vulnerabilities in the software.

5.2.0.1 Documentation

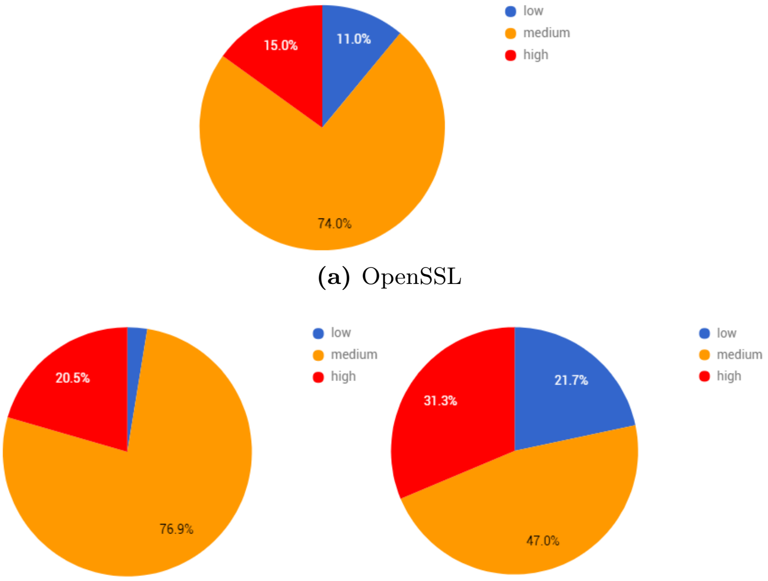
These are the subjective scores given to the documentation based on the criteria specified in Section 4.7.4.
OpenSSL docs [53]
Score:2/3
Motivation: Minimalist writing. Decent orientation and navigation. No search.
GnuTLS docs [54]
Score:3/3
Motivation: Minimalist writing. Excellent orientation and navigation.
LibreSSL docs [55]
Score:1/3
Motivation: Could only find autogenerated documentation. The real documentation seems to be a work in progress.
OpenSSH docs [56]
Score:2/3
Motivation: Minimalist writing. Good orientation and navigation. Search function available as the documentation is part of openBSD manual.
WolfSSL docs [57]
Score:3/3
Motivation: Extensive documentation with minimalist writing. Good orientation and navigation.

5.2.0.2 CVSS base scores

Table 5.5: Average CVSS base score for the chosen software.

Software	Average CVSS base score
OpenSSL	5.413
GnuTLS	5.723
LibreSSL	6.094
OpenSSH	7.5
WolfSSL	3.8

Table 5.5 shows the average CVSS scores for the software and Figure 5.6 compares the CVSS base score distribution in OpenSSL, GnuTLS and OpenSSH. Both Li-



(a) OpenSSL

(b) GnuTLS

(c) OpenSSH

Figure 5.6: CVSS base score distributions

breSSL and WolfSSL had too few CVEs to make a proper distribution.

1. Vulnerabilities are labeled "low" if they have a CVSS base score of 0.0-3.9.
2. Vulnerabilities are labeled "medium" severity if they have a base CVSS base score of 4.0-6.9.
3. Vulnerabilities are labeled "high" severity if they have a CVSS base score of 7.0-10.0.

OpenSSL and GnuTLS have a similar distributions of CVSS scores while OpenSSH has noticeably smaller percentage of medium severity vulnerabilities. The reason for this is examined more closely in Section 5.2.0.3.

5.2.0.3 CVSS Sub-metrics

Figure 5.7 shows the average of the CVSS sub-metrics of the vulnerabilities in the chosen software. Again, note that there are a very low number of CVEs with CVSS metrics for LibreSSL(1) and WolfSSL(4) which makes the average values for these software less reliable. The previous section showed that OpenSSH has a smaller percentage vulnerabilities with a medium CVSS score compared to OpenSSL and GnuTLS. Examining the sub-metrics shows that OpenSSH has a noticeably lower average access vector which means that on average more of its vulnerabilities are not exploitable remotely over the Internet. This of course decreases the risk of the vulnerability being exploited and these non-remote vulnerabilities will in most

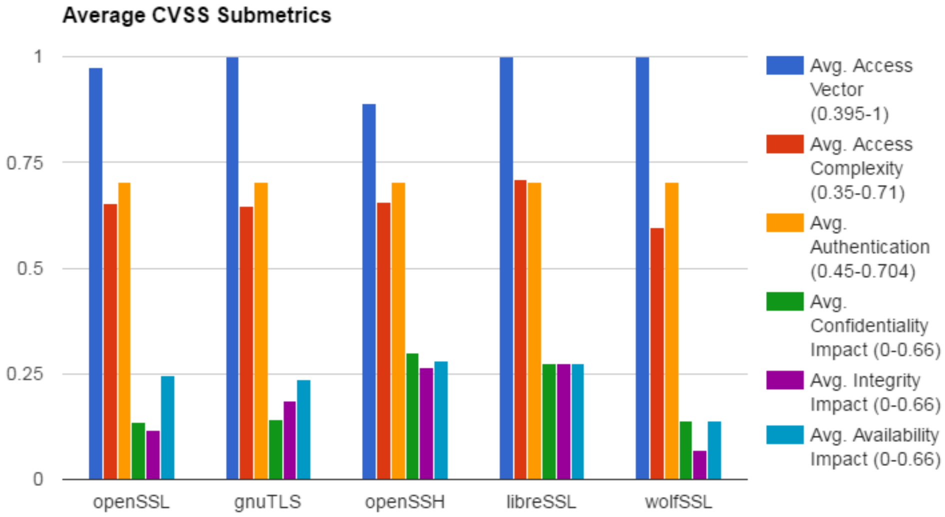


Figure 5.7: The average CVSS sub-metric values

cases be of low severity. This increases the percentage of low severity vulnerabilities and decreases the medium percentage. Other than that, the severity of vulnerabilities in this category seem to be at a similar level and therefore no other conclusions about differences can be drawn.

5.2.0.4 Analysis of exploits

Using the proposed definitions in Chapter 4 Figure 5.8 represents $t_e - t_0$ and it is clear by looking at it that public exploits are far and few between for the chosen software (note that WolfSSL had no exploits). For this reason the entire lifespan (under their current names) of the software has been included in the search. Even though this sample size is small we see that a vast majority of the vulnerabilities belong in what we called ”Disclosure Category 2”, where $t_e >= t_0$. All except three of these vulnerabilities were disclosed before the exploit was released publicly on exploit-db. Comparing this with the results from the exploit source analysis in Section 5.3 leads to the conclusion that software exploits are most likely to be released at the time of disclosure and that the CVE is published soon thereafter. Table 5.6 shows exploit data for the entire lifespan of the chosen software.

5.2.0.5 Analysis of historical data

Figures 5.9 and 5.10 show scatter plots. These graphs are constructed by plotting the disclosure date on the x-axis and on the y-axis the difference between the Discovery and Patch-dates to the disclosure date in days.

Figure 5.9 represents $t_d - t_0$ and only has data entries from OpenSSL because they were the only vendor of those chosen that provide discovery dates. The graph tells us that the vast majority of the vulnerabilities disclosed are fixed

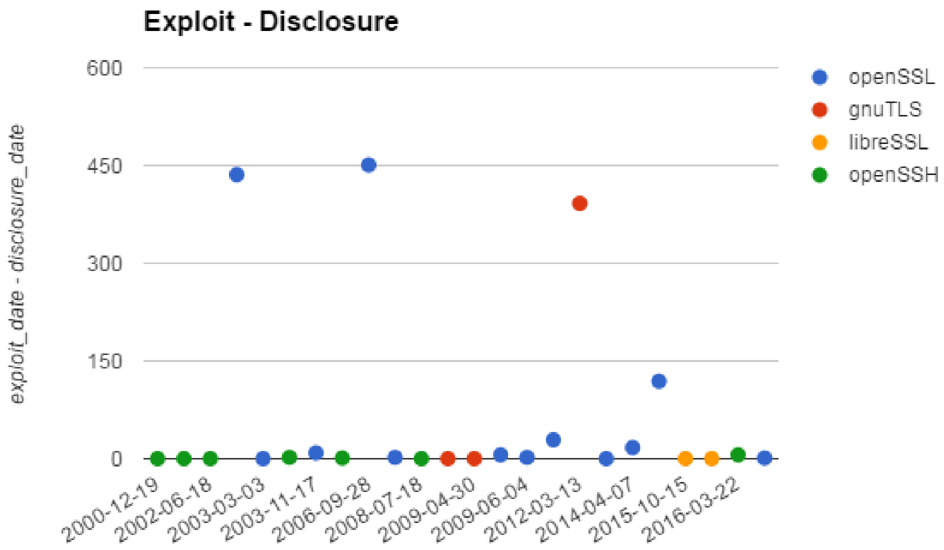


Figure 5.8: The number of days from disclosure to exploit

Table 5.6: Exploit data for the entire lifespan of the software

		Number of CVEs		
	Total amount of exploits	with at least one exploit	total	% of CVEs with exploits
OpenSSL	30	14	214	5,93%
GnuTLS	4	4	40	10%
LibreSSL	2	2	3	66,66%
OpenSSH	22	8	83	10,38%
WolfSSL	0	0	6	0%

within a time-span of 3 months. Figure 5.10 shows $t_p - t_0$ and indicates that most vulnerabilities are disclosed at or very soon after the time of patching which makes them what we in Section 4.4 called patching type 1.

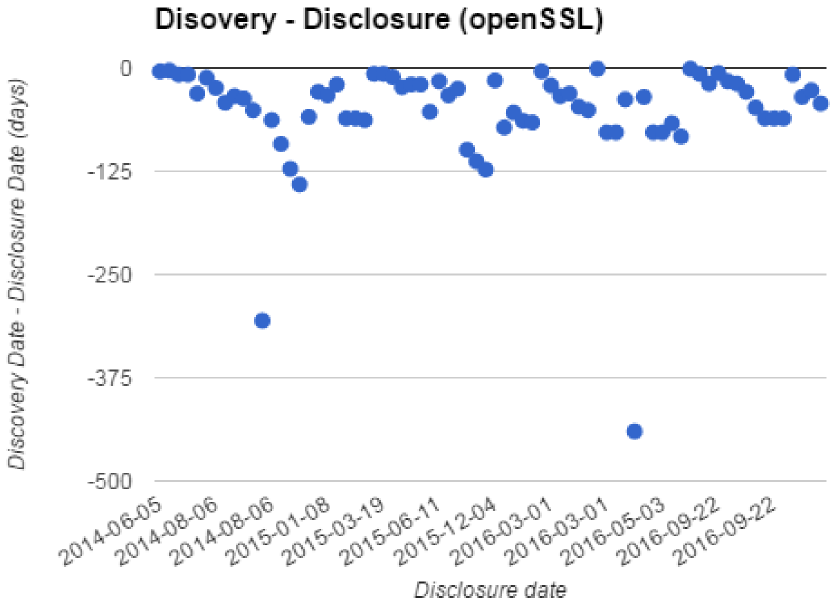


Figure 5.9: The number of days from discovery to disclosure

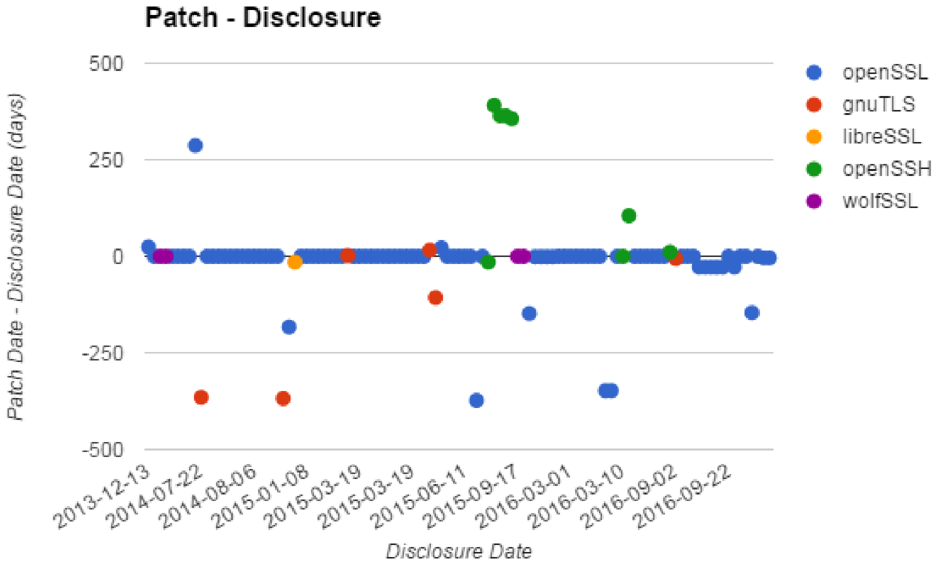


Figure 5.10: The number of days from disclosure to patch

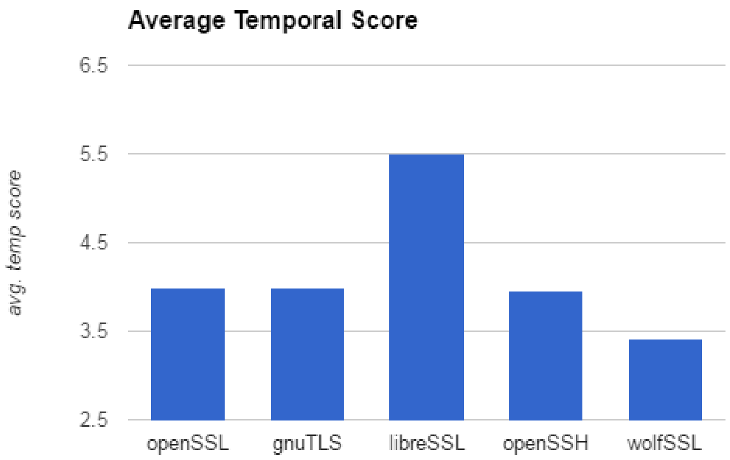


Figure 5.11: Average temporal scores for the software in the encryption category.

5.2.0.6 Temporal scores

Temporal score metrics such as exploit type, remediation level and report confidence were gathered from IBM xForce Exchange. Their calculated temporal scores can not be used directly on our data since NVD and xForce exchange have not always evaluated a vulnerability the same way and therefore given it different CVSS scores. With this discrepancy in mind we can still calculate our own temporal scores based on the temporal metrics gathered from xForce and the CVSS data gathered from NVD. Figure 5.11 shows the average temporal score for the chosen software, please note that the score for LibreSSL is only based of one single vulnerability which is the reason for it being an outlier.

5.3 Exploit source analysis

The source of exploits is exploit-db. Examining how viable it is as a source was done by counting the number of CVEs with a reference to exploit-db:

```
> db.cves.find({references: {$elemMatch: {source: "EXPLOIT
    DB"}}}).count();
```

The result is that 2548 CVEs out of 81769 (3.12%) have a reference to exploit-db. The exploit release dates for these exploits were automatically extracted by using an offline version of exploit-db that can be found on their Github repository [52]. The only other relevant date imported in large amounts is the CVE publish date, this is used as a substitute for the disclosure date since they tend to be close chronologically. Figure 5.12 shows a histogram of the number of days from the CVE publishing date to the public release of the exploit on exploit-db. The figure is based of 1831 data points after filtering out outliers (+-60 days). As

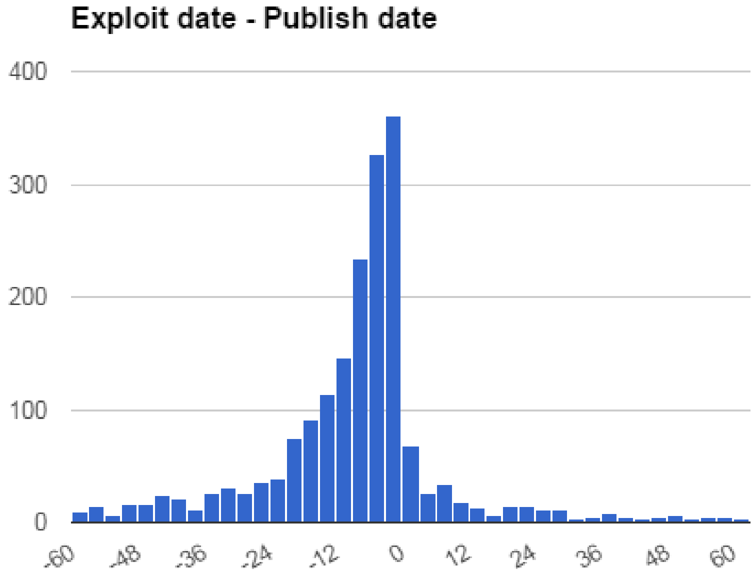


Figure 5.12: Histogram of the number of days from the CVE published date to the exploit date

is evident from the histogram a vast majority of the exploits were released shortly before or the same day as the CVE was published.

5.4 Survey results

As of the time of analysis the survey has had 12 respondents. This is most likely too few to make a proper statistical analysis however the data can still give interesting results. When compiling the results the two exploit columns (low/high severity) were split into four columns to provide both the absolute number of exploits and also the percentage of exploits relative to the vulnerabilities. Using the multivariable linear regression function in R [28] automatically runs a student's t-test and produces the P-values found in Table 5.7.

Table 5.7: P-values of t-test after multiple linear regression in R.

Predictor variables	P-value
measure_date	0.582
cve_count	0.726
cve_month	0.223
avg_cvss	0.341
median_cvss	0.752
lowmed_vulns	0.701
highcrit_vulns	NA
lowmed_exploits	0.621
highcrit_exploits	0.631
lowmed_exploit_percent	0.414
highcrit_exploit_percent	NA
google_hits	NA

The regression shows that none of the predictor variables have a low enough P-value to be used (0.05 was required for 95% confidence). As can also be seen from these results there is multicollinearity. Three predictor variables (Number of high/critical vulnerabilities, High/crit exploit percentage and Google hits) are not linearly independent which means that they can be removed without changing the outcome of the regression. Regression with an ordered logistic regression model was also attempted with the linearly dependent variables removed. This time a Z-test is used by R and the P-values of this test can be seen in Table 5.8. These results also show that the predictor variables all have high P-values. For the exact output of the regression analysis refer to Appendix B.

Table 5.8: P-values of Z-test after regression with an ordered logistic model in R.

Predictor variables	P-value
measure_date	0.4585
cve_count	0.6888
cve_month	0.0924
avg_cvss	0.1086
median_cvss	0.8883
lowmed_vulns	0.6391
lowmed_exploits	0.5566
highcrit_exploits	0.5579
lowmed_exploit_percent	0.4011

To inspect the linear dependency more closely see Figure 5.13 which shows the correlation matrix, the highest correlated predictor variables have been circled. The correlation matrix shows that Google hits correlates heavily with the CVE count and frequency (which was also shown in Section 5.2). It also correlates to both the number of high/critical vulnerabilities and high/critical exploits. The other correlations such as between CVE count and vulnerabilities/exploits are more obvious.

The answers to the surveys free text boxes follow below, very similar answers were combined into one row.

More important variables:

- CVE/Month in combination with average CVSS score. (x2)
- CVE count
- CVE/Month
- Google hits in combination with exploits
- Google hits (x2)
- Google hits / CVE count ratio
- Number of high/critical with exploit
- Number of high/critical vulnerabilities
- Median CVSS Score

Less important variables:

- Google hits (x2)
- CVE metrics (only important in combination with popularity)
- CVE/Month
- Number of high/critical exploits.
- Number of low/med with exploits
- Exploit columns
- Median CVSS score

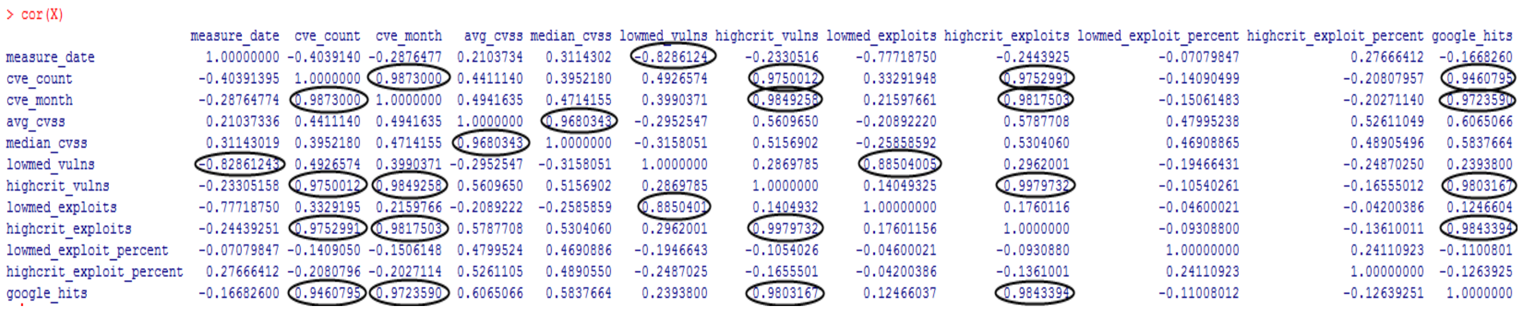
Missing variables:

- Time from exploit to patch (x3)
- Time from vulnerability to patch
- Time from critical vulnerability to patch
- Worst vulnerability to patch time
- Lines of code/Code complexity (x2)
- Ratio between lines of code and CVE count
- Number of active contributors to source

- Release frequency
- Scope (exposed to the Internet?)
- How difficult the vulnerabilities are to fix and business criticality
- What the software is used for (what is at risk?)
- Type of product (security requirements), how important is the component and alternatives to it
- A graph of CVE/month and avg. CVSS on a time line
- Details about how the CVSS score is calculated
- Who maintains the code (their priorities and security views)

Looking at the answers above the following immediate observation can be made: The vast majority of respondents wanted more information than the variables provided in the survey, the most common one being some form of measurement about how fast/easily the software is patched but plenty of other factors were suggested as well. Although the responses leaned towards popularity being an important measurement, there was no clear winner or loser amongst the predictor variables that were included in the survey.

The average ratings the software in the survey received can be seen in Figure 5.14. Except for the two standout (statistically) software in the top and the bottom, the rest of the ratings lean toward the middle of the spectrum. This result is consistent with the indecision between variables and the lack of additional information.



```
> cor(X)
                         measure_date  cve_count  cve_month   avg_cvss median_cvss lowmed_vulns highcrit_vulns lowmed_exploits highcrit_exploits lowmed_exploit_percent highcrit_exploit_percent google_hits
measure_date               1.00000000 -0.4039140 -0.2876477  0.2103734   0.3114302   -0.8286124     -0.2330516     -0.77718750        -0.2443925            -0.07079847               0.27666412  -0.1668260
cve_count                 -0.40391395  1.0000000  0.9873000  0.4411140   0.3952180    0.4926574      0.9750012      0.33291948         0.9752991            -0.14090499              -0.20807957   0.9460795
cve_month                 -0.28764774  0.9873000  1.0000000  0.4941635   0.4714155    0.3990371      0.9849258      0.21597661         0.9817503            -0.15061483              -0.20271140   0.9723590
avg_cvss                   0.21037336  0.4411140  0.4941635  1.0000000   0.9680343   -0.2952547      0.5609650     -0.20892220         0.5787708             0.47995238               0.52611049   0.6065066
median_cvss                0.31143019  0.3952180  0.4714155  0.9680343   1.0000000   -0.3158051      0.5156902     -0.25858592         0.5304060             0.46908865               0.48905496   0.5837664
lowmed_vulns              -0.82861243  0.4926574  0.3990371 -0.2952547  -0.3158051    1.0000000      0.2869785      0.88504005         0.2962001            -0.19466431              -0.24870250   0.2393800
highcrit_vulns            -0.23305158  0.9750012  0.9849258  0.5609650   0.5156902    0.2869785      1.0000000      0.14049325         0.9979732            -0.10540261              -0.16555012   0.9803167
lowmed_exploits           -0.77718750  0.3329195  0.2159766 -0.2089222  -0.2585859    0.8850401      0.1404932      1.00000000         0.1760116            -0.04600021              -0.04200386   0.1246604
highcrit_exploits         -0.24439251  0.9752991  0.9817503  0.5787708   0.5304060    0.2962001      0.9979732      0.17601156         1.0000000            -0.09308800              -0.13610011   0.9843394
lowmed_exploit_percent    -0.07079847 -0.1409050 -0.1506148  0.4799524   0.4690886   -0.1946643     -0.1054026     -0.04600021        -0.0930880             1.00000000               0.24110923  -0.1100801
highcrit_exploit_percent   0.27666412 -0.2080796 -0.2027114  0.5261105   0.4890550   -0.2487025     -0.1655501     -0.04200386        -0.1361001             0.24110923               1.00000000  -0.1263925
google_hits               -0.16682600  0.9460795  0.9723590  0.6065066   0.5837664    0.2393800      0.9803167      0.12466037         0.9843394            -0.11008012              -0.12639251   1.0000000
```

Figure 5.13: Correlation matrix of survey results

Measured from date	CVE count	CVE/Month	avg. CVSS score	median CVSS score	Number of low/med(0-7) with exploit	Number of high/critical(7-10) with exploit	Google hits	Average Rating
2013-11-12	1	0.02	4.7	4.7	0/1(0.00%)	0/0(0.00%)	717 000	3
2008-10-27	5	0.05	8.54	9.3	1/1(100.00%)	2/4(50.00%)	102 000	5.14
2014-10-07	2	0.07	7.5	7.5	0/0(0.00%)	2/2(100.00%)	25 600	5.29
2012-01-03	20	0.32	5.21	4.3	1/18(5.56%)	0/2(0.00%)	767 000	4
2004-12-31	39	0.27	5.72	5	2/30(6.67%)	2/9(22.22%)	404 000	4.57
2006-08-14	73	0.57	5.45	5	4/58(6.90%)	3/15(20.00%)	12 700 000	4.5
2007-12-04	159	1.43	4.5	4.6	2/113(1.77%)	2/46(4.35%)	523 000	5.43
1998-12-27	246	1.13	4.78	4	31/212(14.62%)	11/34(32.35%)	5 670 000	5.86
2015-09-08	16	0.88	7.6	9.3	1/6(16.67%)	5/10(50.00%)	22 500 000	6.71
2005-12-31	975	7.25	9.19	10	8/102(7.84%)	142/873(16.27%)	124 000 000	7.5

Figure 5.14: Survey table with the average ratings of 12 respondents

Chapter 6

Practical application of the results

In order to use the vulnerability metrics that have been identified as metrics of interest during analysis there has to be a mathematical system in place to support them. This chapter provides some basic ideas for ways to implement such a system.

6.1 Predictive algorithms

As described in Section 3.7 attempts are being made at predicting the total number of vulnerabilities that will exist in a specific software using Vulnerability Discovery Models (VDMs) such as the ALM model. In order to estimate the three parameters of the ALM model non-linear regression analysis was used to fit the cumulative number of CVEs for OpenSSL and OpenSSH to the follwing model:

$$\Omega_{AML}(t) = \frac{B}{BC^{-ABt} + 1}$$

Table 6.1 shows the values that were obtained for the ALM parameters and r^2, the coefficient of determination(Section 2.4.4). Figures 6.1 and 6.2 give a visual representation of the cumulative number of CVEs and the fitting attempt for OpenSSL and OpenSSH respectively.

Table 6.1: Estimated parameter values and coefficient of determination

Software	A	B	C	r^2
OpenSSL	0.005	218.131	1.029	0.967
OpenSSH	0.022	66.48	1.045	0.915

Figure 6.1 shows that the OpenSSL data fit the model very well, with an r^2 value of 0.967 this means that 96.7% of the variance is explained by the model. The OpenSSH data did not fit the AML model as well as OpenSSL, this is evident when looking at Figure 6.2. Although its difficult to conclude what r^2 value is "good", an r^2 value of 0.915 seems to be a decent result. It is interesting to note that the OpenSSH data looks almost linear and thus a linear model might fit this data better, on the other hand a linear model would not fit the OpenSSL data.

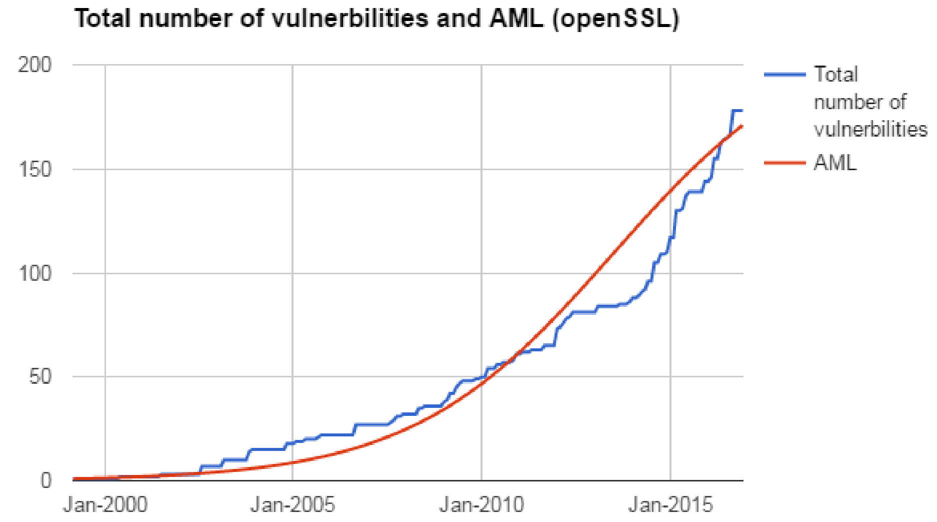


Figure 6.1: Fitting the AML VDM to the cumulative number of OpenSSL vulnerabilities

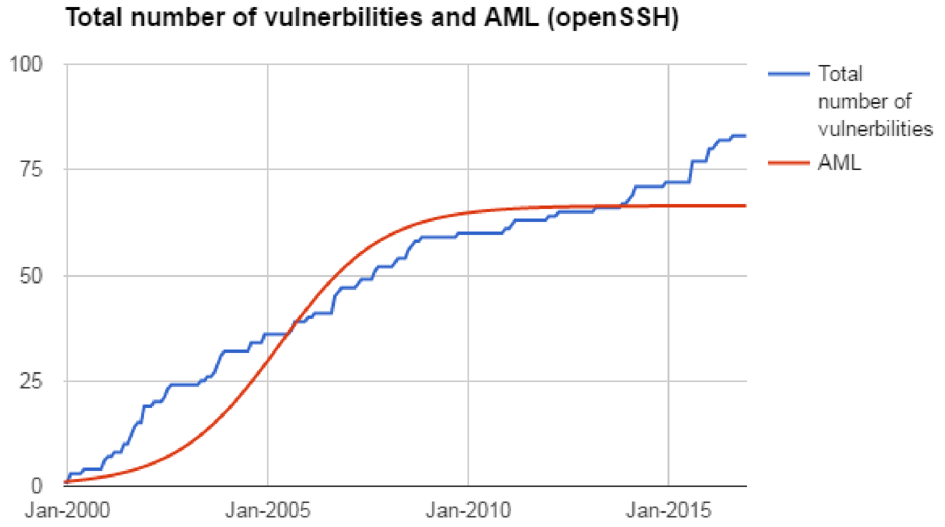


Figure 6.2: Fitting the AML VDM to the cumulative number of OpenSSH vulnerabilities

6.2 Scoring system development

In order to measure the security on a software level some metrics must be included that represent both the number of vulnerabilities discovered and the severity of those vulnerabilities. Since the analysis shows that more popular software have more vulnerabilities discovered in them, an additional sub-metric that measures the user numbers of the software should improve the accuracy and prevent the metric from heavily favoring unknown software. None of the metrics used in the suggested scoring systems have been weighted properly and the suggestions are ideas for someone who wishes to design their own system.

Each metric can be either:

- Collected automatically or collected manually.
- Relative to the other measurements or absolute.

The metrics that are focused on during the scoring system development in this thesis are automatically collected and absolute. However this does not mean that the final score has to be absolute since it can be made relative by normalizing. Among the metrics presented in Chapter 5 the following three have both of these attributes:

- Average number of CVEs per month: The average number of CVE entries published each month. This is the metric that will represent the quantity of vulnerabilities being discovered in the software. It is measured from the earliest CVE publishing date.
- Average CVSS score: The average CVSS score given by NVD to the CVEs used to calculate the metric above. This metric represents the severity of the vulnerabilities discovered. This metric can also be used to further tailor the metric to specific requirements, for example one could choose only to look at CVEs where the CVSS Access Vector is set to "Network".
- Popularity data: The popularity data gathered using Google and Bing will represent the user base size of the software.

Other metrics that do not have both of the desired attributes but that can still be included are:

- Days from vulnerability disclosure to patch: One of the most sought-after variables according to the survey. The data can be found manually and applied in a number of ways. An easy way would be to just use the average time. Another way would be to count the number of critical vulnerability that are not patched within a fixed time period (for example 2 days) and have each of those impact the score negatively.
- Days from disclosure to exploit release: If an exploit is released before the disclosure it could affect the score in a negative way. There can be many reasons why this happens of course, but none of them are positive.
- Days from CVE publishing date to exploit release: If a disclosure date is not available this metric could be used as an inferior alternative to the one

above. The data shows that most vulnerabilities have a CVE published within about two weeks after an exploit has been released, this could be used as a cut-off point for what is considered good disclosure. The reason that it is inferior to using a real disclosure date is that the software vendor is not the only entity responsible for the process of giving a vulnerability a CVE identifier and publishing the CVE. This makes it inconsistent.

- Percentage of vulnerabilities that have an available public exploit: Vulnerabilities with publicly available exploits are more likely to be used in attacks. This metric would also indicate the level of interest exploit developers have for the software in question. However as the data in table 5.6 shows the results of this metric can become very skewed when dealing with software that have few public vulnerabilities available.
- Days from exploit to patch: Based on the collected data and the analysis of the ”Encryption” category, patches are almost always available before the public release of an exploit. This makes the metric difficult to implement in a consistent manner but it could work similar to ”Days from disclosure to exploit” where every instance of this occurring would impact the final score.
- Documentation quality: Despite not finding an obvious correlation between the quality of documentation and the security of software, it could be included as a metric since the documentation is important if a developer wants to use software functions in a secure and proper way. There is no objective way to measure documentation quality and it is therefore up to the user of the metric to choose a reasonable rating system themselves, for example in this thesis the documentation was rated 0 to 3 based on some heuristics (see section 4.7.4). This is an example of a metric that is not automatically collected and therefore not used for any proposed metrics in this thesis.

6.2.1 Naive scoring system suggestion

The following is a very simplistic scoring system based on automatically collected data.

$$a = \text{Number of CVEs published per month}$$

$$b = \text{Average CVSS score for the CVEs used above}$$

$$c = \text{Google Hits}$$

$$score = a \cdot b \cdot \ln(c) \tag{6.1}$$

The idea being that the software is not only judged on the the number of vulnerabilities and their severity, but also its popularity. Popularity was included because it correlated with the number of high/critical vulnerabilities and exploits, no matter what this correlation is caused by. Table 6.2 shows the results of applying this system on the software in the encryption category.

Table 6.2: Score after applying the naive scoring system (lower is better)

Software	CVEs/Month	Avg. CVSS score	Google hits	Score
OpenSSL	0.813	5.413	16922222	73.247
GnuTLS	0.272	5.723	48455	16.793
LibreSSL	0.407	6.094	113333	28.865
OpenSSH	0.043	7.5	237000	3.991
WolfSSL	0.187	3.8	97500	8.163

6.2.2 Modular scoring system suggestion

This scoring system consists of two parts, one base score that can be automatically calculated based on public CVE data and a second part where manually collected data is included in the software comparison. The survey showed that different users look at different criteria, this sparked the idea for a modular system where additional metrics can be added to the system with a user-decided weight. A suggestion for such a system is the following:

$$\begin{gathered} score = ab + \frac{1}{N}\sum_{i=1}^{N} w_i x_i \\ 0 <= x_i <= 1 \end{gathered} \tag{6.2}$$

In the equation above a and b are the required base metrics that can be automatically collected. Like in Section 6.2.1 they represent the number of CVEs per month and their average severity. N is the number of additional metrics the user wants to include. These additional metrics are included as x_i and will have to be weighted and evaluated from a metric to metric basis. The weights are included as w_i. When deciding weights it is important to note that with this scoring system a lower score is better. This means that if a metric is included where a higher number is better the sign of w_i has to be negative. All included metrics x_i are restricted between 0 and 1 to easier gauge the metrics effect, one of the ways this can be accomplished is using a logistic function:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \tag{6.3}$$

where
x_0 = the x-value of the midpoint of the curve
L = the curve's maximum value, and
k = the steepness of the curve

As an example of using this scoring system, we add a popularity metric and include:

$$x_1 = \frac{1}{1 + e^{-10^{-5}(Google\_hits - 10^5)}}$$

Which is a logistic function restricted between 0 and 1 where a lower amount of Google hits will lower the final score. The variables of the function were chosen as to provide a decent distribution amongst the five software we apply it on, the same values would not work for more popular software. To increase the metrics importance the weight is set higher than one.

$$w_1 = 1.5$$

We then add a metric for vulnerability-to-patch duration:

$$x_2 = \frac{1}{1 + e^{-1(avg\_vulntopatch\_days - 5)}}$$

Here we simply use the average vulnerability to patch duration (measured from the disclosure date) and set the upper limit to 10 days (higher averages than 10 will still give the maximum sub-score of 1) in order to remove the effect of abnormally high averages that can be caused by data inconsistencies. We ignore all cases where the number of days is negative or zero. We also add a third metric that measures the percentage of vulnerabilities that have public exploits:

$$x_3 = exploit\_percentage$$

The exploit percentage metric will affect the scoring in the correct manner (lower is better) and this metric is already restricted between 0 and 1 since the percent is in decimal form. An additional idea could be to have the number of vulnerabilities affect this metric to make it more fair. To finalize the example we attach some weights to these metrics.

$$w_1 = 1.5, w_2 = 1, w_3 = 1.3$$

As an example, this is how the score of OpenSSL is calculated with these weights:

$$0.813 \cdot 5.413 + \frac{1}{3}(1.5 \cdot \frac{1}{1 + e^{-10^{-5}(16922222 - 10^5)}} + 1 \cdot \frac{1}{1 + e^{-1(22-5)}} + 1.3 \cdot 0.0593) = 5.25 \tag{6.4}$$

Table 6.3 shows the calculated parameters and the final score after applying this system on the "Encryption" software.

Table 6.3: Parameters and final score

Software	a	b	x_1	x_2	x_3	Score
OpenSSL	0.813	5.413	1	1	0.059	5.25
GnuTLS	0.272	5.723	0.374	1	0.1	2.12
LibreSSL	0.407	6.094	0.797	0	0.666	3.17
OpenSSH	0.043	7.5	0.533	1	0.104	0.96
WolfSSL	0.187	3.8	0.494	0	0	0.96

These results show that the final scores of OpenSSL, GnuTLS and LibreSSL are a fair bit higher than for OpenSSH and WolfSSL who both received 0.96 as their final score. The same final score is a surprising outcome given the fact that OpenSSH and WolfSSL had a noticeable difference in their variables entering into the scoring system. The score difference between the first three and other two is mainly caused by their a & b values (CVEs/Month and avg. CVSS) because those values were not restricted between 0 and 1 like the other metrics. This was the desired effect for this particular system since we value those variables higher than the others, but it could easily be offset by increasing the weights for the other metrics. The results also show that the software received either the maximum(1) or the minimum(0) in the "days from disclosure to patch" metric x_2, nothing in between. This is because according to the data that was collected almost all patches are released before or at the disclosure date and therefore ignored by us. A few of the patches that came late were very late and increased the average enough for x_2 to hit its ceiling. This shows that the implementation of the x_2 metric did not work well in this particular situation and a removal or modification of it is in order.

Since the final scores are not bound within a range it can be a good idea to normalize the results for an easier comparison, this will make the final scores relative to each other. This can be done for example using

$$x' = (x - \min(x))/(\max(x) - \min(x))$$

Through normalization the highest score (most dangerous software) of those we calculate will be 1 and the lowest (the safest) 0. Table 6.4 shows the results after normalization.

Table 6.4: Score results after normalization

Software	Normalized Score
OpenSSL	1
GnuTLS	0.493
LibreSSL	0.737
OpenSSH	0
WolfSSL	0

The weights used during this implementation are just suggestions based on observations during testing. This testing process is required when adding an additional metric and the weights should be tweaked based on how the user prioritizes the metric. It is without a doubt the hardest part about implementing a scoring system.

6.2.3 Scoring system with CVSS modification

Another scoring system suggestion is a system that is also based on the easily collected CVE and CVSS data but instead of adding metrics on top of that it uses the current CVSS sub-metrics, modifies their values based on manually collected data and recalculates the CVSS score. For the specifics of the CVSS score calculation refer to Section 2.3.1.

For example for every vulnerability where an exploit exists the CVSS sub-metric Access Vector is set to Network we will multiply the Attack Complexity sub-metric by 2. If the exploit was released before the vulnerability disclosure we instead multiply it by 4. Alternatively if the correct data is available, the CVSS temporal score could be calculated and modified with any metrics of interest.

6.2.4 IMDB inspired weighting

One idea for a software security metric is to weigh a softwares average CVSS score differently depending on the amount of CVEs are available for the software. This can be done with a so-called shrinkage estimator. One example of a website that uses this method is IMDB [58] where their rating depends not only on the average rating but also on the amount of votes. IMDB used to have the following formula public on their site:

$$\text{weighted rating} = (v/(v+m)) \cdot R + (m/(v+m)) \cdot C$$

Where:
R = average for the movie (mean) = (Rating)
v = number of votes for the movie = (votes)
m = minimum votes required to be listed in the Top 250 (currently 3000)
C = the mean vote across the whole report

This formula will lean towards the average rating across all movies C when there are less votes than the cut-off point m and more towards the average rating for the movie R when there are more votes. This formula can be directly applied on our software. Using mongoDBs aggregate framework we can run the following code to get the average CVSS score of all the CVEs in our database:

```
db.cves.aggregate([
    { "$match": {
        "cvss_score": { "$gt": 0 }
    }},
    { "$group": {
        "_id": null,
        "avg_cvss_score": { "$avg": "$cvss_score" }
    }}
])
```

This will match any CVE that has a CVSS score greater than 0, group them together and take the average of the cvss_score field. The output we receive is:

```
{ "_id" : null, "avg_cvss_score" : 6.257670867512725 }
```

We can use this value (rounded to 6.258) as our ”C” for the IMDB equation. For example Table 6.5 shows the result of applying this formula on the ”encryption” category with a cut-off point m=10. As can be seen in the table, software with low amounts of CVEs are closer to the database average (6.258).

Table 6.5: IMDB formula applied with m=10 (lower is better)

Software	Nbr of CVEs	Avg. CVSS score	IMDB formula score
OpenSSL	173	5.413	5.459
GnuTLS	39	5.723	5.832
LibreSSL	1	7.5	6.371
OpenSSH	83	6.094	6.112
WolfSSL	2	3.8	5.848

Chapter 7

Conclusions and further research

7.1 Summary of the results and conclusions

The larger vulnerability analysis of the different software categories shows a significant difference in average CVSS score and average number of CVEs per month between the different categories. This is attributed to differences in user base size and developer focus. Categories such as ”Firewalls” have less average confidentiality impact, integrity impact and availability impact compared to more popular software categories such as ”Video” or ”Communication”. Apart from the theories above it is also suggested that this difference may be affected by some of the chosen software having Windows versions. In retrospect this study between categories would have had more trustworthy results by limiting all of the software to one operating system.

In the more detailed software analysis a finding that is backed-up by other research is that the size of the user base for a piece of software correlates with the number of vulnerabilities in software. It is important to note that this does not mean that a less used software is more secure, only that fewer vulnerabilities are found and disclosed. In theory, this could decrease the number of exploitation attempts from ”script-kiddies” who only use publicly available exploits but it is unlikely to protect from any kind of targeted attack.

Using the average CVSS score as a metric for software security is less reliable for unpopular software because of the large gap in CVE amount between less popular and popular software. Using weighting to make sure software with fewer CVEs moves closer to the overall average CVSS score is a viable way of making this gap smaller.

Among the other collected data such as documentation quality and Github statistics no obvious correlation was found with the number of vulnerabilities in the software.

The small survey showed multiple things. First of all that there is not any form of consensus as to which metrics included in the survey were the most important or least important, this outcome could potentially have been different with a larger number of respondents but it is likely that there is disagreement in general among experts. However what was agreed upon is that there was information missing in the survey. Most of the suggestions for additional metrics were similar to the ones examined in the detailed analysis section, for example time to patch and open source data like code complexity. At the time of writing these metrics are not available anywhere freely and therefore a scoring system based on automatically collected data is unlikely to be developed without a serious time investment. Most of the respondents answered that they would evaluate the software manually looking more closely at things such as what parts of the software they have to use, the scope of the software and how business critical it is which also indicates to me that such a scoring system might not even be the best tool for the job. Instead a service where all relevant data was gathered and presented to the user, without evaluating it, might be a solution more in line with the requirements of end-users. After all, the data gathering process is time-consuming.

7.2 Thoughts on future research

NIST published a report in 2010 [59] that identifies the following five research areas that could help progressing the state of the art in security metrics.

- Formal Models of Security Measurement and Metrics
- Historical Data Collection and Analysis
- Artificial Intelligence Assessment Techniques
- Practicable Concrete Measurement Methods
- Intrinsically Measurable Components.

Out of these five points this thesis focused on historical data collection and analysis. Similar to the Policy Security Score in Section 3.5 the main idea was to use historical vulnerability data combined with CVSS. This approach is an interesting one and one worth exploring further. There are a few challenges, the main one being data collection and the lack of data in general. Most of the research in this field, including this thesis, is heavily dependent on the CVE identifiers and the databases that collect information about them. However these databases lack important data such as exploit/patch dates and the CVE identifiers themselves do not represent all the vulnerabilities that exist, they are far from optimal. This was more apparent when OSVDB (Open Source Vulnerability Database) was still active since they had a large number of vulnerabilities and exploits (with dates) without CVE identifiers. Many vulnerabilities still have no identifier or reference other than the OSVDB ID. The content of OSVDB was commercial only (the ”Open” part of the website title is rather inappropriate) which makes hosting a mirror of OSVDB illegal. As such the closing of OSVDB was a big loss (of data) for everyone researching exploits and vulnerabilities.

If further progress is to be made towards a fully automated scoring system, there will need to be more data easily available online. Vulnerability data, what specific software the vulnerabilities affect (with build numbers), software data about the specific builds (release date, user data etc), patch frequency, developer data. The more data the better. If a large amount of data was easy to collect, an interesting approach would be using machine learning and apply some clustering method or other ”big data”-approach. Using machine learning could be very effective with a large amount of data to find which metrics are interesting and which metrics can be ignored. Another approach could use an effort-based vulnerability discovery model, like the folded VDM mentioned in 3.7 to automate the process of predicting the number of vulnerabilities in the future based on the number of vulnerabilities per "user-month". Overall software security is a complex system depending on a large number of variables. Similar to predicting the weather, it is unlikely that the problem can be solved with a simple algorithm.

Additionally if any security metric is to be developed with a focus on Internet of Things, measuring the potential environmental impact is a must. A suggestion for this is creating a completely separate scoring system that is applied on the product the software is used in. This scoring-system could for example range from 0 (harmless) to 100 (loss of human life) and be used as a weight on the severity of any vulnerabilities found within the measured product.

References

[1] *Internet of Things growth prediction.* URL: `http://www.gartner.com/newsroom/id/2636073` (visited on 2017-05-26).

[2] *Heartbleed.* URL: `http://heartbleed.com/` (visited on 2017-05-26).

[3] *Wikipedia on vulnerability(computing).* URL: `https://web.archive.org/web/20170429232133/https://en.wikipedia.org/wiki/Vulnerability_(computing)` (visited on 2017-02-05).

[4] *Common Vulnerabilities and Exposures.* URL: `https://cve.mitre.org/` (visited on 2016-08-26).

[5] *CVE Numbering Authorities.* URL: `https://cve.mitre.org/cve/cna.html` (visited on 2016-09-13).

[6] *CVE FAQ Question A6.* URL: `https://cve.mitre.org/about/faqs.html#a6` (visited on 2016-09-13).

[7] *National Vulnerability Database.* URL: `https://nvd.nist.gov/home.cfm` (visited on 2016-08-26).

[8] *Common Weakness Enumeration.* URL: `https://cwe.mitre.org/` (visited on 2016-08-26).

[9] *CWE FAQ Question A2.* URL: `https://cwe.mitre.org/about/faq.html#A.2` (visited on 2016-09-13).

[10] *Vulndb.* URL: `https://www.riskbasedsecurity.com/vulndb/` (visited on 2016-09-15).

[11] *CVE Details.* URL: `http://www.cvedetails.com/` (visited on 2016-09-15).

[12] *IBM X-Force Exchange.* URL: `https://exchange.xforce.ibmcloud.com` (visited on 2016-11-28).

[13] *vulndb 2015 vulnerability trends.* URL: `https://www.riskbasedsecurity.com/vulndb-quickview-2015-vulnerability-trends/` (visited on 2016-09-15).

[14] *Risk Based Security: PreBreach Risk Reduction Through Data Analysis.* URL: http://web.archive.org/web/20160915124408/http://banking-insurance.cioreview.com/vendor/2016/risk_based_security (visited on 2016-09-15).

[15] Fabio Massacci and Viet Hung Nguyen. *Which is the Right Source for Vulnerability Studies? An Empirical Analysis on Mozilla Firefox.* Tech. rep. University of Trento, Italy, 2010. DOI: 10.1145/1853919.1853925. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.452.9317&rep=rep1&type=pdf.

[16] *Seclists.* URL: http://seclists.org/ (visited on 2016-09-15).

[17] *Bugtraq.* URL: http://seclists.org/bugtraq/ (visited on 2016-08-26).

[18] *Full Disclosure Mailing List.* URL: http://seclists.org/fulldisclosure/ (visited on 2016-09-15).

[19] Elizabeth Chew et al. *Performance measurement guide for information security.* Tech. rep. National Institute of Standards and Technology, Gaithersburg, 2008. URL: http://csrc.nist.gov/publications/nistpubs/800-55-Rev1/SP800-55-rev1.pdf.

[20] George Jelen. *SSE-CMM Security Metrics.* Tech. rep. National Institute of Standards et al., 2000.

[21] *Common Vulnerability Scoring System version 2 specification.* URL: https://www.first.org/cvss/cvss-v2-guide.pdf (visited on 2016-09-13).

[22] *Common Vulnerability Scoring System version 3 specification.* URL: https://www.first.org/cvss/cvss-v30-specification-v1.7.pdf (visited on 2016-09-13).

[23] *CVSS and the Internet of Things.* URL: http://web.archive.org/web/20160810152902/https://insights.sei.cmu.edu/cert/2015/09/cvss-and-the-internet-of-things.html (visited on 2016-09-15).

[24] *CERT/CC Vulnerability Metric.* URL: http://www.kb.cert.org/vuls/html/fieldhelp#metric (visited on 2016-09-18).

[25] Elisa Bertino et al. *Security for Web Services and Service-Oriented Architectures.* 2009, pp. 43–44.

[26] *Security Bulletin Severity Rating System.* URL: https://technet.microsoft.com/en-us/security/gg309177.aspx (visited on 2016-09-22).

[27] *Redhat Severity Ratings.* URL: https://access.redhat.com/security/updates/classification/ (visited on 2016-09-22).

[28] *R.* URL: https://cran.r-project.org/ (visited on 2017-04-17).

[29] *IBM SPSS Statistics.* URL: https://www.ibm.com/us-en/marketplace/spss-statistics (visited on 2017-05-15).

[30] Georgios Spanos, Angeliki Sioziou, and Lefteris Angelis. "WIVSS: a new methodology for scoring information systems vulnerabilities". In: *Proceeding PCI '13 Proceedings of the 17th Panhellenic Conference on Informatics* (2013), pp. 83–90. DOI: 10.1145/2491845.2491871.

[31] Yulong Wang and Yi Yang. *PVL: A Novel Metric for Single Vulnerability Rating and Its Application in IMS.* Tech. rep. State Key Laboratory of Networking, Switching Technology, Beijing University of Posts, and Telecommunications, 2012. URL: http://archive.is/G3VPT.

[32] Qixu Liu and Yuqing Zhang. "VRSS: A new system for rating and scoring vulnerabilities". In: *Computer Communications Volume 34 Issue 3, March* (2011), pp. 264–273. DOI: 10.1016/j.comcom.2010.04.006.

[33] Qixu Liu et al. "Improving VRSS-based vulnerability prioritization using analytic hierarchy process". In: *Journal of Systems and Software Volume 85 Issue 8, August* (2012), pp. 1699–1708. DOI: 10.1016/j.jss.2012.03.057.

[34] *OWASP Risk Rating Methodology.* URL: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology (visited on 2016-09-18).

[35] Muhammad Abedin et al. *Vulnerability Analysis For Evaluating Quality of Protection of Security Policies.* Tech. rep. Department of Computer Science, The University of Texas at Dallas, 2006. URL: https://www.utdallas.edu/~lkhan/papers/Abedin2006.pdf.

[36] Lingu Wang et al. "k-Zero Day Safety: A Network Security Metric for Measuring the Risk of Unknown Vulnerabilities". In: *IEEE Trans. Dependable Sec. Comput 11(1)* (2014), pp. 30–44. URL: http://csrc.nist.gov/staff/Singhal/ieee_tdsc_2013_final_version.pdf.

[37] Omar Alhazmi, Yashwant Malaiya, and Indrajit Ray. *Security Vulnerabilities in Software Systems: A Quantitative Perspective.* Tech. rep. Department of Computer Science, Colorado State University, 2005. URL: http://www.cs.colostate.edu/~malaiya/635/IFIP-10.pdf.

[38] Eric Rescorla. "Is finding security holes a good idea?" In: (). URL: https://www.dtc.umn.edu/weis2004/rescorla.pdf.

[39] Jinyoo Kim, Yashwant K. Malaiya, and Indrakshi Ray. "Security in Open versus Closed Systems – The Dance of Boltzmann, Coase and Moore". In: (). URL: http://emidio.planamente.ch/docs/linux/toulouse.pdf.

[40] HyunChul Joh, Jinyoo Kim, and Yashwant K. Malaiya. *Vulnerability Discovery Modeling using Weibull Distribution.* Tech. rep. Department of Computer Science, Colorado State University, 2008. URL: http://www.cs.colostate.edu/~malaiya/pub/weibull08.pdf.

[41] O. H. Alhazmi and Y. K. Malaiya. *Modeling the Vulnerability Discovery Process.* Tech. rep. Department of Computer Science, Colorado State University, 2005. URL: http://www.cs.colostate.edu/~malaiya/pub/issre05.pdf.

[42] HyunChul Joh and Yashwant K. Malaiya. *Modeling Skewness in Vulnerability Discovery Models in Major Operating Systems.* Tech. rep. Department of Computer Science, Colorado State University, 2010. URL: http://www.cs.colostate.edu/~malaiya/p/joh.skewness.2010.pdf.

[43] "Quantifying the security risk of discovering and exploiting software vulnerabilities". In: (2016), pp. 19–21. URL: https://dspace.library.colostate.edu/bitstream/handle/10217/176641/Mussa_colostate_0053A_13672.pdf?sequence=1.

[44] Ali Mosleh, E. Richard Hilton, and Peter S. Browne. *Bayesian probabilistic risk analysis.* Tech. rep. 1985, pp. 5–12. DOI: 10.1145/1041838.1041839. URL: http://oplab.im.ntu.edu.tw/download/pubication/journal/J43_2013_A%20novel%20approach%20to%20evaluate%20software%20vulnerability%20prioritization.pdf.

[45] Peng Xie et al. "Using Bayesian Networks for Cyber Security Analysis". In: (). URL: http://people.cs.ksu.edu/~xou/publications/dsn10_preprint.pdf.

[46] Maxwell Dondo. *A Vulnerability Prioritization System Using A Fuzzy Risk Analysis Approach.* Tech. rep. 2008. URL: http://cradpdf.drdc-rddc.gc.ca/PDFS/unc112/p533528_A1b.pdf.

[47] Chien-Cheng Huang et al. *A novel approach to evaluate software vulnerability prioritization.* Tech. rep. 2013, pp. 2822–2840. DOI: 10.1016/j.jss.2013.06.040. URL: http://oplab.im.ntu.edu.tw/download/pubication/journal/J43_2013_A%20novel%20approach%20to%20evaluate%20software%20vulnerability%20prioritization.pdf.

[48] Muhammad Shahzad, M. Zubair Shafiq, and Alex X. Liu. *A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles.* Tech. rep. Department of Computer Science and Engineering Michigan State University, 2012. URL: https://www.cse.msu.edu/~alexliu/publications/VulnerabilityDB/VulnerabilityDB_ICSE2012.pdf.

[49] Stefan Frei et al. *Large-Scale Vulnerability Analysis*. Tech. rep. Computer Engineering and Networks Laboratory ETH Zurich, Switzerland, 2006. URL: `http://x3.techzoom.net/Papers/Large_Scale_Vulnerability_Analysis_(2006).pdf`.

[50] *Exploit-DB*. URL: `https://www.exploit-db.com/` (visited on 2016-11-21).

[51] *openSSL Security Advisory*. URL: `https://www.openssl.org/news/secadv/` (visited on 2017-01-16).

[52] *Exploit-DB github*. URL: `https://github.com/offensive-security/exploit-database` (visited on 2017-01-27).

[53] *openSSL Documentation*. URL: `https://www.openssl.org/docs/` (visited on 2017-01-17).

[54] *gnuTLS Documentation*. URL: `http://gnutls.org/documentation.html` (visited on 2017-01-17).

[55] *libreSSL Documentation*. URL: `https://fossies.org/dox/libressl-2.5.0/` (visited on 2017-01-17).

[56] *openSSH Documentation*. URL: `https://www.openssh.com/manual.html` (visited on 2017-01-17).

[57] *wolfSSL Documentation*. URL: `https://www.wolfssl.com/wolfSSL/Docs.html` (visited on 2017-01-17).

[58] *Exploit-DB*. URL: `http://www.imdb.com/` (visited on 2016-12-05).

[59] Wayne Jansen. *Directions in Security Metrics Research*. Tech. rep. National Institute of Standards and Technology, 2010, p. 16.

Appendix A

Software selection and categories

List of selected categories and software. The software are listed by their search keyword in the database, most commonly ”vendor:product”.

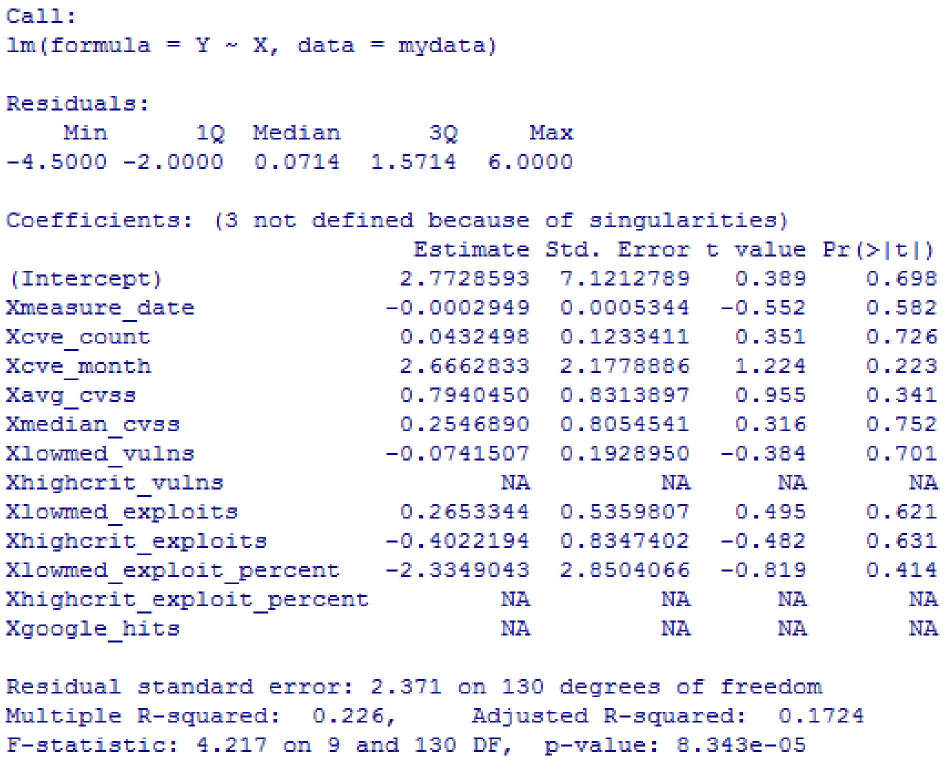
- **Video players**
  - ffmpeg:ffmpeg
  - apple:quicktime
  - Libav:libav
  - videolan:vlc_media_player
  - qemu
- **HTTP-Servers**
  - apache:Http_Server
  - nginx:nginx
  - Rejetto:Http_File_Server
  - IBM:Http Server
  - Lighttpd
  - Monkey-project:Monkey_Http_Daemon
- **Encryption**
  - openbsd:OpenSSL
  - gnu:GnuTLS
  - openbsd:OpenSSH
  - openbsd:LibreSSL
  - WolfSSL:WolfSSL
- **Script languages**
  - PHP:PHP
  - rubyonrails:Ruby_On_Rails
  - perl:perl

  - microsoft:Asp.net

- **Firewalls**
  - Pfsense:Pfsense
  - Netfilter_Core_Team:Iptables
  - Ipcop:Ipcop
  - Comodo Internet Security
- **Virtualization**
  - vmware:player
  - docker:docker
  - microsft:Hyper-V
  - oracle:Vm_Virtualbox
  - openvz
  - xen:xen
- **FTP-Servers**
  - CoreFtp:core_FTP
  - Proftpd:ProFTPD
  - Glftpd:Glftpd
  - Pureftpd
  - beasts:vsftpd
  - Zftpserver
- **Communication (chat programs)**
  - kvirc:kvirc
  - irssi:irssi
  - cisco:spark
  - Skype:Skype
  - Microsoft:Skype_For_Business
- **SQL-Servers**
  - microsoft:sql_server
  - mysql:mysql
  - postgresql:postgresql

Appendix B

Regression results



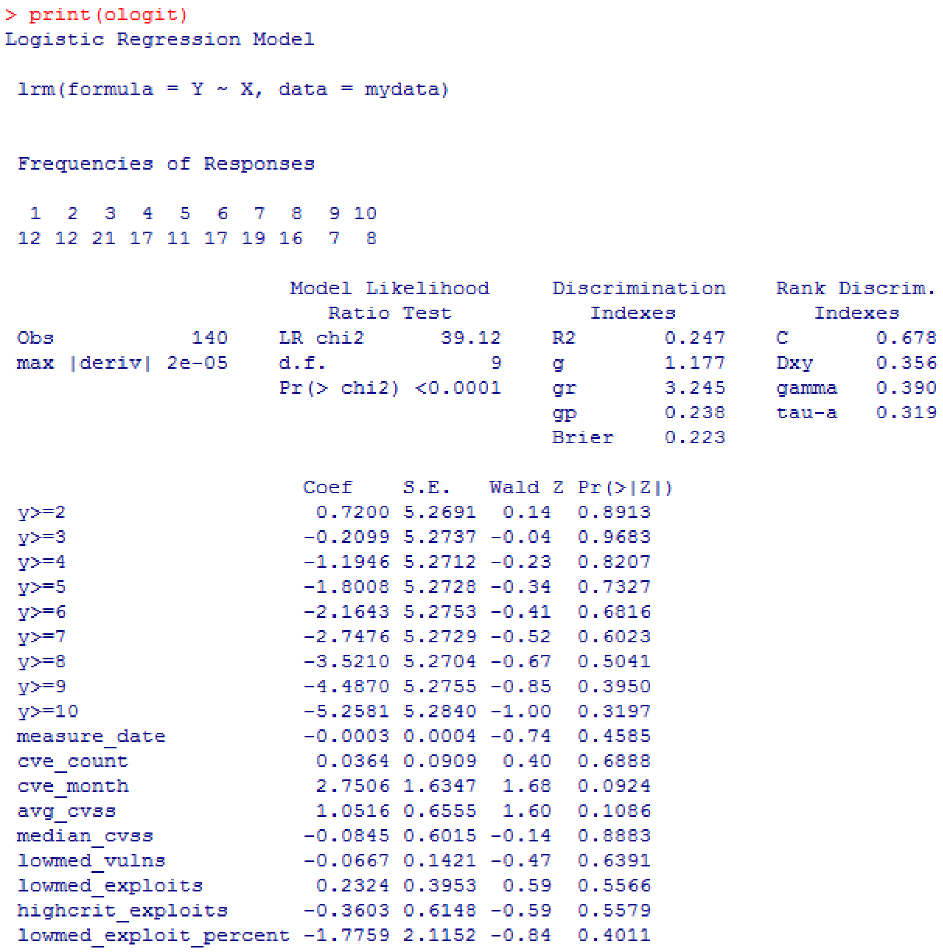
```
Call:
lm(formula = Y ~ X, data = mydata)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.5000 -2.0000  0.0714  1.5714  6.0000 

Coefficients: (3 not defined because of singularities)
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                2.7728593  7.1212789   0.389    0.698
Xmeasure_date             -0.0002949  0.0005344  -0.552    0.582
Xcve_count                 0.0432498  0.1233411   0.351    0.726
Xcve_month                 2.6662833  2.1778886   1.224    0.223
Xavg_cvss                  0.7940450  0.8313897   0.955    0.341
Xmedian_cvss               0.2546890  0.8054541   0.316    0.752
Xlowmed_vulns             -0.0741507  0.1928950  -0.384    0.701
Xhighcrit_vulns                   NA         NA      NA       NA
Xlowmed_exploits           0.2653344  0.5359807   0.495    0.621
Xhighcrit_exploits        -0.4022194  0.8347402  -0.482    0.631
Xlowmed_exploit_percent   -2.3349043  2.8504066  -0.819    0.414
Xhighcrit_exploit_percent         NA         NA      NA       NA
Xgoogle_hits                      NA         NA      NA       NA

Residual standard error: 2.371 on 130 degrees of freedom
Multiple R-squared:  0.226,	Adjusted R-squared:  0.1724 
F-statistic: 4.217 on 9 and 130 DF,  p-value: 8.343e-05
```

Figure B.1: Regression analysis summary of the survey results



```
> print(ologit)
Logistic Regression Model

 lrm(formula = Y ~ X, data = mydata)


 Frequencies of Responses

  1  2  3  4  5  6  7  8  9 10
 12 12 21 17 11 17 19 16  7  8

                      Model Likelihood     Discrimination      Rank Discrim.
                         Ratio Test            Indexes            Indexes
 Obs             140    LR chi2      39.12    R2        0.247    C       0.678
 max |deriv| 2e-05     d.f.             9    g         1.177    Dxy     0.356
                        Pr(> chi2) <0.0001    gr        3.245    gamma   0.390
                                              gp        0.238    tau-a   0.319
                                              Brier     0.223

                        Coef    S.E.   Wald Z Pr(>|Z|)
 y>=2                    0.7200 5.2691  0.14  0.8913
 y>=3                   -0.2099 5.2737 -0.04  0.9683
 y>=4                   -1.1946 5.2712 -0.23  0.8207
 y>=5                   -1.8008 5.2728 -0.34  0.7327
 y>=6                   -2.1643 5.2753 -0.41  0.6816
 y>=7                   -2.7476 5.2729 -0.52  0.6023
 y>=8                   -3.5210 5.2704 -0.67  0.5041
 y>=9                   -4.4870 5.2755 -0.85  0.3950
 y>=10                  -5.2581 5.2840 -1.00  0.3197
 measure_date           -0.0003 0.0004 -0.74  0.4585
 cve_count               0.0364 0.0909  0.40  0.6888
 cve_month               2.7506 1.6347  1.68  0.0924
 avg_cvss                1.0516 0.6555  1.60  0.1086
 median_cvss            -0.0845 0.6015 -0.14  0.8883
 lowmed_vulns           -0.0667 0.1421 -0.47  0.6391
 lowmed_exploits         0.2324 0.3953  0.59  0.5566
 highcrit_exploits      -0.3603 0.6148 -0.59  0.5579
 lowmed_exploit_percent -1.7759 2.1152 -0.84  0.4011
```

Figure B.2: Summary of the regression with an ordered logistic model

Printed by Tryckeriet i E-huset, Lund 2017

LUND UNIVERSITY

Series of Master’s theses
Department of Electrical and Information Technology
LU/LTH-EIT 2017-598

http://www.eit.lth.se