Methodological Considerations in Exploit Prediction Systems

Sam van Hooff

Master of Science Thesis

TU Delft

METHODOLOGICAL CONSIDERATIONS IN EXPLOIT PREDICTION SYSTEMS

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
4TU Cyber Security Specialisation

by

Sam van Hooff
4247620

June 2022

The work in this thesis is done in the:

Cyber Security Group
Intelligent Systems Department
Faculty of EEMCS
Delft University of Technology

In cooperation with:

Adyen
Cyber Security Team

Committee Members:	Prof.dr. Mauro Conti	(Chair)
	Dr. Alex Stefanov	
	Dr. Kaitai Liang	
	Dr. Chhagan Lal	(Co-supervisor)
	Mr. Ignacio Jimenez Pi	(Company supervisor)

ABSTRACT

The growing number of software vulnerabilities being disclosed is posing a challenge to many organisations. With limited patching resources and only a fraction of the vulnerabilities posing a *real* threat, prioritization is key. Current prioritization methods, such as CVSS, are failing and are sometimes no better than random guessing. Exploit Prediction Systems (EPS) try to fill this gap leveraging a data-driven approach. Related works in the exploit prediction domain make EPS design decisions based on different methodological assumptions. Some of these assumptions are unrealistic or faulty, yielding models that fail to represent a real world situation. The first contribution of this thesis is the identification of critical methodological assumptions in EPS design and the magnitude of their effects. Then, as second contribution, EPS performance is optimized under restricting yet realistic circumstances, by exploring different techniques to handle class-imbalance, creating richer textual features and/or leveraging different prediction algorithms. The third contribution of this thesis is the implementation of an open-source framework that enables easy experimentation with different machine learning techniques for exploit prediction.

Six critical methodological assumptions have been identified in the area of realistic data collection, correct processing of data, and proper model evaluation. Experiments show that when adhering to the most realistic assumptions, only a fraction of the predictive power of the evaluated EPS is sustained. Almost all prior works fall victim to at least one faulty or unrealistic assumption, and thereby report overoptimistic results.

Substantial improvements are achieved in the optimization step of this thesis. With an optimized EPS with a F1-score of 0.366, performance is insufficient to justify its deployment in a production environment. With the current level of maturity, exploit prediction could have value as a complementary measure to existing vulnerability prioritization systems. Further improvements and more transparent systems are essential for EPS to be suitable for practical usage.

PREFACE

Before you lies the thesis "Methodological Considerations in Exploit Prediction Systems", a research project in which design decisions in Exploit Prediction Systems are identified, evaluated and optimized. The thesis represents my graduation project for the Master's Degree Computer Science at the EEMCS faculty of Delft University of Technology. Over the last months, I conducted my graduation research in cooperation with Adyen. I would like to make use of the opportunity to thank a few people that have proven themselves pivotal in the process of writing this thesis.

First of all I would like to thank my supervisor Mauro Conti and co-supervisor Chhagan Lal, for the guidance and feedback. I think we did well as an international team often communicating from 3 different countries. Also, I would like to thank my two other committee members, Alex Stefanov and Kaitai Liang, for the time and commitment to my graduation.

Secondly, I would like to express my gratitude to Ignacio Jimenez Pi for supporting me in navigating Adyen, and involving me in your passion project of vulnerability management. Although some pivoting has taken place over the time, the basis of this thesis lies there. The weekly meetings where helpful and interesting, and I appreciate that you kept making time for me, also in your new role.

Finally, I would like to thank my family, friends and above all my girlfriend. A special acknowledgement should be extended to my Grandfather, a vibrant 94 year old, who promised me to be present the day I graduate. Even though the delay, he kept his word, awarded with being eternalized in the TU Delft Repository. I propose we set a new deal soon!

Sam van Hooff
28-07-2022
Amsterdam

CONTENTS

1 INTRODUCTION

Timely patching —the updating of vulnerable software— of software pieces running on the infrastructure of organizations is of critical importance. Exploitation of vulnerable software can cause major losses and might in some cases even endanger business continuity. An example of the devastating impact a vulnerability can have was seen in 2017, when ransomware malware named Wannacry hit thousands of institutions across the globe paralysing their computer systems. Among the victims where hospitals, critical infrastructure and commercial companies. The financial loss as a result of this cyber-attack is estimated at 4 billion USD [Jonathan, 2017]. This malware took advantage of a vulnerability in Microsoft Windows. Microsoft had released a security patch fixing this vulnerability two months before Wannacry hit, which means that all this could have been prevented by timely deployment of patches [Microsoft, 2017]. This does not only hold for the big, well known, breaches; research reports that more than half of the data breaches are the result of exploited vulnerabilities for which a patch has already been released [Ponemon, 2020]. This shows how great the impact of a(n) (im-)proper patching strategy can be.

1.1 BACKGROUND

The number of disclosed vulnerabilities has been growing vastly and consistently over the past few years, which makes patching management increasingly challenging [MITRE, 2022b]. Organisations typically have more exposed vulnerabilities than resources to fix them. Because the process of patching is resource intensive, patching everything straight away is highly inefficient. This means that organisations have to make decisions about what to patch, and what not to patch. This decision is a trade-off between coverage (how much of all vulnerabilities are remediated) and efficiency (of the remediated vulnerabilities how much of this effort was in vain).

Risk based prioritization of patches can help facilitating this decision, by choosing a patching strategy which starts with the remediation of vulnerabilities with the highest risk. Currently the most widely adopted technique to asses this risk is the Common Vulnerability Scoring System (CVSS) Base score, even though it is not meant to reflect the overall risk and fails to measure the threat of a vulnerability or if it will be exploited [Allodi and Massacci, 2014]. A common approach is to remediate all vulnerabilities which have a CVSS Base score higher than a certain threshold. Even security standards adopt this approach. For example the payment card industry data security standard (PCI-DSS) requires vulnerabilities with a CVSS Base score higher than 4.0 to be remediated by organizations storing or processing credit card data [PCI, 2018].

CVSS has long been criticized for being unable to indicate the exploitability of vulnerabilities. Patching prioritization based on the CVSS Base score is often a sub optimal strategy, and in some cases even no better than the random choice [Dey et al., 2015; Beattie et al., 2002; Allodi and Massacci, 2014]. So this score is not an effective measure of risk, but is often interpreted and used as such, resulting in a distorted view of risk and inefficient resource allocation in patching management.

1.2 PROBLEM

The main reason the established approaches are ineffective is that organisations can't effectively assess whether a given vulnerability poses a meaningful threat. Only a fraction of all disclosed vulnerabilities ever have exploit code written to exploit them, and only a fraction thereof ever gets exploited in the wild. The fraction of vulnerabilities exploited in the wild varies per research, but is ranging between 1.4% and 5.5% of all disclosed vulnerabilities [Bozorgi et al., 2010; Jacobs et al., 2020]. Patching effort could be drastically reduced if these exploitable vulnerabilities can be identified. Exploit Prediction Systems (EPS) try to leverage this and find these exploitable vulnerabilities.

Predicting which vulnerabilities will be exploited in the wild is not trivial. Because the scarcity of (high-quality) data and the imbalanced nature of the problem, the conditions for exploit prediction are more challenging compared to other machine learning applications. Obtaining data about a vulnerability being exploited in the wild is very difficult. Exploit data is not broadly collected and (mostly commercial) owners are often not keen on sharing it, also because of its sensitive nature [CISA, 2015]. This is one of the reasons that data used for exploit prediction is often severely imbalanced, i.e. containing only a very small fraction of positive samples. The other reason is that simply the most vulnerabilities never get exploited in the wild [Nayak et al., 2014]. Regardless of the cause, exploit prediction is a problem which is heavily imbalanced. Most classification algorithms don't perform well if classes are not roughly equally distributed. This can be made explicit if one considers a binary classification task with 95 negative samples and 5 positive sample. A machine learning classifier might learn to classify all samples as negative, resulting in a classification accuracy of 95% without ever correctly classifying a positive sample.

The first exploit prediction system has been developed about a decade ago, and after that many followed. They vary in what data they use as ground truth for their models, what data is used to create features, what kind of machine learning techniques are being used, how the data is (pre-)processed and even how the performance of the model is evaluated. These design decisions are based on several methodological assumptions about the context of the prediction task. These assumptions differ greatly among different existing exploit prediction system, ranging from fairly realistic to simply faulty.

1.3 CONTRIBUTIONS

Researchers in the field tend to make concessions in the methodological assumptions they base their work on, often boosting the performance of the exploit prediction systems they develop [Bullough et al., 2017]. Several of these assumptions critically affect the evaluation of their predictive models, possibly undermining the performance results altogether. The focus of this thesis is to bring these methodological concessions to light, and to propose how the harsh conditions in exploit prediction can be overcome while complying to more realistic methodological assumptions. This leads to the following contributions:

- Related works will be analysed to identify critical methodological assumptions on which EPS are based and how often they are observed (Chapter 3). The effects of the identified methodological assumptions on EPS performance will be evaluated by conducting experiments (Chapter 5). Several existing experiments from Reinthal et al. [2018] and Bullough et al. [2017] are reproduced to investigate conflicting results.

- An exploit prediction system will be developed and optimized based on restrictive yet realistic assumptions, improving on earlier work where such a system is built under similar methodological constraints. Optimization is done by focusing on the handling of class-imbalance (Section 5.2.2), the design of more advanced textual features (Section 5.2.3), and by exploring and tuning of different machine learning algorithms (Section 5.2.4).
- A framework is implemented with the goal of enabling easy, automated experimentation with different machine learning and optimization techniques. The source-code of this framework will be open-sourced to facilitate future research in this domain (Section 4.1.3).

Invalid or unrealistic methodological assumptions lead to models which are over-optimistic about their ability to predict exploits [Bullough et al., 2017]. Not only will this lead to an unequal playing field for academia, it also results in a false sense of security when such systems are deployed in production. Setting a mutual standard for methodological assumptions in exploit prediction, is a first step in overcoming these issues.

1.4 COMPANY PERSPECTIVE

This thesis research is conducted in cooperation with industry partner Adyen. Adyen is one of the world's largest payment service providers, with an annual processed volume of over 300 billion euros [Adyen, 2020]. They offer end-to-end solutions to enable businesses to accept payments using over 250 payments methods in more than a 150 countries. Adyen operates in a strictly regulated banking industry and handles highly sensitive payment data, leaving little to no room for error. Therefore, they take patching management very seriously. Adyen has expressed its interest in the exploration of opportunities lying in risk based patch prioritization by exploit prediction, to further improve their patching process. It goes without saying that the performance evaluation of such a system should aspire to be representative of the performance in a real-world environment.

1.5 THESIS STRUCTURE

The rest of the thesis is organized as follows. In Chapter 2, the background of the problem is outlined. Chapter 3 contains an extensive analysis of related works in the exploit prediction field, with the goal of identifying critical methodological assumptions. Next, the method for the proposed work of this thesis is laid out in Chapter 4, where the experimental setup and EPS optimization strategies are presented. Following on this, Chapter 5 contains the results of these analyses. Lastly, the conclusion, limitations and recommendations about future work, can be found in Chapter 6.

2 BACKGROUND

The world of patching, vulnerabilities and exploits is complex. This complexity can cloud effective decision making on how to avoid loss due to (data-)breaches. In this background chapter it is attempted to provide a high level and complete picture of the vulnerability landscape (Section 2.1) and the place patch prioritization has in this context (sections 2.2 and 2.3).

2.1 VULNERABILITY LANDSCAPE

To get a feel for the vulnerability landscape the important players active in it are introduced in Section 2.1.1. After this, the different phases a vulnerability typically goes through are analysed in Section 2.1.2. And lastly, in Section 2.1.3, attention is being paid to the challenges organisations face with respect to vulnerability management.

2.1.1 Players

The backbone of the vulnerability landscape is the Common Vulnerabilities and Exposures (CVE) system. This system, maintained by MITRE, is a public database in which all known vulnerabilities are documented. A unique CVE-ID is issued for each new vulnerability. This identifier is used widely in the vulnerability domain to aggregate and share information about the vulnerability.

The National Vulnerability Database (NVD) is a database fully synchronized with the CVE database. The NVD, maintained by NIST, provides additional information about the vulnerabilities assigned with a CVE-ID. Entries contain information about the affected products, references to additional information sources and probably the most important, a severity score. This score is called the Common Vulnerability Scoring System (CVSS) and will be explained in more detail in Section 2.2.1.

Another important organisation is ExploitDB. The ExploitDB (EDB) is one of the most extensive databases containing Proof-of-Concept exploit code (exploits). This database is crowd sourced and has largely adopted CVE-IDs to identify the exploits on it.

2.1.2 Vulnerability life-cycle

A software vulnerability is defined as a flaw or error in the source code of software, which enables an attacker who exploits this vulnerability, to gain control over the system the software runs on. This flaw can be introduced by a human coding or design mistake, but also for example through the use of third party libraries. An attacker generally makes use of certain code to exploit a vulnerability, this code is called exploit code, or simply an exploit. The last term to complete the circle is a patch. A patch is a new "fixed" version of a software piece that previously contained a vulnerability, generally provided by the vendor of the software. The deployment of this patch is called patching.

Most software vulnerabilities follow a time-line with the same events. This time-line is essential to be able to understand what the risks are following from a vulner-

ability, and to what extend this can be avoided. Figure 2.1 illustrates this life-cycle including the following events:

- **Creation** - This is the moment a vulnerability is introduced in the software, typically by accident as a result of a coding mistake. With the existence of this flaw the software now contains a vulnerability.
- **Discovery** - A vulnerability gets discovered. The entity discovering the vulnerability has a large influence on the further order of events within this life-cycle. If the vulnerability gets discovered by the vendor itself it can start mitigating the possible effects and it will keep the discovery private. When a hacker discovers a vulnerability things can take a different course. The hacker can inform the vendor, which would then be able to fix things, but an other narrative could be that it sells the exploit online to actors with bad intend. When a vendor is not informed and no public disclosure has taken place, software can be vulnerable for a very long time before action is taken.
- **Exploit availability** - Exploit code becomes available somewhere on the internet, for example on a exploit code database such as the ExploitDB. The publishing of exploit code does not necessarily take place at this point on the timeline, but can occur at any moment after the discovery of the vulnerability. When exploit code is available, this increases the risk for a vulnerability to be exploited in the wild. One of the reasons for this is that now less skilled attackers can also strike using the published code.
- **Disclosure** - The event of public disclosure is a pivotal point in the vulnerability life-cycle. The software vendor can disclose a vulnerability by means of a security advisory, if it is discovered internally. If this is the case normally a patch has already been published before that, so users of the vulnerable software have had the chance to deploy the patch before the vulnerability is made public. In another scenario a hacker could disclose a vulnerability, or it can be disclosed by a vulnerability database such as the NVD. After public disclosure the risk of exploitation is increased dramatically.
- **Patch release** - This is the moment that a patch is released by the vendor of the vulnerable software. As described earlier, if the vulnerability is discovered by the vendor this patch will probably become available at the time of disclosure or before that. If a vendor is surprised by another entity disclosing a vulnerability in their software, the patch release can be delayed since the patch still has to be developed.
- **Patch deployment** - Patch deployment is the moment a vulnerability is mitigated. By installing the patched software, the vulnerability can no longer be exploited.

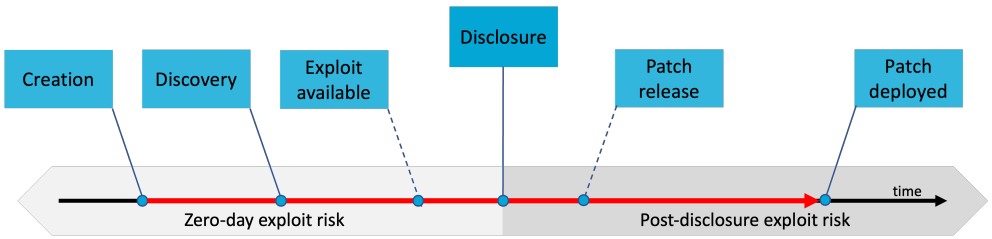


Figure 2.1: Vulnerability lifecycle

The order of events as described in this section is not fully static. Especially the moment an exploit becomes available and the moment a patch gets released can vary. From the moment a vulnerability is discovered until the patch fixing it is

deployed, the user of the software is vulnerable (represented by the red arrow in Figure 2.1). The earlier a user is aware of the vulnerability, the better, but generally the public disclosure is the moment a user is made aware. Before the disclosure of the vulnerability, there is a risk for zero-day exploitation. A zero-day is an exploit that targets a vulnerability that is not yet known. It is very difficult to protect against this, since you don't know what you are looking for. After disclosure, there is a post-disclosure exploit risk. This risk can be addressed, also if there is not yet a patch available. Sometimes there are temporary fixes (workarounds) available, and in the most extreme situation an organisation could stop using the vulnerable software. So, after disclosure an organisation is in some form in control over the situation. That is why the focus of vulnerability management lies in addressing the post-disclosure exploit risk.

2.1.3 Challenges for organisations

The main challenge for organisations is that lots of newly disclosed vulnerabilities come to light on a daily basis, especially when operating a large company infrastructure. Every year increasing numbers of vulnerabilities are disclosed by CVE (and NVD), making it more and more difficult for organisations and security experts to keep up. Especially if you consider that patching is a resource intensive task, involving man-hours of specialized personnel, downtime and the risk of disruption to production systems. The deployment of those resources is more often than not in vain, because only a fraction of all vulnerabilities are ever exploited in the wild [Bozorgi et al., 2010; Jacobs et al., 2020].

The combination of the overwhelming stream of new vulnerabilities, the limited resources companies have available, and the fact that only a fraction of all vulnerabilities ever get exploited, makes that it is critical to apply an efficient patching strategy. To efficiently deploy patching resources it is key to separate those vulnerabilities that pose a serious threat from those that don't. Effective prioritization of patches is challenging and remains an open problem. The next section will cover some techniques.

2.2 PATCH PRIORIZATION

The previous section highlighted the importance of patch prioritization. In this section two methods for patch prioritization will be covered. CVSS is the most widely adopted method which almost all organisations, at least partly, rely on. CVSS has long been criticized for not being very predictive of actual exploitation of a vulnerability, more on this in 2.2.1. An alternative method, gaining ground over the last years, is exploit prediction. This method will be covered in Section 2.2.2.

2.2.1 Common Vulnerability Scoring System (CVSS)

CVSS captures the principle characteristics of a vulnerability and produces a numerical score to reflect its severity. Several metrics are combined to calculate the score, which is ranging from 0 to 10, with 10 being the most severe. The goal of this score is to help organisations properly asses and prioritize vulnerability management processes. The CVSS score is a combination of 3 main components:

- Base Metrics for qualities intrinsic to a vulnerability. These metrics depend on the exploitability of a vulnerability and the potential impact it has.
- Temporal Metrics for characteristics that evolve over the lifetime of vulnerability. The metrics change, as exploits are developed, disclosed and automated and as mitigations and fixes are made available.

- Environmental Metrics for vulnerabilities that depend on a particular implementation or environment. These metrics use the Base and Temporal scores, to assess the severity of a vulnerability considering the context the vulnerable software is deployed in.

The combination of these three metrics is designed to give a score which represents the risk an organisation is facing from a certain vulnerability. The Base score is calculated and published by the NVD but the Temporal and Environmental metrics are left for the user to calculate. The problem is that the calculation of these scores cannot be automated, which leads to organisation ignoring them, and interpreting the Base score as a measure of risk instead. This is not what it is designed for and it fails to represent the risk accurately.

CVSS has long been criticized for being unable to indicate the exploitability of vulnerabilities. Patching prioritization based on the CVSS Base score is often a suboptimal strategy, and in some cases even no better than the random choice [Dey et al., 2015; Beattie et al., 2002; Allodi and Massacci, 2014]. Using only the CVSS Base score, is leading to a very inefficiend allocation of patching resources. An alternative method, gaining ground over the last years is patch prioritization by exploit prediction.

2.2.2 Exploit prediction

Exploit prediction is a response to the failing of current patch prioritization methods such as CVSS. The first exploit prediction system has been developed in 2010 by Bozorgi et al. [2010]. After this, multiple have followed. Different approaches will be analysed in detail in Chapter 3, but the general goal of exploit prediction is to be able to assign a probability to a vulnerability, reflecting the chance it will get exploited. This is done by leveraging data from different sources and a variety of machine learning techniques. Since only a small fraction of all vulnerabilities is exploited, doing this successfully could dramatically reduce patching efforts. However, it is a challenging problem, as will become clear later in this thesis in chapters 3 and 4. For this chapters to be comprehensible the basics of designing a machine learning model and the application to the exploit prediction domain, will be elaborated in Section 2.3.

2.3 EXPLOIT PREDICTION PIPELINE

In the design of exploit prediction systems generally a typical machine learning pipeline is followed. Figure 2.2 illustrates this pipeline which starts with the collection of raw data and yields predictions at the end. These steps in the design and implementation of machine learning models are high level processes that are part of most prediction projects. The exact techniques used in this steps are not set in stone. Machine learning is a dynamic and innovative field, where new techniques are put to the test regularly. Besides the choice of what techniques to use, different methodological assumptions have to be made about how to represent the real world in a model realistically . Unfortunately, "realistic" is not an objective term and thus susceptible to interpretation. In this thesis those assumptions are identified and their effect is evaluated.

Figure 2.2: General machine learning pipeline

By inspecting the steps of this machine learning pipeline, certain general design decisions can be identified, which are the starting point for the analysis of methodological assumptions in exploit prediction systems in Chapter 3. In the data-collection phase for example, important assumptions are made about how to model a problem. What input data is used? Would this information be available in a realistic setting, or only retrospectively? What is considered ground truth? These fundamental questions all impact the final result of the prediction model, but answers can differ when asked to different researchers. The structure of this machine learning pipeline is considered through the other chapters of this thesis.

3 IDENTIFICATION OF CRITICAL ASSUMPTIONS FROM STATE-OF-THE-ART

Along the steps of the machine learning pipeline related works make different design decisions. These decisions are often based on certain assumptions on how to represent the real world in a model. These assumptions affect the reliability, robustness and performance of the proposed systems. In this chapter related works in exploit prediction are reviewed with the goal of identifying these assumptions, and the ones critical for the performance of EPS.

In Section 3.1 the importance of the gathering of realistic data for EPS is outlined. After this attention will be paid to how this data is correctly handled and how EPS can be evaluated, in sections 3.2 and 3.3. Then in Section 3.4 prior works recognizing some of these assumptions are introduced. Finally, a recap of the main findings of this chapter can be found in the chapter summary in Section 3.5.

3.1 COLLECTION OF REALISTIC DATA

The foundation of any machine learning problem is the collection of realistic data of the best possible quality. Input data is used to create predictive features to predict the class-label of a sample. To obtain these class-labels, ground truth data needs to be collected. What all works have in common, is that the CVE-ID as provided by MITRE's CVE system is used to identify vulnerabilities and aggregate data about them.

3.1.1 Input data

Input data for EPS is collected from a wide range of open and closed sources. Almost all works use data, such as CVSS-scores, text description, vendor information, and reference information from NIST's National Vulnerability Database (NVD). Some collect additional data that is collected from other vulnerability databases such as Open Source Vulnerability Database (OSVDB, depreciated), SecurityFocus, IBM X-force Exchange and/or Vulners [Bozorgi et al., 2010; Suciu et al., 2021; Zhang and Li, 2020]. Other sources are leveraged by methods using data from social media such as Twitter [Allodi and Massacci, 2012; Sabottke et al., 2015; Mittal et al., 2016], vulnerability discussions on the dark web [Marin et al., 2016; Allodi et al., 2013; Samtani et al., 2016].

Input sources vary in coverage, timeliness, and quality of data [Miranda et al., 2021; Rodriguez et al., 2018]. While the assumption of what input data to use for EPS certainly affects prediction performance, it falls out of the scope of this work.

3.1.2 Ground truth data

Ground truth data is collected to label vulnerabilities as exploited or not exploited. In other words the ground truth of a vulnerability is the proof that it is exploited or not. This data is hard to come by because exploit events are rare, and information about an exploit is often not shared. Some related works use commercially owned data which is unavailable for the public, making it impossible to reproduce their

work and verify their reported performance [Edkrantz, 2015; Jacobs et al., 2019, 2020]. Others attempting to obtain an adequate ground truth dataset, typically follow one of two paradigms; existence of proof of concept (PoC) exploit code or proof of exploit in the wild. These two paradigms are explained below.

PoC exploit code

Since proof of exploitation in the wild is hard to obtain, a lot of related works in exploit prediction choose availability of PoC exploit code as characteristic to predict. This means that if exploit code is publicly available, the vulnerability is labeled as exploited. In most methods the presence of a CVE-ID in an exploit database such as ExploitDB (`www.exploit-db.com`) or OSVDB (depreciated) is used as ground truth [Alperin et al., 2019; Bhatt et al., 2021; Bozorgi et al., 2010; Bullough et al., 2017; Chen et al., 2019; Edkrantz, 2015; Fang et al., 2020; Reinthal et al., 2018; Sabottke et al., 2015; Yang et al., 2020; Yin et al., 2021]. Others use information from Metasploit (a framework for pentesting) or rumor on security forums, social media, or in security advisories [Tavabi et al., 2018; Fang et al., 2020; Sabottke et al., 2015]. While this form of ground truth data is widely used, it can be misleading because the availability of exploit code somewhere on the internet doesn't necessarily mean that it is also used in the wild. Some exploits require a lot of skills to use or are not feasible for an attacker to exploit [Reinthal et al., 2018].

Exploit in the wild

Another view on ground truth is exploitation in the wild, which means that a vulnerability gets labeled as exploited, when there is proof of the vulnerability being exploited in a real world system. The most pure form of this proof is when a detection rule for an exploit is hit in an Intrusion Detection System (IDS) or firewall running in an organisations network. This data is very scarce and mainly commercially owned. The most well known publicly available dataset for this use is the Symantec dataset (SYM), derived from the Symantec Attack Signature [Symantec, 2018b] and Intrusion Protection signature [Symantec, 2018a] data. This dataset records real-world exploits of vulnerabilities in networks of Symantec customers together with the reported date. It is widely adopted by prediction methods in related work trying to predict exploits in the wild [Almukaynizi et al., 2019, 2017; Chen et al., 2019; Fang et al., 2020; Sabottke et al., 2015; Tavabi et al., 2018]. Other works have a more loose view when it comes to exploits in the wild, and take rumors on security blogs as truth for exploitation [Fang et al., 2020].

Combination of multiple sources

Some works evaluate their prediction on both PoC exploit code as exploit in the wild ground truth [Chen et al., 2019; Fang et al., 2020]. Others gather ground truth data from multiple sources and combine them [Hoque et al., 2021; Sabottke et al., 2015; Suciu et al., 2021].

Implications for EPS performance

Prior work suggests that only about 10 to 15% of all vulnerabilities disclosed, ever have exploit code written for them, and even a smaller amount have functional exploits in exploit kits such as Metasploit [Bozorgi et al., 2010; Hoque et al., 2021]. Even a way smaller amount ever gets exploited in the wild. It is commonly agreed in literature that the percentage of exploits in the wild is under 3% [Allodi and Massacci, 2014; Nayak et al., 2014; Sabottke et al., 2015]. Sabottke et al. [2015] finds, that about 1.4% of all vulnerabilities ever get exploited in the wild. This number corresponds to the findings in this thesis.

This implies that a large part of vulnerabilities which are flagged by EPS as having PoC exploit code available for them, are actually never exploited. EPS using availability of PoC exploit code as ground truth, generally report better performance than works with exploit in the wild as ground truth data. However, it is the question if this advantage holds, considering that not all vulnerabilities with exploit code are actually exploited in the wild, and not all vulnerabilities exploited in the wild, have exploit code available.

So, hopefully, this makes clear that the choice of ground truth has an effect on EPS performance, and might inflate their performance. Considering this, the following critical assumption is formed;

Assumption 1. *Ground truth; are vulnerabilities labeled as exploited when PoC code exists, or when there is proof of exploit in the wild?*

3.1.3 Respecting order of events

The goal of EPS is to say something useful about new vulnerabilities that are being disclosed. In practice this would mean that an organisation is using EPS in real-time, every time a new CVE gets disclosed, using the information available at that time to predict the event of exploitation. The way such a system would operate in a practical setting, needs to be kept in mind when designing an EPS. Because most machine learning models are trained and evaluated on batches of data, the temporal order of events is not naturally respected. Prior works show different approaches when considering the order of events, making it debatable if their reported performance would translate to a real world setting. The most common caveat is the prediction of past data. What is observed in a lot of EPS is the prediction of an exploit that has already occurred. When using an EPS in a real world scenario the events of interest are the ones in the future. There is no real challenge in predicting what has already happened. Most prior works do not filter out events that have happened before disclosure time (which is also prediction time), meaning that the performance they report is probably lifted by the predictions of the past. Yin et al. [2021] clearly shows heavily increased prediction performance in the subset of vulnerabilities with exploit dates before disclosure.

Instinctively one would argue that the prediction of past events would inflate EPS performance. While Yin et al. [2021] and Bullough et al. [2017] clearly show a decrease in performance once past samples are excluded from the data, Reinthal et al. [2018] reports conflicting results. That is why the following assumption is formed;

Assumption 2. *Order of events; is the temporal aspect of the data respected and are vulnerabilities which are already exploited at prediction time excluded or not?*

3.2 PROCESSING OF DATA

After the assumption of what data is used, how this data is handled has great consequences for EPS performance. Three areas of focus are found in literature; feature design, resampling of data, and label-leakage.

3.2.1 Feature design

The correct processing of raw data into usable features is important, but some works fail to do so. An example of that is when the difference between batch learning and prediction in a real world setting is not considered. As discussed earlier, prediction takes place when a vulnerability gets disclosed. EPS are limited by the information which is available at that time. Bozorgi et al. [2010], Hoque et al. [2021] and Reinthal

et al. [2018] design and use features which are only available in a batch setting, and not in a real world setting. They use the difference in publication-date and last-modified-date of a certain CVE in the NVD database as a feature. While this feature could be differentiating in a batch setting, in a real world setting it cannot, since prediction takes place at publication time where publication-date equals last-modified-date. This results in a feature which is the same for each prediction.

Another example of an issue observed in literature is when CVSS is used as a feature. CVSS-scores are calculated and published by the NVD, after thorough analysis of the vulnerability. This analysis takes time, and historically, the NVD did not have enough capacity to timely publish CVSS scores. As identified in Chen et al. [2019], this resulted in a gap between CVE publishing and the publishing of CVSS. The reported size of this gap is 132 days on average (based on data between 2016 and 2018). The problem lies in the fact that typical EPS make their prediction at the moment a new vulnerability gets disclosed, which means CVSS would not yet be available. When batches of historic data are used for training it is possible to use the CVSS-score anyways, while in a real life setting this would be impossible. While this is a flaw, and it will probably influence EPS performance, it is not taken into consideration in this thesis work. The reason for this is that exact data about when CVSS scores are published are not available, making it difficult to simulate a real world situation. Besides, nowadays this gap between disclosure of a vulnerability and publishing of its CVSS is generally way smaller, and the CVSS score is almost always published within a few days.

While the incorrect usage of features is expected to have an effect on EPS performance, it is not widely observed, and difficult to analyse. Because of this no further experimenting will be done on this subject, but it was mentioned for completeness.

3.2.2 Resampling of data

Exploit events are rare, meaning that only a small percentage of vulnerabilities ever get exploited. This holds regardless of the choice of ground truth. This results in an imbalanced dataset; not all classes occur equally often. Machine leaning models generally perform worse on imbalanced datasets. There are different techniques to combat this, such as data-resampling or the use of a cost-function. While these techniques are widely used and accepted, errors in the application can lead to inflated performance of the prediction system. When rebalancing data, equal classes are created by leaving out samples of the majority class, or by synthetically generating extra samples is in minority class. The main rule when rebalancing data is that only training data can be used in the process and the test-data should be left as is. That means that after resampling the training data contains an equal amount of exploited and unexploited samples, while the test set has its original distribution. If this is violated, information about the true class of a sample leaks into the model, which results in overoptimistic performance scores.

It is a general rule of machine learning that the test data can never be touched, except for the actual testing. That being said, a lot of related works manipulate the test data. Bhatt et al. [2021]; Bozorgi et al. [2010]; Edkrantz [2015]; Hoque et al. [2021]; Sabottke et al. [2015] do so by rebalancing the full dataset, before train and test sets are defined, implicitly meaning that the original distribution of the test set is not preserved. The influence of this is expected to be severe. The following assumption is formed;

Assumption 3. *Distribution of test set; is the original distribution of the test set preserved, or is this data manipulated?*

3.2.3 Label leakage

The goal of a prediction system is to predict the class-label of a sample with features derived from data. It is essential that information about these labels do not leak into the features. This is a common pitfall in machine learning and leads to unrealistic performance statistics. There are multiple ways this leakage of information can occur. Two of them, one per ground truth dataset, are explained below;

Label leakage with exploit in the wild data

As mentioned earlier the most common dataset used for the prediction of exploits in the wild is the Symantec dataset (SYM). This data is aggregated by security company Symantec, by combining detection alarms from the networks of their customers. When this data was collected their largest customers where Microsoft and Adobe, which can also be derived from the data; most of the exploits are in Microsoft or Adobe products. Working with this biased data is possible, and even believed to be the best possible way to predict exploitation in the wild, but care has to be taken not to be victim to label leakage.

Because of the bias towards specific vendors such as Microsoft and Adobe, having "vendor name" as a feature would result in information about the label leaking into the features and thus an unfair prediction. Emitting "vendor name" from the features intuitively should solve this problem, and is commonly done. What is often overlooked is that textual descriptions used as features also contain information about the vendor. This can clearly be observed in Tavabi et al. [2018], where the most important features for prediction as exploited are words like "Flash", "Adobe", "Windows" and "Microsoft".

Label leakage with PoC exploit data

In case the availability of PoC exploit code is assumed to be the ground truth, a vulnerability is marked as exploited if code exploiting it is publicly available. The most common source of this information are exploit databases, such as Exploit-DB (`exploit-db.com`) or the database with Metasploit exploits (`rapid7.com/db`). Consider a situation where the existence of a CVE-ID in the Exploit-DB (EDB) is used as the ground truth for a prediction task. A lot of related works use references as one of the features for prediction. These are references listed on the NVD page of a CVE, leading to other domains with additional information about the vulnerability. A problem occurs when these references point to `exploit-db.com`, because the occurrence of this reference in the feature set is inherently the same as having a positive label. While this seems a bit naive, some related works fail to filter out this label leaking references. Also there are more obscured versions of the same problem. One could for example remove references to `exploit-db.com` but not to other databases, say `rapid7.com/db`. In this situation there is no direct label leakage, but because of the large overlap between EDB and Metasploit, a lot of information can be deducted about the probability an exploit exists, which still leads to a unfair prediction.

Implications for EPS performance

If the problem of label leakage is not attended to, performance metrics will be artificially high [Yang et al., 2020]. The impact of this design flaw on EPS performance is expected to be significant. That is why the following critical assumption is formed;

Assumption 4. *Label leakage; are elements in the feature set which leak information about the class-label filtered out, or not?*

3.3 EVALUATION OF MODEL

Model evaluation is of critical importance in any machine learning project. A lot of different techniques exist, which are not all appropriate for every situation. In this section the focus lies on cross-validation and data separation for parameter tuning.

3.3.1 Cross-validation

Cross-validation is used to test how robust a prediction model is and how well it generalises on data it has not seen before. The most basic form is 10-Fold cross-validation, where the full dataset is split in 10 folds. Then the model is trained on 9 of these folds, and evaluated on the last remaining. This is repeated 10 times, every time with a different fold as evaluation fold (see 3.1 for illustration). The scores of all 10 evaluations is averaged, leading to a more realistic score which is less vulnerable to overfitting on the training data. Alperin et al. [2019]; Bhatt et al. [2021]; Bozorgi et al. [2010]; Edkrantz [2015]; Fang et al. [2020]; Hoque et al. [2021]; Jacobs et al. [2019, 2020]; Sabottke et al. [2015]; Tavabi et al. [2018]; Yang et al. [2020] all use this a cross-validation form similar to this, while it is not so suitable for EPS. The reason for this is that exploit prediction data is temporal, while this temporality is not respected by this form of cross-validation.

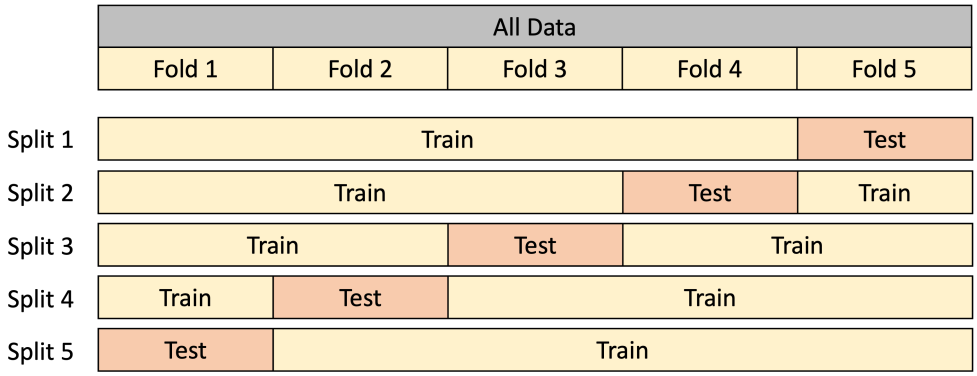


Figure 3.1: K-Fold cross-validation

New vulnerabilities are disclosed every day, meaning that the information available for EPS also changes over time. In a real life setting all historic data is available for prediction, and the predictions you would be interested in, lay in the future. When using k-Fold cross-validation to evaluate an EPS, this property is not respected. In most splits, test data precedes (part of the) train data. This essentially means that data from the future can be used to make predictions in the past. Besides this being counter intuitive, it also yields exaggerated results [Bullough et al., 2017].

Temporal cross-validation is form of cross-validation which adheres to the temporal nature of exploit data. In this form test data always precedes train data. Different techniques can be leveraged to accomplish this [Almukaynizi et al., 2017, 2019; Bullough et al., 2017; Chen et al., 2019; Reinthal et al., 2018; Suciu et al., 2021; Yin et al., 2021; Zhang and Li, 2020].

Different views on cross-validation are observed in literature, and the implications for EPS performance are tremendous. That is why the following critical assumption is formed;

Assumption 5. *Cross-validation; is the temporal nature of exploit data respected in cross-validation or not?*

3.3.2 Data separation for parameter tuning

Most classification algorithms have parameters which can be tuned, to optimize performance on the data at hand. The optimal set of parameters is obtained by simply trying different settings and observing how the model performs. This practice is known as a gridsearch; searching all possible settings for optimal performance. This is a common method in machine learning. Problems occur when the same data is used for parameter tuning as is used for evaluation of the final model. When this is done, the parameters are optimized to yield the best performance on a specific evaluation dataset, and chances are this model does not do so well on new unseen data. This practice is in contradiction with the machine learning rule that test data should only be touched when actual testing is done.

Prior works often don't report clearly on this matter, but Hoque et al. [2021]; Jacobs et al. [2020]; Sabottke et al. [2015]; Suciu et al. [2021]; Tavabi et al. [2018] seem not to respect this rule, while Bozorgi et al. [2010]; Edkrantz [2015]; Fang et al. [2020]; Reinthal et al. [2018] do. This design choice is expected to have an effect on EPS performance, that is why the following assumption is formed;

Assumption 6. *Parameter-tuning; is separate data used for parameter-tuning and final evaluation?*

3.4 RELATED WORKS RESEARCHING ASSUMPTIONS

This thesis work is not the first to acknowledge that certain design decisions greatly impact performance of EPS. Bullough et al. [2017] was the first work that experimented with this. They conduct experiments on the effect of rebalancing data, (temporal) cross-validation and the order of events, similar to assumptions 3, 5, 2 respectively. Reinthal et al. [2018] also experiments with (temporal) cross-validation and the order of events. Where in some experiments similar results are reported, there are some contradictions, for example in the order of events experiment. This thesis work is different because more assumptions are identified and analysed, and additional ground truth data is evaluated. Besides, it tries to reproduce parts of Bullough et al. [2017] and Reinthal et al. [2018], to investigate contradicting results.

3.5 CHAPTER SUMMARY

Related works in the exploit prediction domain follow different approaches, and adhere to different assumptions while making decisions in the EPS design process. In this chapter an extensive analysis of nineteen related works has been conducted, focusing on these assumptions. A detailed overview of the analysis can be found in table form in Appendix A. In short, three important distinctive phases in the EPS pipeline are identified: the collection of realistic data, the processing of data, and the evaluation of the model. Within these phases certain assumptions about how to correctly model an EPS and its context are made. Some of these assumptions are unrealistic or faulty, leading to models that fail to represent a real world situation. In the three phases as described earlier, the following six critical assumptions have been identified.

- Collection of realistic data
    1. **Ground truth** (are vulnerabilities labeled as exploited when PoC code exists, or when there is proof of exploit in the wild?)
    2. **Order of events** (is the temporal aspect of the data respected and are vulnerabilities which are already exploited at prediction time excluded or not?)

- Processing of data
    3. **Distribution of test-set** (is the original distribution of the test set preserved, or is this data manipulated?)
    4. **Label leakage** (are elements in the feature set which leak information about the class-label filtered out, or not?
- Model evaluation
    5. **Cross-validation** (is the temporal nature of exploit data respected in cross-validation or not?)
    6. **Parameter-tuning** (is separate data used for parameter-tuning and final evaluation?)

The goal of this chapter was to analyse the methods of related works in the exploit prediction domain, to identify which methodological assumptions are adhered to. This has resulted in an extensive related works analysis (Appendix A) and in six critical methodological assumptions as listed above. Knowing the effects of these assumptions on EPS performance is important to place reported results of prior work into context. In this thesis, experiments will be conducted to quantify these effects. Results will be compared to Bullough et al. [2017] and Reinthal et al. [2018], who have also experimented with some of these assumptions.

In the next chapter, Chapter 4, the methods that will be used for this experiments, and for the optimization steps after it will be elaborated. Results can be found in Chapter 5.

4 METHOD

In this chapter the method of this research will be elaborated. Section 4.1 describes how the assumptions as obtained from literature are evaluated and compared. Basics as data collection and the design of features and baseline models are explained. Next, in Section 4.2, the method for optimization of EPS is outlined. Finally, in Section 4.3, the information of this chapter is summarized.

4.1 EVALUATION OF ASSUMPTIONS

In this section the used data is described, and how it is collected. After this, baseline models are constructed and the experimental design is presented. Lastly, the performance metrics used to evaluation are explained and justified.

4.1.1 Data collection and feature design

The goal of this analysis is to evaluate the effects of different assumptions in the prediction system design phase. Earlier works such as Bullough et al. [2017] and Reinthal et al. [2018] tested some of these assumptions. It is attempted to replicate some of their work, also to obtain more clarity about conflicting results. Both works have different time-frames of analysis; 2009-2015 and 2015-2018 respectively. In the data collection phase of this thesis, it is attempted to obtain all data from different sources that is available. A selection is made later what data to use, for comparison to other works. First, input data is collected from NVD. Ground truth data is collected from Symantec and Exploit-DB. Information from different sources is aggregated based on its CVE-ID.

Input data

Input data is obtained from NVD. This is a database maintained by NIST and has the most complete collection of known vulnerabilities. All vulnerabilities have a CVE-ID as identifier. This identifier is broadly adopted in the security community and functions as common ID across different sources. All data from the NVD is extracted amounting to 163,602 samples in total ranging from 1988 till 2022. Figure 4.1 shows the distributions of published vulnerabilities between 2000 and 2022 (the few vulnerabilities from 1988 till 2000 are excluded, to prevent a very long-tailed histogram).

Each sample consists of different information fields including CVSS information, vendor information (Common Platform Enumeration), vulnerability type (Common Weakness Enumeration), textual description of the vulnerability, and reference information. Some data preprocessing is needed to make these information fields accessible and useful for machine learning algorithms. Numerical features such as CVSS Bases score and reference count are scaled from 0 to 1. Categorical features are encoded using One-Hot-encoding, and textual features are encoded using a Natural Language Processing method called Text-Frequency Inverse-Document-Frequency (TF-IDF) vectorization. TF-IDF is a numerical statistic that is intended to reflect the importance of a certain word in a corpus of words. Feature-space tends to get very big when TF-IDF is used, because a new feature is created for each word

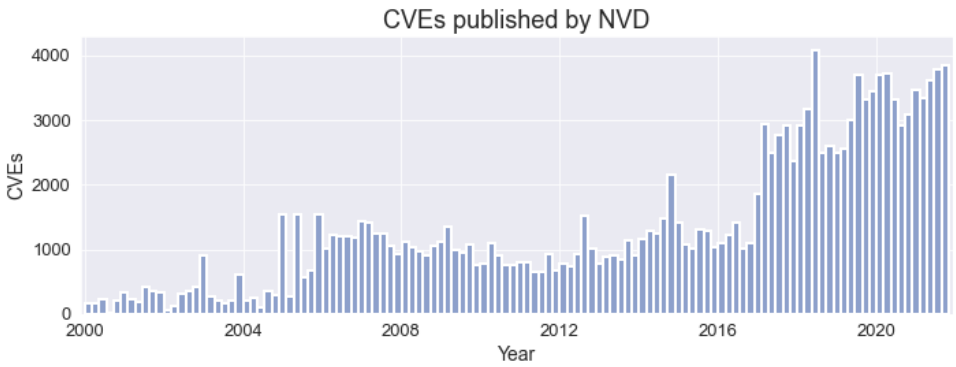


Figure 4.1: Amount of vulnerabilities published by NVD

occurring in the corpus. To limit feature space, only the 1000 most important words are considered. An overview of all features used, and from what source they origin can be found in Table 4.1.

Ground truth data

Ground truth data is collected from two sources; Symantec for exploit in the wild data, and Exploit-DB for PoC exploit code data. Figure 4.2 shows the amount of exploited vulnerabilities from both sources between 2009 and 2018.

Symantec is a security company which is now owned by another company called Broadcom. Unfortunately Broadcom does not make the same exploit data available as Symantec did. Therefore extra effort has been done to collect the same data from symantec as other works have reported. This is accomplished by using a website archiving tool called the WaybackMachine [1] . The tool enables to search and scrape previously archived websites. Different historic copies of the Symantec websites, between 2014 an 2018 are have been scraped. Samples from both Symantec Attack Signature [Symantec, 2018b] and Intrusion Protection signature [Symantec, 2018a] pages have been collected. The IPS signatures do not have temporal information available making them unusable, as it cannot be confirmed that these exploits take place after disclosure of the vulnerability. Most samples from the IDS attack signatures have the date of discovery available. Since the archiving tool did not archive all the needed pages, vulnerabilities are gathered from different Symantec webpages such as an overview page, separate writeups and an archived RSS feed. For some Attack Signatures the discovery date was not available. For these the publishing date is used, which was available. Manual inspection of samples with both the discovery date and publishing date available, showed that these dates lie very close to each other. This justifies using the publishing dates for samples without a discovery date. In total 645 samples with date have been collected, ranging from 2014 to 2018.

Exploit-DB is one of the main public collections of PoC exploit code. Although the completeness of EDB is on decline, it is considered the most complete set of exploits. To use exploits on EDB as ground truth, samples from EDB need to be mapped to a CVE-ID. MITRE (maintainer of CVE database) publishes an official mapping from CVE-ID to EDB-ID [MITRE, 2022a]. This source contains mappings to about 12000 CVE-IDs, but is not complete. An additional mapping is obtained from Fang et al. [2020] which, combined with MITRE's mapping, leads to a set of 23478 unique CVEs which have exploit code available on EDB. The publishing date of the exploit is obtained by scraping the EDB website.

1 https://www.archive.org

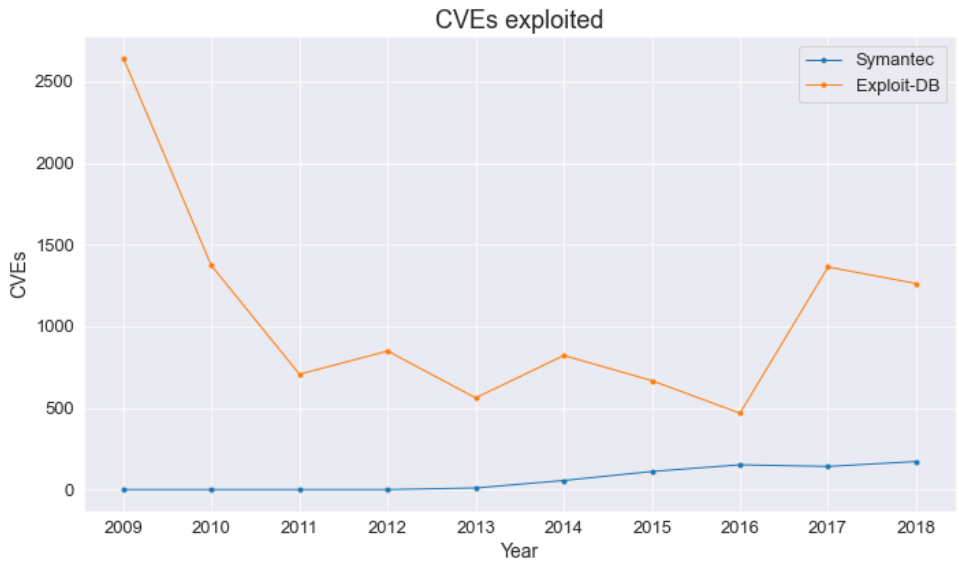


Figure 4.2: Amount of vulnerabilities exploited per source

Once the exploit date of a vulnerability is gathered, either from Symantec or from Exploit-DB, class labels are generated in the following fashion; a sample gets labeled as positive, if the exploit date is within 6 months of the publishing date. A certain cut-off time is essential, because otherwise it would never be possible to assign a negative label to a sample. After all, exploitation could still happen even a long time after NVD publishing. Different cut-off times have been tested, and a cut-of time 6 months have proven to be ideal. When a 12 month window is asserted, performance does not change much in comparison with 6 months. The shorter window of 6 months, however, gives a more precise indication of when a vulnerability is expected to be exploited and thus of the urgency of remediation.

4.1.2 Baseline models

To test the effect of the methodological assumptions on EPS performance 3 baseline models have been created. Two of these models are reproductions of the baseline models of Bullough et al. [2017], Reinthal et al. [2018], to enable comparison to their works. The third baseline model is created for experiments which have not yet been conducted by prior works. All models use the same input data from the NVD and the features as described in Table 4.1. Table 4.2 contains an overview of the settings of the 3 baseline models.

Baseline 1 (BUL)

The period of analysis of their work is from 2009 through 2016. Input data is collected from the NVD, and the ground truth is based on exploit availability in EDB. They attempt to prevent label leakage by removing references to "Exploit-DB", "Milworm" and "OSVDB". Zero-day exploits are not removed from the training data. A Support Vector Machine (SVM) with a linear kernel is used as prediction algorithm, which is validated using K-Fold cross-validation.

Baseline 2 (REIN)

The period of analysis is from 2015 through 2018. The feature-set consists of data collected from the NVD and additional "web chatter". The latter source is not available, so in this work only NVD data will be used. They use availability of exploit code

Source	Name	Type	Processing
CVSS (NVD)	Access vector	Categorical	OHE
	Access Complexity	Categorical	OHE
	Authentication	Categorical	OHE
	Confidentiality impact	Categorical	OHE
	Integrety impact	Categorical	OHE
	Availability impact	Categorical	OHE
	Base score	Numerical	N(0,1)
CPE (NVD)	Vendor name	Text	TF-IDF
	Product count	Numerical	N(0,1)
CWE (NVD)	Vulnerability type	Categorical	OHE
	CWE count	Numerical	N(0,1)
References (NVD)	Reference domain	Text	TF-IDF
	Reference count	Numerical	N(0,1)
Summary (NVD)	Description text	Text	TF-IDF
	Description length	Numerical	N(0,1)
Labels (Symantec)	Exploited	Boolean	True/False
	Disclosure date	Date	Datetime
Labels (EDB)	Exploited	Boolean	True/False
	Disclosure date	Date	Datetime

Table 4.1: Features per data source

in the EDB as ground truth. NVD references are also used as a feature, however, no filtering of references to exploit databases is conducted, possibly leading to label leakage as described in 3.2.3. Zero-day exploits are removed from the training-data and the performance is temporally evaluated with a separation date of 08-08-2017. This means that all samples before this date are in the training set, and all samples after this date in the test set. They have chosen to use XGBoost as prediction algorithm.

Baseline 3 (OWN)

The third baseline model will be based on data collected between 2014 and 2018. Features will be created from NVD data and the availability of an exploit in EDB is chosen as ground truth. Label leakage will be prevented by filtering out references containing "Exploit-DB", "Milworm" and "OSVDB". Zero-day exploits will not be removed from the training data. Three classification algorithms are selected; Logistic Regression Classifier (LR), Linear Support Vector Classifier (SVC) and XGBoost (XGB). Multiple algorithms are used in the experiments to assure that observed behaviour is generic, and not just the specific behaviour of a certain classifier in that scenario. LR, LinearSVC and XGB belong to very different "families" of classification algorithms, and rely on different mathematical foundations. This choice is made to solidify the outcome of the experiments, and to be able to make statements about machine learning models in general. Besides these three algorithms are used by a lot of related works, and SVC and XGB specifically in Bullough et al. [2017] and Reinthal et al. [2018]. Temporal cross-validation is used to evaluate the models. The temporal cross-validation is implemented by using a rolling window approach. The rolling window technique dynamically defines training and testing datasets from the data available. The data is ordered by date and an initial model is trained on batch data of the first 12 months. With a trained model, performance metrics are calculated on the following 6 months of data. After this the testset of 6 months is

added to the training data and the model is retrained (see 4.3 for illustration). This goes on until the end of the data is reached.

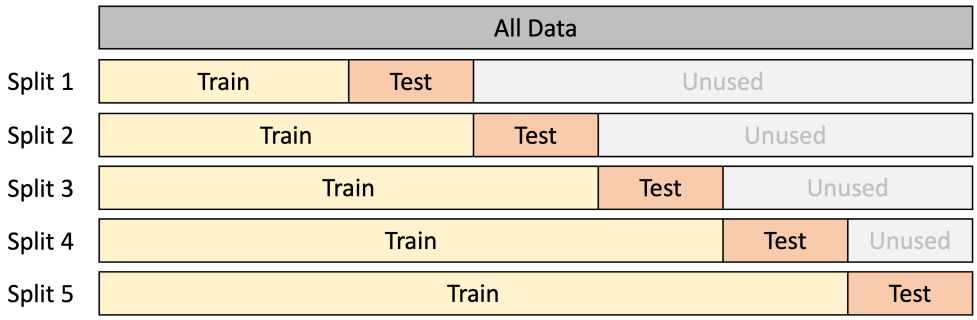


Figure 4.3: Temporal cross-validation (rolling window)

In the design of this baseline model all choices are made to simulate an environment that is closest to a real world situation, which reflects in the methodological assumptions. The only exception is the choice of ground truth. The decision has been made to use EDB data as ground truth, because the use of different ground truth would make comparison difficult and less intuitive. Evaluation of SYM ground truth data will be done at a later stage.

The Python package *sklearn* is used for implementation of all classification algorithms. Instead of hard classification predictions, prediction probabilities are used to make it possible to create Precision-Recall curves (more on evaluation metrics in Section 4.1.4).

Setting	**BUL**	**REIN**	**OWN**
Period	2009-2015	2015-2018	2014-2018
Input	NVD	NVD	NVD
Ground truth	EDB	EDB	EDB
Zero-days	Not excluded	Excluded	Excluded
Label leakage	Filtered	Not filtered	Filtered
Validation	K-Fold	Temporal single date	Temporal rolling window
Classifier(s)	SVC	XGBoost	LR, XGBoost, SVC

Table 4.2: Design settings baseline models

4.1.3 Experimental design

For each of the 6 identified assumptions as described in Chapter 3 an experiment has been set up. Table 4.3 gives an overview of all experiments. Per assumption 1 experiment will be conducted. An X is present in the columns *BUL* and/or *REIN* if the experiment is conducted by prior work an will be re-evaluated. The baseline model will be chosen as outlined in Section 4.1.2. Per experiment the experimental variable (*Experiment* column) will be changed, and all other settings will remain as is in the baseline model. By doing so, it is attempted to isolate the effect of this single variable on EPS performance as good as possible.

#	Assumption	Experiment	BUL	REIN	OWN
1	Ground truth	EDB or SYM			X
2	Order of events	Remove zero-days or not	X	X	X
3	Distribution of testset	Resample testset or not	X		X
4	Label leakage	Prevent label leakage or not			X
5	Cross-validation	Temporal or intermixed	X	X	X
6	Parameter tuning	Using only training set or not			X

Table 4.3: Experimental design

Experimental Prediction Framework

To facilitate the extensive experiments in both the evaluation of the methodological assumptions and in the optimization steps, an exploit prediction framework has been designed and implemented. The goals of this framework was to automate the collection and processing of publicly available data, to simplify experimentation with different machine learning techniques in the field of exploit prediction. Flexibility for future additions and adjustment has been one of the main priorities. Another requirement was that all of the parameters, from the choice of (ground truth) data to the selected ML techniques, can be supplied in a *json* configuration file, enabling queueable and efficient experimenting. An example of this file, with an explanation of the parameters, can be viewed in Appendix B. The framework consists of 3 modules that handle different subtasks; the Data Collection Engine, the Experimentation Engine and the Visualisation Engine.

The Data Collection Engine collects input data from NVD, the IBM Threat Exchange, and the CVE database. Ground truth data is collected from EDB and Symantec. Data is collected through different webscrapers and API scripts, aggregated and matched on CVE-ID, and is stored efficiently in a SQL-database. Information in the database can easily be queried and preprocessed according to the desirable format. The Experimentation Engine is the backbone of the exploit prediction framework, and relies on the the structure of *sklearn* Pipelines. This module enables experimenting with different (pre-)processing steps, rebalancing techniques, machine learning algorithms, optimization approaches, and cross-validation methods. All of the experimental parameters are supplied in the configuration file. Lastly, the Visualisation Engine aggregates detailed results and creates insightful illustrations. Each evaluated candidate can be inspected and analysed.

Currently most prior works are closely guarding their advancement by shielding the used data and methods. As mentioned earlier, the source-code of this project will be open-sourced. This is important, because it enables collective and sustained progress in the field of exploit prediction. The full Exploit Prediction Framework will be pulished on Github [2].

2 https://github.com/EPFramework/Exploit-Prediction-Framework

4.1.4 Performance metrics

Using the right performance metrics to evaluate EPS is important to prevent skewed results. To illustrate this, consider the performance metric accuracy. Accuracy is defined as the number of correctly predicted samples, divided by the total number of predictions. While this is the right metric for some situations, in others it is not. In a situation with heavy class-imbalance, such as in an exploit prediction environment, it does not make sense to use accuracy as a metric. Suppose only about 2.5% of all vulnerabilities are exploited. A classifier which would classify all samples as non-exploited, would have a accuracy of 97.5%. This seems like a good score, but the classifier did not predict any exploited vulnerability. To prevent these problems, in this thesis Precision, Recall, and F1-score are used, in combination with a Precision-Recall plot. To understand these metrics first the basic classification results need to be clear. Table 4.4 is a confusion matrix, showing the 4 different possibilities for a prediction result. There are two kinds of correct predictions and two kinds of incorrect predictions. Correct predictions happen when a sample is predicted as exploited and it's true label says it is exploited, or when it is predicted as not exploited, and it's true label says it is not exploited. These are called True Positives (TP) and True Negatives (TN) respectively. Incorrect predictions occur in two situations; when a sample is predicted as exploited, while it actually is not, or when a sample is predicted as not exploited but it actually is. These are called False Positives (FP) and False Negatives (FN). A perfect classifier would not have False Positives and False Negatives.

	Predicted: Exploited	Predicted: Not Exploited
True: Exploited	**T**rue **P**ositive	**F**alse **N**egative
True: Not Exploited	**F**alse **P**ositive	**T**rue **N**egative

Table 4.4: Confusion metrics

Precision

Precision is defined as the amount of True Positives divided by the sum of the True Positives and False Positives (equation 4.3). In terms of exploit prediction the precision score reflects the fraction of all vulnerabilities predicted as exploited that are correctly classified as exploitable. Precision can also be seen as the efficiency of an EPS; of all vulnerabilities marked as exploited, how much are correctly flagged as such, and how much are not. The latter, False Positives, cause organisations to put effort in the remediation of vulnerabilities which probably never get exploited, thus leading to an inefficient patching strategy.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

Recall

Recall is defined as the amount of True Positives divided by the sum of True Positives and False Negatives (equation 4.2). For EPS this score represents the fraction of all exploited vulnerabilities which are marked as such by the EPS. The recall score can be seen as a measure of coverage; of all exploitable vulnerabilities, how many did the EPS find? Coverage is important for organisations because a low coverage

means that there is a large blind spot of vulnerabilities which can be exploited but are not picked up by the system, thus leading to a larger (invisible) risk for the organisation.

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

F1

The F1 score is the harmonic mean of the precision score and the recall score, and can be calculated with equation 4.3. As outlined above, both precision as recall are important for measuring EPS performance. Unfortunately, there is a trade-off between the two. A higher precision means a lower recall and vice versa. An EPS with a high recall score is useless if precision is lacking, and the other way around. F1 gives a clear score in a situation that precision and recall are equally important.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.3}$$

Precision-Recall curve

As mentioned earlier both recall and precision are important metrics for EPS, and there is a trade-off between the two. The F1-score represents the harmonic mean of those two scores. However, situations can occur where one of those scores is more important than the other. Consider for example a situation where an organisation has a very small security team responsible for remediating vulnerabilities. It is then most important that an EPS leads to improved efficiency (precision), because all wasted effort it too much. Another example is a situation where an organisation is not willing to take risk and has the resources to remediate most vulnerabilities. In this situation coverage (recall) is most important, because an EPS missing a lot of exploitable vulnerabilities results in a larger risk and in (extensive) resources being idle.

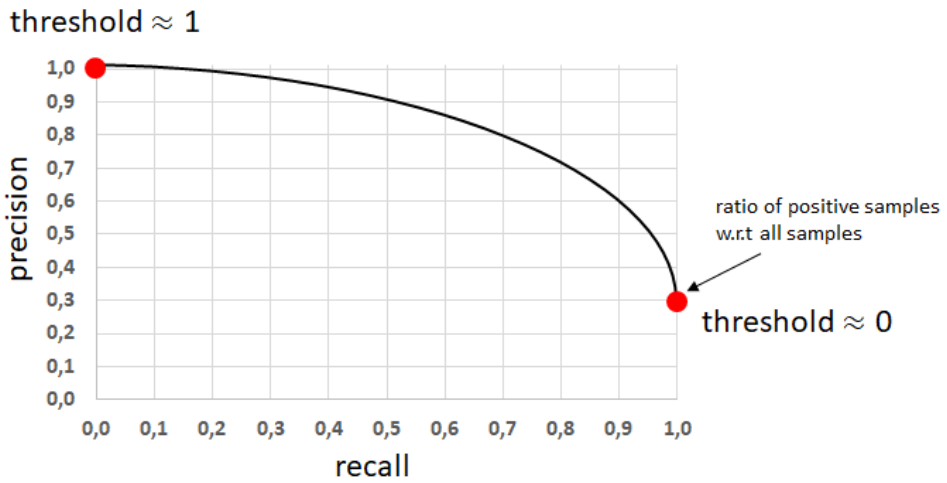


Figure 4.4: Example of Precision-Recall curve

By adjusting the classification threshold of a classifier (the minimal probability to classify a vulnerability as exploited) one can influence this precision-recall trade-off. Increasing the threshold, leading to a less sensitive classifier, results in EPS with a higher precision and a lower recall. The other way around, when lowering the classification threshold, the classifier gets less sensitive and classifies more vulnerabilities as exploited. This leads to higher recall, at the price of precision. To capture

the dynamics of this behaviour, a Precision-Recall curve is plotted. Figure 4.4 is an image of such a plot in the most basic form. As can be seen in this PR-curve, a threshold of ±1 (all vulnerabilities classified as not-exploited) leads to a recall near 0 and a precision near 1 (top left dot). Conversely, a threshold of ±0 (all vulnerabilities classified as exploited) leads to a recall near 1, because all truly exploited vulnerabilities are obviously identified, and a precision which is equal to the ratio of positive samples with respect to all samples. For all classification threshold between 0 and 1, the precision and recall is calculated and plotted on this curve. An ideal classifier would have a PR-curve which would reach all the way to the top right corner of the figure.

4.2 OPTIMIZATION UNDER REALISTIC CIRCUMSTANCES

As leads from Chapter 3, different assumptions in the design stage of an EPS have an impact on its performance. Some choices in this process are more realistic than others. In this section the most realistic assumptions for an EPS will be discussed in Section 4.2.1. After this, promising paths are explored to optimize EPS performance under these restricted but realistic assumptions. This is done by leveraging different techniques of handling class-imbalance (Section 4.2.2), by designing more advanced features (Section 4.2.3), and by exploring and tuning different machine learning algorithms (Section 4.2.4). These optimization steps will be evaluated sequentially. This means that the results of the first optimization step will be incorporated in the second, and so on. In this section the theory behind all evaluated techniques is elaborated. Results of the actual analysis can be found in Chapter 5.

4.2.1 Restricted realistic environment

To get an idea of how EPS would perform once operating in a real world situation, it is attempted to create an environment that most closely resembles such a situation. This is done by complying to the most strict methodological assumptions, as identified in Chapter 3. So to recap, this entails the following situation. Exploit in the wild ground truth data is used instead of Proof-of-Concept ground truth. The order of events is strictly respected by removing all zero-days from the data, to prevent "prediction" of the past. The class-distribution of the test set data is not tempered with. Features are filtered so that no illegitimate leaking of the class-labels can occur. And lastly, temporal intermixing is prevented by applying cross-validation that respects the temporal aspect of the data.

Three models (Logistic Regression, LinearSVC and XGBoost) are trained in an environment as described above. These 3 models function as the baseline models for this optimization step. All optimized models will be compared against the baseline models, to get a view of what effect the optimization techniques have.

4.2.2 Imbalance handling

The first step in this optimization section, is the step of handling class-imbalance. Imbalanced data generally causes problems for machine learning models. To illustrate this, consider the following example. Suppose, that there is an imbalanced dataset with a minority class that consists of 10% of all samples. This means that the other samples (90%) are of the majority class. A classifier might learn to classify all samples as belonging to the majority class, reaching an accuracy of 90%. An accuracy score of 90% seems quite decent, while such a classifier fails to correctly predict any of the samples in the minority class. Since the goal of exploit prediction is to identify the minority samples (exploited vulnerabilities), the handling of class-imbalance is extra important. In this section the theory behind different techniques

to mitigate the negative effect of class-imbalance are explained. First, different data resampling techniques are covered (oversampling, undersampling, and a combination). Then, algorithmic utilization of class-weights to minimize the negative effects of class-imbalance is explained. The performance of these methods is evaluated in Chapter 5.

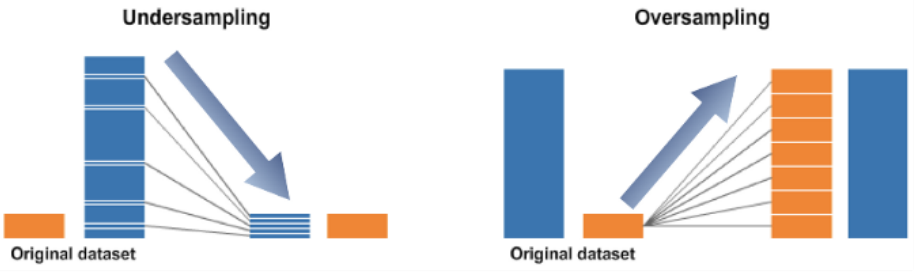


Figure 4.5: Visualisation of undersampling and oversampling

Oversampling

The first technique to handle class-imbalance is oversampling. The goal of oversampling is to obtain an equal class-distribution by artificially creating new samples similar to samples of the minority class (see Figure 4.5). To build further on the earlier example, suppose a dataset has 100 data samples with a distribution of 90 negative samples and 10 positive ones. Oversampling techniques would generate 80 extra positive samples, to obtain equal classes of both 90 samples. These new data samples can be obtained in various ways. A few of the evaluated techniques are SMOTE, BorderlineSMOTE, SVMSMOTE and ADASYN.

A widely adopted resampling technique is Synthetic Minority Oversampling Technique (SMOTE), developed by Chawla et al. [2002]. SMOTE is a technique that generates new samples which are of similar to existing samples of the minority class. It does so by by following a few predefined steps. To illustrate these steps a very simple 2 dimensional example is considered. The first step is to draw a random sample from the minority class, and identify its k nearest neighbors (see 4.6b). When this is done one of the nearest neighbors is randomly selected. A new sample is generated which lies along the line between the drawn sample and the selected nearest neighbor (see 4.6c). The position on this line is also chosen randomly. This procedure is repeated for all samples in the minority class, until the desired class-distribution is reached.

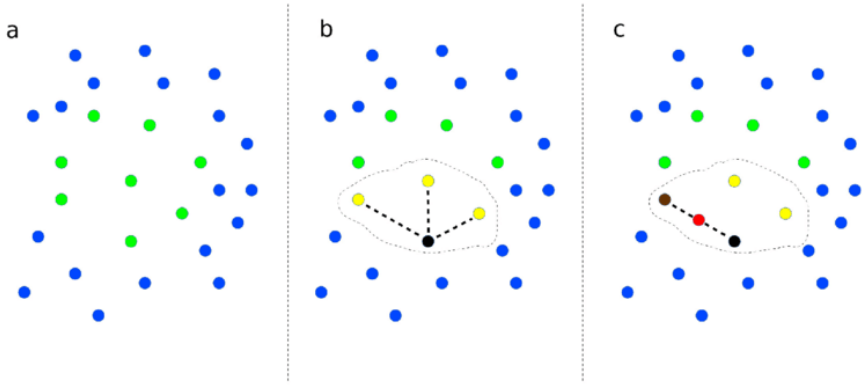


Figure 4.6: Graphical demonstration of SMOTE oversampling

SMOTE works quite well in situations like the considered example, where the classes show little to no overlap. Problems might occur when there is a lot of overlap between the different classes, or if there are a lot of outliers. Several adaptations of the SMOTE algorithm, such as BorderlineSMOTE and SVMSMOTE, have been developed to tackle these shortcomings.The BorderlineSMOTE strategy has a more advanced selection of minority samples, before the generation of new data is started. The algorithm identifies outliers by checking the nearest neighbors of a drawn minority sample. If all of these neighbors are of the majority class, the drawn point is considered an outlier and is not used for the creation of new samples. Also samples along the class-border are identified (points with both minority and majority nearest neighbors), and more synthetic samples are generated near these borderline points. SVMSMOTE, is a technique that builds further on the principles of BorderlineSMOTE. The difference is that SVMSMOTE uses a SVM instead of KNN algorithm to estimate the class-border. SVMSMOTE generally generates more data further away from the region of class-overlap, enabling easier classification. The last oversampling technique considered in this thesis is Adaptive Synthetic Sampling (ADASYN). This technique is very similar to the other approaches described earlier. ADASYN has a different way of choosing where in the feature space new samples are generated. BorderlineSMOTE and SVMSMOTE pay special attention to minority samples near the class-border and to outliers, while ADASYN adaptively decides which areas should be accentuated. It generates more new samples in areas where there is a low density of minority samples, and less where the density is already higher.

Undersampling

Another resampling technique is undersampling, where samples from the majority class are removed to obtain a more equal class-distribution (see Figure 4.5). There are several undersampling techniques considered in this thesis, such as RandomUndersampling, TomekLinks, EditedNearestNeighbors and OneSidedSelection. The simplest of all is RandomUndersampling, where random samples from the majority class are removed until the desired class-distribution is achieved. The other techniques have other criteria for selecting which samples to remove.

The TomekLinks approach selects majority samples to remove by identifying so called Tomek links. Two samples, A and B, share a Tomek link if A is B's nearest neighbor and B is A nearest neighbor, *and* A and B belong to a different class. These Tomek links can be used to identify noisy samples that lie close to each other and might be difficult to distinct for classification models. When this technique is used for undersampling, the majority sample of the Tomek link is removed. The EditedNearesNeighbors (ENN) technique removes samples by finding the k nearest neighbors of all samples in the data. For each data sample its class is compared with the most prevalent class of its nearest neighbors. If the most prevalent class is different than the class of the selected sample, the sample and all its nearest neighbors are removed from the data. This procedure is repeated until the desired class-distribution is reached. OneSidedSelection combines TomekLinks with another approach called Condensed Nearest Neighbors (CNN). TomekLinks is used to remove noisy and borderline samples, and CNN focuses on removing redundant samples from the majority class.

Over-undersampling

A drawback of the application of undersampling in highly imbalanced environments, is that a lot of the training data is lost in the process, Figure 4.5 illustrates this. In exploit prediction datasets often as little as 2% of the samples is marked as exploited. Undersampling would result in a dataset with these positive samples,

and the same amount of negative samples, meaning that 96% of the data remains unused. By applying a combination of oversampling and undersampling techniques, this can be prevented. This is done by first using an oversampling technique to reach a certain desired class-distribution, and after this, undersampling is used to remove samples of the majority class to obtain a balanced dataset. To continue with the example above, consider a dataset with 100 samples of which 2 are positive and 98 are negative. With only undersampling 4 of the 100 original samples will be used. Now suppose that first a form of oversampling is applied to reach a 30/70% positive/negative class-distribution. This would mean that 40 synthetic samples would be generated (so 2+40 positive samples, and 98 negative samples). By applying undersampling, a balanced dataset will be obtained consisting of 42 positive and 42 negative samples. With this approach 2 positive and 42 negative samples of the original dataset are used, in stead of 2 positives and 2 negatives when only applying undersampling. In the implementation of the optimization step all combinations of undersampling and oversampling are considered.

Class weight of algorithm

Most machine learning algorithms have the option to provide the *class_weight* parameter. When this parameter is specified, mis-classifications of positive and negative samples are weighted differently. This can serve as an approach to mitigate the bias of classification algorithms due to class imbalance. This process is most intuitively explained by means of a simple example. Consider an imbalanced dataset with a minority class that consists of 10% of all samples. This means that the other samples, 90 percent, are of the majority class. A classifier might learn to classify all samples as belonging to the majority class, since this would yield a decent accuracy of 90%. Consider now a situation where a classifier is provided with class weights, so that mis-classification of a sample of the minority class is 9 times as "costly" as a mis-classification of a sample of the majority class. In this situation the cost of mis-classifying all minority samples (10% x 9 = 90) is equally expensive as mis-classification of all majority samples (90% x 1 = 90). In theory, by applying these different class weights, a classification algorithm no longer favours classification to the majority class.

4.2.3 Feature design

As outlined in Section 4.1.1 input data is collected from the NVD. In this optimization step it is attempted to improve prediction results by creating richer features from the information available. Another option would obviously be to collect more information from different sources. In this thesis the choice is made to focus on making more efficient use of the data available from the NVD, instead of the collection of additional data. The main reason for this is that the collection of new data might introduce issues concerning the temporal nature of the data. As described in Section 3.2.1 some related works design features based on data which would not be available at the time of prediction in a realistic situation. When collecting historic data retrospectively, it is very difficult to assess when certain data is published. Guaranteeing that is was already available at the time of prediction, is often impossible. With data from the NVD this risk is minimized, since it reports about a vulnerability as soon as it gets assigned a CVE-ID. The information the NVD reports is often not the most extensive, but it is guaranteed to be available at the time of prediction.

The data collected from NVD consists of numerical, categorical and textual data (see feature Table 4.1). The features obtained from numerical and categorical data are pretty straight forward; raw information is used directly with minimal normalization and encoding. Improvements of these features are unlikely. That is why the focus of this optimization step lies on improving features constructed from textual

data. The art of presenting textual data in a machine readable form is called Natural Language Processing (NLP). There exist loads of different techniques with the purpose of effectively extracting meaningful features from text. Building further on the previous optimization step, two fundamentally different approaches are considered in this thesis work. The first approach is Text Frequency-Inverse Document Frequency (TF-IDF, as used in the baseline models of Section 4.1), and the second is text featurization by leveraging word embeddings.

Text Frequency - Inverse Document Frequency (TF-IDF)

TF-IDF is a technique to statistically describe words in a sentence, making textual data more suitable for interpretation by machines. In the most simple words this technique keeps a count of the occurrences of words in a set of sentences (Text Frequency), and normalizes this count by diminishing the weight of very frequent words and increasing the weight of terms that rarely occur (Inverse Document Frequency). This technique yields one vector per sentence in which every word of a sentence is represented as a decimal number. In the case of exploit prediction the sentence is the textual description of a vulnerability as obtained from the NVD. To calculate the TF-IDF value for a sentence, the first step is to calculate the Text frequency. Text frequency is nothing other than the number of occurrences of a specific word in a sentence. The TF operation can be be expressed mathematically following equation 4.4.

$$TF(w,s) = \frac{\text{\# of occurences word in sentence}}{\text{\# of words in sentence}} \tag{4.4}$$

Relying only on the TF of a word in a sentence, would result in high values for words that are prevalent in a lot of sentences. This is unwanted because words that occur in a lot of sentences (stopwords) are generally not distinctive, and thus of little significance. This issue can be overcome by introducing the Inverse Document Frequency in the calculation. The IDF value of a word can be calculated by taking the logarithm of the total number of sentences over the number of sentences containing the specific word (equation 4.5). This results in lower values for words that are occuring in a lot of sentences.

$$IDF(w) = \log\left(\frac{\text{\# of sentences}}{\text{\# sentences containing word}}\right) \tag{4.5}$$

Finally, the TF-IDF value of a certain word in a sentence is calculated by multiplying the TF and the IDF values (equation 4.6). By calculating this value for each word in a sentence, a sentence can be represented as a vector of decimal numbers which can be processed by machine learning models. TF-IDF creates a single feature for all unique words that occur in any of the sentences, which can lead to an enormous feature space. To contain this, often the maximum number of features is defined. By doing so only the most important words are kept as a feature.

$$TFIDF(w,s) = TF(w,s) * IDF(w) \tag{4.6}$$

Word embeddings

Representing textual data with word embeddings is the second approach considered in this thesis. This process is more complex than the calculation of a TF-IDF vector. One of the main advantages of using word embedding over using TF-IDF is that embeddings also capture semantic properties of words. This means for example that similar words like "cat" and "tiger" also have similar embedding vectors, which can be valuable for the interpretation of words and sentences. The details about the working and the construction of word embedding models is out of scope

for this thesis, but the general structure of several models will be covered is this section. The following techniques will be considered: Word2Vec, GloVe, and fastText.

Word2Vec is a NLP technique that uses a neural network to learn word associations from a large corpus of training text. Each word is represented as a decimal number, so a sentence consists of a vector of decimal numbers. The value of Word2Vec is that similar words also have similar embeddings. The Word2Vec algorithm can be used to train a custom model, but it is also possible to utilize pre-trained models which are publicly available. Both of these options are explored in this optimization step. The pre-trained model that is used, is a model trained by Google based on more than 3 million unique words.

GloVe stands for Global Vectors and is a unsupervised learning algorithm for obtaining vector representations for words, developed by researchers at Stanford University [Pennington et al., 2014]. This algorithm generates word embeddings by aggregating global word co-occurrences matrices for a given corpus. A co-occurence matrix shows how often a certain word pair occurs together in a sentence. The goal is to derive the relation between words from these statistics. For the GloVe model there are also pre-trained models available. The first one used in this thesis is trained on Twitter and contains 1,2 million unique words, and the second is trained on Wikipedia data and contains 400k words.

fastText is a library for the learning of word embeddings created by Facebook's AI Research lab [Bojanowski et al., 2017]. It offers both supervised as unsupervised algorithms to obtain vector representations of words. The working of fastText is quite different from the first two word embedding algorithms. Word2Vec and GloVe consider words as smallest unit to train on, while fastText uses n-gram character strings as smallest unit. This means for example that a word like "apple" will be represented as "ap","app", "ppl" and "ple" before an embedding is created. The advantage of fastText over other algorithms is that it performs well on words that are rare, or even not present in the training data. Both pre-trained as custom trained models are considered. The pretrained model is trained on Wikipedia news and contains 1 million words.

4.2.4 Algorithm optimization

In this optimization step the focus lies on finding the optimal machine learning algorithm, and tuning it so that it performs as good as it potentially can. This is done by hyper-parameter tuning, and by testing a lot of different machine learning algorithms.

Hyper-parameter tuning

Generally machine learning algorithms have a set of hyper-parameters, which are parameters that in some form control the learning process, and can not be interfered with during the fitting of the model to the training set. These parameters generally influence the speed and/or the quality of the learning process. These parameters can for example influence the learning rate, the amount of regularization or the structure and topology of the model. An other example of a hyper-parameter is the class weight parameter we saw earlier in the imbalance handling section (Section 4.2.2).

It is difficult to find the optimal set of hyper-parameters. The most common approach in finding those parameters is by exhaustively evaluating every different set of parameters. This can effectively be done by *GridSearchCV* which is a Python class developed as part of the *sklearn* library. This class enables you to specify different classifiers and different parameter ranges, and returns the detailed results when it is done. This seems easy, but the limitation lies in the computational effort it requires. Suppose for example that we have parameter x, with possible values

{1,2,3}. To find the optimal setting for this parameter, just 3 models need to be trained. Now if another parameter y is introduced, with possible values {10,20,30}, the problem scales exponentially. Now 3x3 = 9 models have to be trained. With only two parameters with small value ranges, this is no issue. But when a dozen of parameters need to be tuned and possible value ranges are extensive, this becomes infeasible very quickly. In this thesis it is attempted to find a balance in this trade-off between computational effort and reaching the optimal performance of a model.

Prediction algorithms

The baseline algorithms used in the experiments of the methodological assumptions are Logistic Regression, LinearSVC and XGBoost. Besides these 3 algorithms there exist tons of others in the field of machine learning. The following have been selected for further evaluation; AdaBoost Classifier (with both DT and LogReg as base), Decision Tree, K-Nearest-Neighbors Classifier, RandomForest Classifier, SGD-Classifier, (regular) SVC and a Neural Network Classifier. Also, a Voting Classifier is used to test the effect of assembling different algorithms. A voting classifier trains different algorithms, that separately make their predictions. To reach the final prediction a majority vote is casted. The hyper-parameters of all algorithms have been tuned roughly to limit the computation time. Algorithms yielding promising results are then subjected to a more thorough grid search.

4.3 CHAPTER SUMMARY

This chapter consisted of two parts. In the first part the method and experimental design is outlined in order to evaluate the methodological assumptions as identified in Chapter 3. The second part contains optimization strategies, to improve EPS performance under restricting but realistic assumptions.

To evaluate the effect of different methodological assumptions, first a few baseline models are constructed. The BUL and REIN models are created to enable comparison to Bullough et al. [2017] and Reinthal et al. [2018] respectively. It is attempted to reproduce the models of their works as accurately as possible. Then, the OWN baseline model is constructed, which is constructed following the most realistic methodological assumptions. In different experiments these baseline models will be evaluated, following the experimental design as described in Table 4.3.

In the second part of this chapter promising optimization strategies are described. The goal of those strategies is to boost EPS performance under realistic circumstances. This is done by exploring different techniques of handling class-imbalance, by designing more advanced features, and by exploring and tuning different machine learning algorithms.

The methods for the experimental evaluation of assumptions and the optimization steps, are implemented in the Exploit Prediction Framework. This framework enables easy and automated experimentation with different machine learning techniques, and handles all steps from data collection to exploit prediction. This system requires a configuration file in *json* format, in which the experiment or optimization parameters should be defined. The working of this file, and the framework, is described in Appendix B. The results of the experiments and optimization steps, can be found in Chapter 5.

5 RESULTS

In this chapter the results of this thesis will be presented. For the results, the same two part structure as in the previous chapter is followed. In the first part the influence of six critical methodological assumptions in EPS design will be evaluated (Section 5.1). Section 5.2, the second part, is an attempt to optimize a EPS using more restricting, but realistic assumptions. Additionally, in Section 5.3, these results will be interpreted from an organisational perspective.

5.1 ASSUMPTION EXPERIMENTS

This section contains the results of the conducted experiments as outlined in the experimental design (Table 4.3). Per assumption one experiment will be conducted and evaluated against at least one baseline model, depending on if similar experiments have been conducted in prior work. Experiments 2 and 5, can be compared to Reinthal et al. [2018] and thus will be evaluated using the REIN baseline model. Experiments 2, 3, 5 which are compared to Bullough et al. [2017] will be evaluated using the BUL baseline model. Experiments which are not previously conducted will be evaluated on data from 2014 until 2018, using the OWN baseline model. The results of the experiments will be presented in the form of a Precision-Recall plot. Full experimental results can be found in Appendix C. In all PR-curves, the same classifier has the same color line. So for example a green solid line and a green dotted line, represent a classifier in the baseline and experimental environment respectively.

5.1.1 Reproduction of baseline models

To be able to compare the results of this experiments to prior work, the baseline models of Reinthal et al. [2018] and Bullough et al. [2017] are reproduced in this section. It is important to confirm that similar results are observed with the baseline models, otherwise later comparison might not make sense.

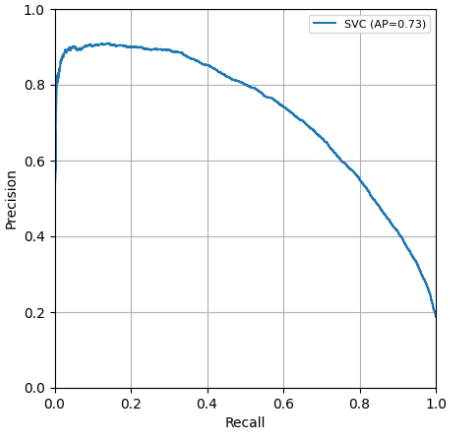
Bullough et al. [2017] (BUL)

The period of analysis of their work is from 2009 through 2016. All features are created from NVD data, and the ground truth is based on exploit availability in EDB. They try to prevent label leakage by removing references to "Exploit-DB", "Milworm" and "OSVDB". Zero-day exploits are not removed from the training data. A Support Vector Machine (SVM) with a linear kernel is used as prediction algorithm, which is validated using K-Fold cross-validation.

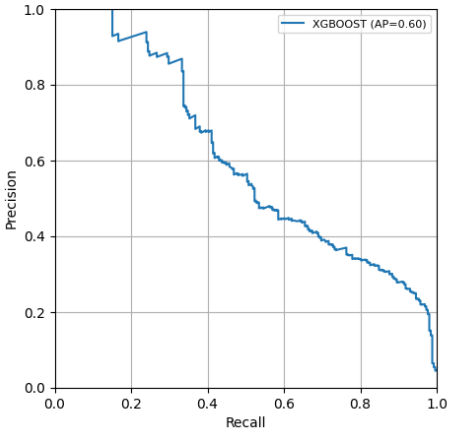
Reinthal et al. [2018] (REIN)

The period of analysis is from 2015 through 2018. The feature-set consists of data collected from the NVD and additional "web chatter". The latter source is not available, so in this work only NVD data will be used. They use availability of a CVE-ID on the EDB as ground truth. NVD references is also used as a feature, however, no

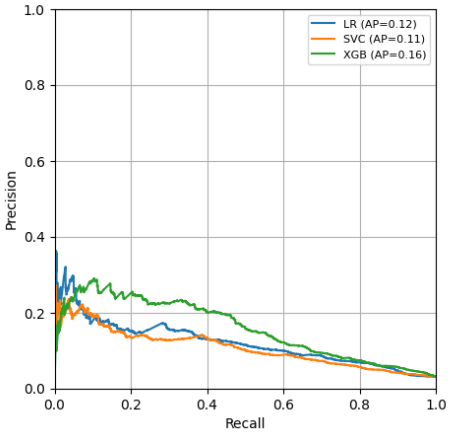
filtering of references leading to exploit databases is conducted. Zero-day exploits are removed from the training-data and the performance is temporally evaluated with a separation date of 08-08-2017. This means that all samples before this date are in the training set, and all samples after this date in the test set. They have chosen to use XGBoost as prediction algorithm.



(a) LinearSVC in BUL setting



(b) XGBoost in REIN setting



(c) 3 classifiers in OWN setting

Figure 5.1: Baseline models of related works

Figure 5.1 contains Precision-Recall curves for all 3 baseline models. The obtained results for BUL and REIN are very similar to what is observed in the original works. The reported F1-score of the baseline model in BUL is 0.647, while it was 0.641 in this reproduction. For REIN the difference is a bit bigger. Their reported F1-score is 0.407, while 0.485 is found in this reproduction. Even though slightly different features are used, the performance of the reproduced models is very close to that of the original work. The OWN baseline model performs worse than the other two models, with an average F1-score between 0.002 and 0.310. This is the result of more strict assumptions.

5.1.2 Experiment 1: Assumption of Ground Truth

In this experiment the choice of ground truth is evaluated. This is not yet done in prior work, so comparison to the OWN baseline model will be conducted. The

baseline model is trained with ground truth data from Exploit-EB (EDB), and the experimental models are trained using exploit-in-the-wild ground truth from Symantec (SYM). Because the baseline and experimental models are based on other data, it is difficult to compare performance one-to-one. What this experiments shows, however, is that based on the same input data and features, the choice of ground truth affects the performance of an exploit prediction system. These two choices of ground truth are widely adopted as outlined in Chapter 3, making it interesting to discover what the effects of this choice are.

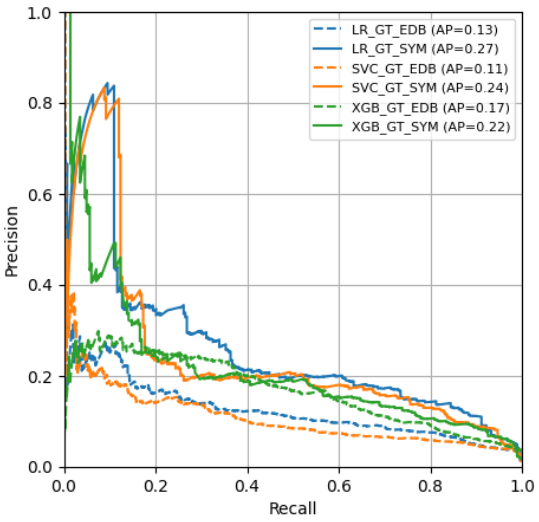
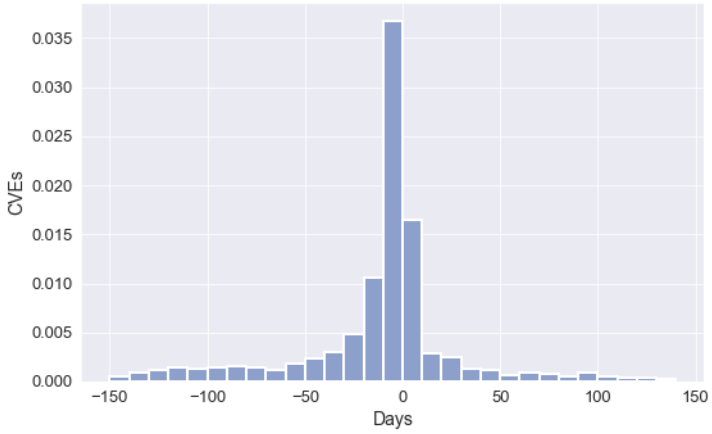


Figure 5.2: Experiment 1: Baseline versus SYM ground truth (OWN baseline)

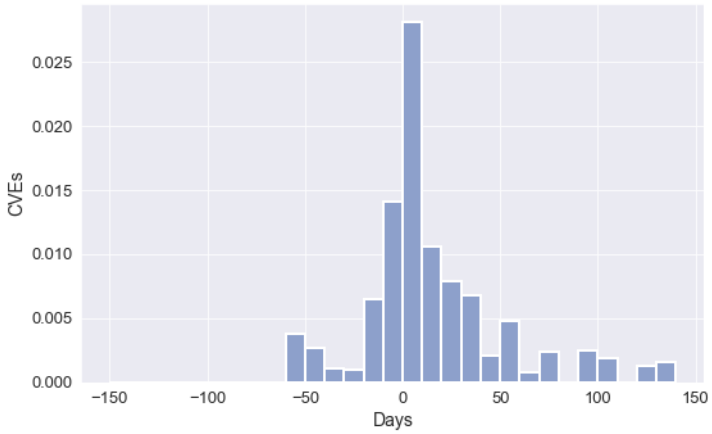
As can be seen in Figure 5.2 performance of the experimental models is slightly higher than that of the baseline models with EDB ground truth. This is unexpected behaviour, because most related works using EDB ground truth report much better results. This is normally attributed to SYM having limited positive samples, and thus being very imbalanced. The EDB dataset is bigger and less imbalanced. When zero-days vulnerabilities are excluded though, as in the OWN baseline setting, this difference in imbalance gets very small (±1% for SYM and ±2% for EDB). Figures 5.3a and 5.3b show the time difference in days between NVD publication, and the reported exploit date for SYM and EDB ground truth respectively. This figures show that the proportion of zero-day vulnerabilities in the EDB ground truth data is way higher than in the SYM data. In experiment 2 this behaviour will be evaluated further.

5.1.3 Experiment 2: Assumption of Order of events

In this experiment the effect of zero-day vulnerabilities present in the training data will be evaluated. Zero-day vulnerabilities are defined as vulnerabilities which already have PoC exploit code available, before they are disclosed by the NVD. In this case the prediction if exploit code for a vulnerability will be developed, does not make sense, because it already happened. The amount of vulnerabilities which appear on EDB before they are published on NVD is quite extensive. Figure 5.3a shows a histogram of the difference in days between NVD publishing and the exploitation date (date published on EDB). The amount of zero-days is vast; in the BUL setting from 2009 until 2016, 6,282 of all 7,625 vulnerabilities that have exploit code available on EDB, were published before being disclosed on NVD. This is almost 83% of all vulnerabilities. For the REIN setting in period 2015-2018, this number is 53%.



(a) EDB exploit date



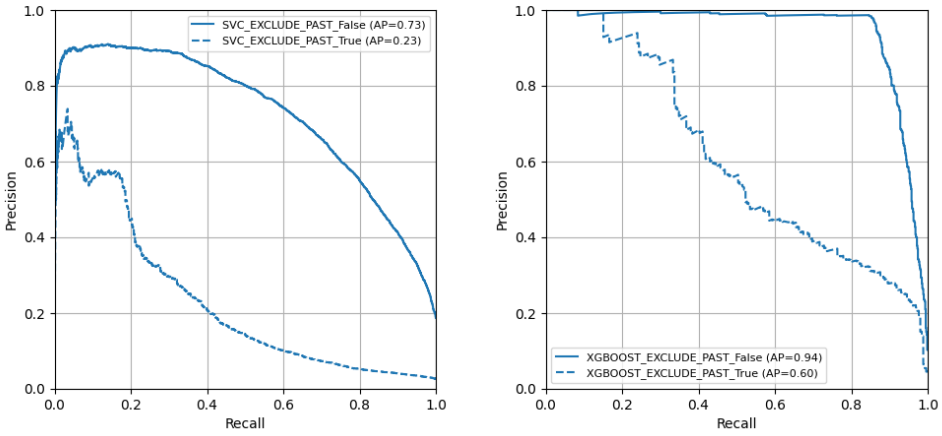
(b) SYM exploit date

Figure 5.3: Difference in days between NVD publishing date and exploit date

The effect of excluding zero-days will be tested by removing these vulnerabilities from the training-set and compare it to a situation where these vulnerabilities are not filtered out. The result of this experiment can be observed in Figure 5.4.

The influence of removing zero-day exploits from the training data is extensive. F1-scores drop 62% and 42% respectively for BUL and REIN. In both baseline settings, more than half of the positive samples is removed. This also results in a shift in class-distribution of the dataset. BUL went from 18.7% to 2.5% positive samples and REIN from 8,6% to 3.5%. Possibly, this interferes with classifier performance. To test this effect the experiment has been conducted, but now with resampling of the training data to create a balanced dataset. The effect of applying SMOTE oversampling was minimal, and performance even dropped a bit (full results in Appendix C). This shows the effect of imbalance is minimal and the performance drop is likely attributable to the removal of zero-day exploits.

The severe drop in performance is in line with the results Bullough et al. [2017] report. The results for the period from 2015-2018, the REIN setting, are in conflict with what they report. They report an increase of performance when zero-day exploits filtered out. This is the opposite behaviour observed in this work and in other works. While the baseline model of Reinthal et al. [2018] is successfully reproduced, this behaviour is not easily explained. Upon more thorough investigation, it seems that the data used in Reinthal et al. [2018] is incomplete. Table 2 in their report gives an overview of the collected data. They report to have collected 24,944



(a) BUL setting

(b) REIN setting

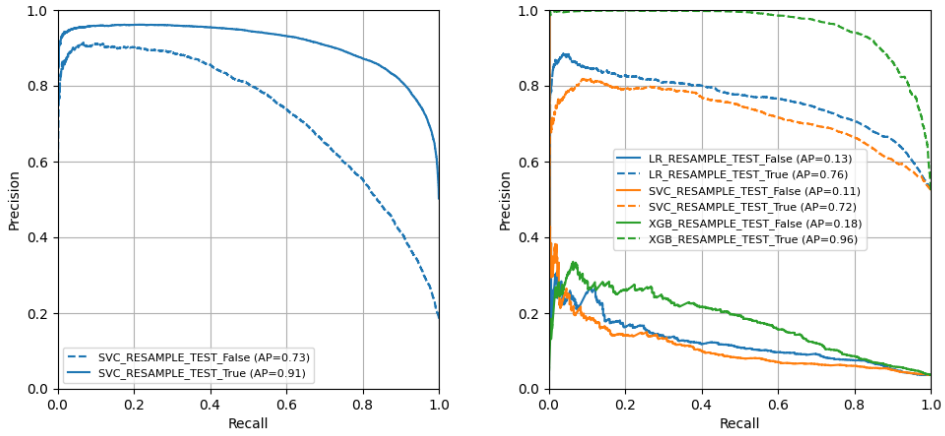
Figure 5.4: Experiment 2: Exclusion of zero-day exploits

samples from NVD of which 809 are exploited, whereas in this work 28,889 NVD samples have been collected in the same timeframe, of which 2,511 are exploited. The cause of this is unknown, but it is reasonable to assume that this difference in data would affect EPS performance. As a benchmark the collected data in this work is compared to the data of BUL (2009-2015). The difference of collected data was very small (NVD 38,129 versus 39,430, EDB 6,470 versus 7,380). Also Yin et al. [2021] reports having found 23,413 vulnerabilities with exploit code in EDB, from 1990 until 2020 which is the same amount as what is found in this work.

5.1.4 Experiment 3: Assumption of Distribution of test-set

In this experiment the effect of incorrect data-sampling to mitigate the effects of class imbalance is evaluated. Resampling is done to create an artificial dataset in which different classes are roughly equally distributed. This is done to optimize classifier training, and often yields improved results. One way to do this is by generating synthetic samples from the minority class (oversampling), by using SMOTE. Using oversampling is legit and widely used in machine learning. The only condition is that SMOTE must be applied to the training data only, leaving the test data with it's original class-distribution. What is observed in a lot of prior works, is that the full dataset is resampled, before training and testing data is defined. This way, the distribution of the testset is also changed. As can be seen in Figure 5.5 EPS performance gets a big boost when the distribution of the testset is being artificially altered. While this should ring all alarm bells, this kind of behaviour is observed in multiple prior works (see Chapter 3). Figure 5.5a shows the result of the experiment compared using the BUL baseline settings. The F1-score observed in this experiment is inflated by 21% by resampling the testset, which is comparable to results found in Bullough et al. [2017]. Figure 5.5b shows the experiment in the OWN baseline setting. This extra experiment has been conducted to visualise how enormous this effect is on EPS performance, and that the behaviour generalises over different classification algorithms.

The altering of the testset is wrong, because information about the labels leak into the model, and unrealistic, because in a real world setting the labels that are attempted to be predicted are not know at that time.



(a) BUL setting **(b)** OWN setting

Figure 5.5: Experiment 3: SMOTE oversampling of test set

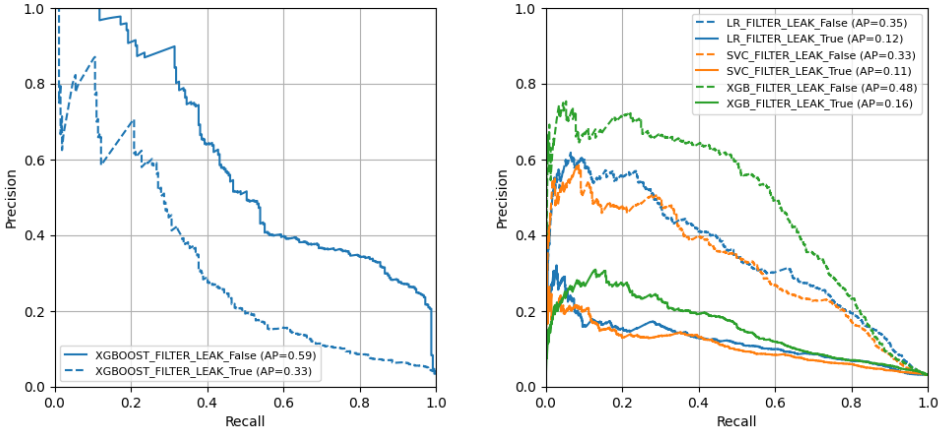
5.1.5 Experiment 4: Assumption of Label Leakage

This experiment tests the effect of label leakage. This experiment is not yet conducted in previous work and will be evaluated in the OWN baseline setting. Besides, the REIN baseline is evaluated, because in their work they fail to avoid label leakage. In machine learning the number one rule is that the testset should never be available in any form when training a classifier. When this does occur a foul and unrealistic situation arises yielding overoptimistic classification results. The EDB ground truth dataset is prone to a form of label leakage where references of a vulnerability on NVD contain information which should not be available for training. Since the EDB dataset is constructed using the availability of PoC exploit code on the Exploit-DB website, references to this website should not be available in the feature set. NVD references often contain a url either to the EDB website or to the website of another exploit database. The existence of such a url implicitly reveals the class label. Every vulnerability with a link to EDB is labeled as positive, no exceptions, meaning that a machine learning classifier might learn this pattern leading to overoptimistic classification results. To test this, the baseline models are compared to a model where references to 4 exploit databases are filtered out (`www.exploit-db.com`, `www.rapid7.com`, `www.osvdb.org`, `www.milw0rm.com`). The results of this experiment can be seen in Figure 5.6.

Full results can be found in Appendix C. When filtering the references for unintended references to EDB, the F1-score decreases 30% on average over the 3 tested classifiers in the OWN baseline configuration, and about 10% for the REIN configuration. All other configurations are unchanged, meaning that the isolated effect of label leakage is extensive.

5.1.6 Experiment 5: Assumption of Cross-validation

In experiment 5 the effect of different cross-validation methods is evaluated. The main distinction between different methods observed in prior work, is if the temporal nature of the data is respected. The most basic form of cross-validation is k-Fold cross-validation. When applying k-Fold validation, the available data is divided in k parts (folds). In multiple rounds, different train-test splits are evaluated. So say k=3, then the data is divided in 3, and in the first split fold 1 and 2 are training data and fold 3 is test data. In the next split the test data is changed to fold 2 and the rest is training data, and so on. The scores of all splits are averaged to obtain the



(a) REIN setting **(b)** OWN setting

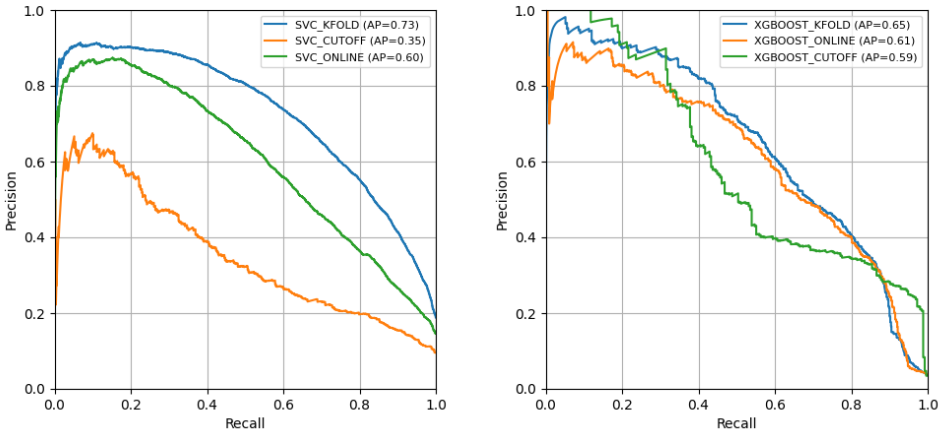
Figure 5.6: Experiment 4: Filtering of label leaking references

final score. One of the issues of this approach is that in most train-test splits, future data is used for training, which would never be available in a real world setting. The intermixing of temporal data can be result in inflated performance metrics of EPS. To overcome this, some form of temporal cross-validation has to be applied. The most straightforward technique is by splitting the data in using some cutoff date. All samples before this cutoff date are in the training set, and all the other samples are in the test set. While this solves the problem of temporal intermixing, only a single split and score is available. One of the goals of cross-validation is to produce more robust metrics, by testing the model on different splits of data. This goal is defeated by this form of cross-validation. Lastly, there is another form of temporal cross-validation which is called online learning or rolling window validation. When using this technique the data should be sorted chronologically. The first split consists of a initial batch of data (in this work 12 months), on which a model is trained. This model is then evaluated using a test-set that consists of the next 6 months. Once the performance metrics of this split have been calculated, the test-set is added to the training data. A new model is then trained on this new training-set, and is evaluated on the following 6 months. This goes on until the end of the available data. All data except for the initial training set, is in the test set at some point. The scores of all different splits are averaged to obtain the final score. This approach mitigates the problem of temporal intermixing of data, while still producing robust results over multiple different splits.

Figure 5.7 shows the result of this experiment. There is a clear pattern which holds for both the BUL and the REIN baseline setting. Performance of regular k-Fold cross-validation, with temporal intermixing, yields the best performance. Second and third come the temporal evaluation methods. Worst performing is the temporal method where a single cutoff date is chosen to split training and test set. Somewhere between those two extremes the temporal cross-validation method of online learning is observed. This order holds for both baseline evaluation settings. The difference between methods is greater in the BUL setting than in REIN.

5.1.7 Experiment 6: Assumption of Parameter Tuning

This experiment shows the effect of improper separation of data uses during the tuning of hyper parameters. Parameter tuning is the art of finding the right parameters for a classification algorithm, with the goal of optimizing its performance. Tuning these parameters enables a classifier to optimize its working on the data at



(a) BUL setting

(b) REIN setting

Figure 5.7: Experiment 5: Evaluation of cross-validation methods

hand. More on this topic will be explained later in the optimization part of this chapter.

Searching for the optimal parameters is often done in a brute force manner where all possible combinations of parameters are tested (called a grid-search). The parameter set yielding the best performance is selected for further usage in the model. When the classifier performance during parameter tuning is evaluated on the same data that will later be used for the final scoring, the final parameters depend on the labels of this data. In a real situation it would not be possible to optimize classifier performance on this data.

To test this effect a nested gridsearch is performed. The difference between a nested gridsearch and a regular gridsearch is that the final evaluation data is always fresh and unseen by the model, even during parameter tuning. This is accomplished by conducting a gridsearch on the training data only. The parameters this search yields, are then used to train a model on the full training set, to evaluate performance on the test-set.

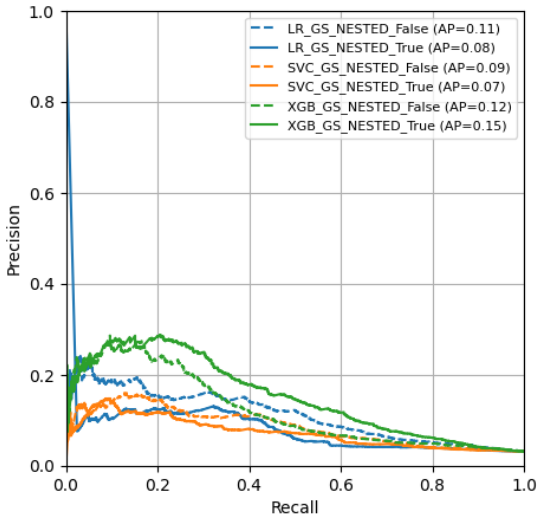


Figure 5.8: Experiment 6: Parameter tuning with regular gridsearch compared to a nested gridsearch

Figure 5.8 shows the results of this experiment where parameter optimization with a gridsearch and a nested-gridsearch are compared. As can be seen the effects are minimal but there is a slight difference. All classifiers except the XGBoost classifier show a small decrease in performance. With an additional analysis with a bigger parameter space, also the XGBoost performed worse with a nested gridsearch. While this result is apparent, it will not be taken into account in the optimization phase of this chapter. The reason for this is that the process of conducting a nested gridsearch is extremely time consuming, even more than a standard gridsearch. With the performance difference being neglectable, ignoring this factor is legitimized.

5.1.8 Summary of Assumption Experiments

This section provided the experimental results for the methodological assumption experiments as outlined in Section 4.1. In total six experiments have been conducted, following from the six methodological assumptions. In each experiment the most realistic design decision is compared to a less restrictive alternative observed in literature. These design decisions following from methodological assumptions, are evaluated using three baseline models. The BUL and REIN models are reproductions of the EPS from Bullough et al. [2017] and Reinthal et al. [2018]. These reproductions yield results very similar to the original works. The OWN baseline model is introduced in this thesis, testing multiple classification algorithms using assumptions which are considered the most realistic. The outcomes of each experiment will be briefly discussed.

1. **Ground truth** - In this experiment ground truth based on availability of PoC exploit code (EDB) is compared to ground truth based on exploitation in the wild (SYM). This experiment is not yet conducted in prior work, and therfore is evaluated with the OWN baseline model. The results are unexpected; SYM ground truth returns better results than EDB ground truth, while prior works using EDB ground truth generally report much higher results. This could be attributable to most of these works failing to adhere to the order of events, which results in a much more balanced EDB dataset.

2. **Order of events** - This experiment compares a situation where all vulnerabilities which are exploited before the disclosure (zero-days) are excluded, versus a situation where they are not excluded. BUL and REIN baselines will be used, because both those works conduct this experiment as well. Both BUL and REIN show a dramatic drop in performance when zero-day exploits are excluded. This is in line with what Bullough et al. [2017] reports, but contradicts Reinthal et al. [2018]. The latter reported an increase in performance when zero-day exploits are excluded. Possibly this is attributable to the fact that they use a smaller (incomplete) dataset.

3. **Distribution of test-set** - In this experiment the effect of tempering with the distribution of the test set is evaluated. This is done by comparing a situation where the complete data set is resampled to a situation where only the training data is resampled. Resampling the full data set is methodologically wrong. The performance of EPS when resampling the full data set are near perfect for both BUL and OWN baseline models.

4. **Label leakage** - In this experiment the effect of label leakage is evaluated. Failing to prevent label leakage in both the REIN as the OWN baseline models, leads to heavily inflated performance. This experiment is not conducted in Reinthal et al. [2018] but they do not prevent label leakage, making it interesting to compare against the REIN baseline model.

5. **Cross-validation** - The fifth experiment tests the influence of different cross-validation methods. The two compared methods are K-fold cross-validation and temporal cross-validation using a rolling window approach. The difference between the two is that the latter respects the temporal aspect of the data. The BUL and REIN baseline models have been used for evaluation. They both show decreased performance when temporal cross-validation is used, which is in accordance what Bullough et al. [2017] and Reinthal et al. [2018] reported.

6. **Parameter-tuning** - The last experiment tests the influence of parameter-tuning on the same data as the evaluation data, by comparing a regular gridsearch with a nested gridsearch. This experiment is not conducted in prior work. While it was expected that a regular gridsearch, which uses the same data as the evaluation data, would produce inflated performance metrics, this was not the case. Both situations yielded near identical results on the OWN baseline model.

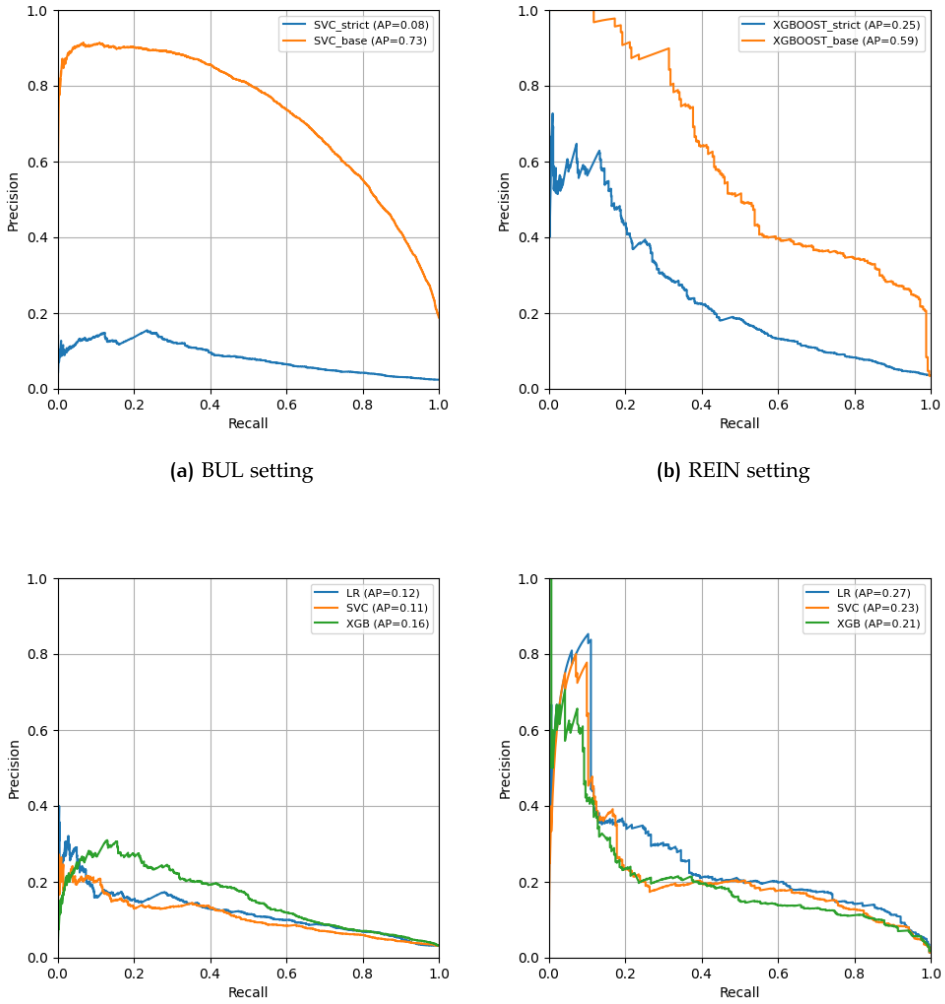
This section has visualized that different methodological assumptions have the power to make or break an EPS. By comparing the effect of these decisions to existing work, and analysing newly found assumptions, reported advancements in EPS design can be placed in context. Figures 5.9a and 5.9b show the final performance of the BUL and REIN baseline models when the most restricting, but also the most realistic methodological assumptions are respected (blue line), compared to their original settings (orange line). In figures 5.9c and 5.9d performance of the OWN baseline setting is evaluated on EDB ground truth as well as SYM ground truth. It can be concluded that only a fraction of the predictive power of EPS is sustained when realistic methodological assumptions are adhered to. In Section 5.2 it is attempted to improve this performance by leveraging a variety of machine learning techniques.

5.2 OPTIMIZATION

As follows from the previous section, EPS adhering to critical and realistic methodological assumptions tend to yield poor prediction results. In this section an effort is made to improve EPS performance under these circumstances, leveraging different techniques, while complying with realistic methodological assumptions. In Section 5.2.1 the performance of baseline EPS are presented. The following 3 sections cover different optimization strategies. First, performance is optimized by mitigating the effects of class imbalance in Section 5.2.2. Then is sections 5.2.3 and 5.2.4 optimization attempts through improved feature design and different (tuned) classification algorithms is conducted. In Section 5.2.5 the final, best performing models are presented.

5.2.1 Baseline performance in restricted environment

Figure 5.9 displays the final performance of EPS under the most restrictive and realistic assumptions. Specifically, 5.9d shows the result of 3 classifiers evaluated on a timeframe from 2013 until 2018, using SYM exploit-in-the-wild data as ground truth. This setting will be used as baseline for the optimization task in this chapter. The reason for this is that exploit-in-the-wild ground truth data is generally considered to have the closest resemblance to a real world situation. Performance metrics for the 3 classifiers (Logistic Regression, LinearSVC and XGBoost) can be found in Table 5.1. These models are considered the baseline in this optimization section.



(a) BUL setting

(b) REIN setting

(c) OWN setting

(d) OWN setting SYM ground truth

Figure 5.9: Final performance in most restrictive and realistic scenario

5.2.2 Imbalance handling

The first approach to improve the baseline exploit prediction models is by addressing the imbalanced nature of the data. The greatest amount of vulnerabilities will never get exploited, making an exploit event a rare event. Some machine learning models fail to effectively learn patterns when there is a heavy class imbalance within the data. As discussed earlier, in Section 4.2.2, there are different techniques to limit or eliminate this negative effect. The techniques used for the handling of class imbalance, can be subdivided in two groups; resampling techniques and algorithmic techniques (there are more ways to address imbalance, but these are out of scope for this thesis).

Resampling techniques

Altering the class distribution of a dataset can be done in two ways; undersampling of the majority class, or oversampling of the minority class. It is also possible to apply a combination of the two. Different undersampling techniques yield different results, because they adhere to other rules on which samples to exclude. The following techniques are evaluated: Random undersampling, undersampling by application of Tomek Links, Edited Nearest Neighbors, and One-sided selection.

Classifier	PRE	REC	F1
LR	0.696	0.105	0.182
SVC	0.462	0.08	0.135
XGBOOST	0.607	0.126	0.202

Table 5.1: Performance of baseline classifiers in restricting but realistic environment

Oversampling is the act of generating new syntetic data samples, which closely resemble samples of the minority class. Different techniques are evaluated: SMOTE, ANASYN, Borderline SMOTE, and SVMSMOTE. It is possible to control how much syntethic samples will be created. This is done be defining the sampling-strategy of the algorithms, which is used to specify the desired class-distribution after resampling. Lastly, a combination of over- and undersampling is tested, by first applying oversampling of the minority class, and then undersampling of the majority class. Results of this analysis can be found in Table 5.2.

Algorithmic technique

Machine learning models generally perform worse on imbalanced data, because the algorithm tends to be biased in favour of the majority class. The odds of a certain sample belonging to the majority class are bigger after all. With resampling one tries to eliminate this bias by creating equally distributed classes. Another way to address this bias is by assigning different costs to the mis-classification of samples of the majority and minority class. Exact working of this is outlined in Section 4.2.2. All algorithms are tested with a class weight between 0.8 and 0.99, with steps of 0.01, to find the class weight that yields the best performance. Results can be viewed in Table 5.2.

	LogReg			**LinearSVC**			**XGBoost**		
	PRE	REC	F1	PRE	REC	F1	PRE	REC	F1
Oversampling									
SMOTE	0.22	0.624	0.31	0.209	0.495	0.273	0.255	0.2	0.206
BorderlineSMOTE	0.252	0.426	0.286	0.229	0.395	0.261	0.184	0.205	0.169
SVMSMOTE	0.227	0.504	0.289	0.278	0.357	0.256	0.302	0.173	0.209
ADASYN	0.26	0.486	0.309	0.227	0.422	0.256	0.255	0.162	0.193
Undersampling									
RandomUndersamp	0.086	**0.872**	0.155	0.094	0.861	0.168	0.099	**0.907**	0.177
TomekLinks	**0.921**	0.144	0.242	**0.759**	0.141	0.224	0.433	0.164	0.23
EditedNN	0.433	0.26	0.254	0.306	0.319	0.268	0.209	0.303	0.225
OneSidedSelection	0.921	0.144	0.242	0.704	0.125	0.196	0.433	0.164	0.23
Combined									
SMOTE + ENN	0.223	0.599	**0.311**	0.204	**0.572**	0.284	0.198	0.538	**0.27**
Algorithmic									
Class weight	0.246	0.49	0.306	0.286	0.415	**0.301**	0.5	0.157	0.234
Baseline	0.696	0.105	0.182	0.462	0.08	0.135	**0.607**	0.126	0.202

Table 5.2: Results of different imbalance handling techniques on EPS performance

Design choice

As can be seen in Table 5.2, comparable results are obtained by the best resampling techniques and by the algorithmic approach. It depends on the classification algorithm which (combination of) resampling technique(s) works best. It is observed that handling class imbalance by adjusting class weights is nearly always about as effective as resampling of the data. The most successful resampling techniques are (partly) based on oversampling, which leads to bigger datasets. Working with a bigger dataset results in significantly higher training times. Because model optimization is already computationally expensive, the choice has been made to continue to use class weight as a measure to mitigate the effects of class imbalance in further optimization steps.

5.2.3 Feature design

Feature design is an important tool to obtain optimal predictions with the data at hand. As can be seen in feature Table 4.1, there are categorical, numerical and textual features. In the optimization process of the features the focus lies on improving the textual features. Natural Language Processing (NLP) is a promising field where more advanced techniques of representing textual data exist. The two main improvement areas focused on in this work are in the optimization of TF-IDF text vectorizers and in the optimization of NLP methods using word embeddings. The techniques are explained in more detail in Section 4.2.3.

Text Frequency - Inverse Document Frequency (TFIDF)

The TF-IDF technique has a few parameters to fine tune to obtain optimal results. The following parameters are considered: the n-gram range, the maximum amount of features, and whether or not to combine textual columns. Firstly, the so called n-gram range. A n-gram is a combination of words next to each other in a sentence, which are grouped together. Consider for example the following sentence; "I am exploited". If only 1-grams are considered all words are used separately (so "I", "am", "exploited"). If 2-grams are used also words next to each other are considered in addition to the 1-grams (so, "I am", "am exploited", "I", "am", "exploited"). The same pattern is followed for higher order n-grams. When n-grams are used, more attention is paid to the context of a word, which could hold important information for the prediction task. The tested n-gram ranges are (1,1) through (1,5), meaning only 1-grams and all n-grams up to and including 5-grams respectively. Secondly, the max features parameter is optimized. This parameter controls how many features can be created when vectorizing text. If this is not controlled, TD-IDF creates a feature for every word that occurs in one of the sentences, resulting in an enormous features set. By defining the max features parameter only the most important features are used. Limiting the amount of features used can positively impact the computational costs as well as EPS performance. The maximum features are varied from 500 through 2000, in steps of 500. Lastly, a parameter not directly related to TF-IDF if tuned. This parameter prescribes if the textual columns should be merged before TF-IDF analysis, or if the textual columns should be treated as separated sets of words (text columns; vulnerability description, domain name). Combining columns can influence the predictive power of vectorized text. Some distinctive words might for example stand out when considered in a context of its own text column, while becoming insignificant when combined with text from other columns. The decision of combining columns or not is binary, so either true or false. Results can be found in Table 5.3.

Custom word embeddings

Two forms of word embedding techniques are tested in this part of the feature optimization. The first flavour is when custom word embeddings are learned from the specific words in the textual data, to later form a feature set for the prediction task. When creating custom word embeddings different parameters can be altered to achieve optimal prediction results. The most important evaluated parameters are the following; vector size (defining the maximum length of the embedding of a sentence), the mode of combining word embeddings into sentence level embeddings (summation or averaging), type of training-method (either by Skip-gram or Bag of Words), and the minimal count of a word for it to be included in the model (value range from 5 to 20 in steps of 5). Two custom word embedding methods will be evaluated; Word2Vec and fastText

Pretrained word embeddings

Another form of NLP with word embeddings is by leveraging excising, pre-trained, word embeddings. There are several pre-trained models, with different training sets and slightly different training methods. The pre-trained models used are as follows: Word2Vec (trained on Google news), fastText (trained on Wiki news), Glove (trained on Twitter), and Glove (trained Wikipedia). The result of this different models can be seen in Table 5.3.

	LogReg			**LinearSVC**			**XGBoost**		
	PRE	REC	F1	PRE	REC	F1	PRE	REC	F1
TF-IDF	0.268	**0.597**	0.35	0.33	0.498	**0.346**	0.482	0.159	0.234
Word2Vec custom	0.308	0.501	**0.353**	0.303	0.441	0.316	0.439	0.119	0.184
Word2Vec	0.263	0.559	0.333	0.306	0.469	0.333	0.569	0.147	0.224
fastText custom	0.253	0.557	0.329	0.301	0.489	**0.346**	0.465	0.109	0.174
fastText	0.242	0.495	0.301	0.332	0.396	0.321	0.538	0.138	0.213
GloVe Twitter	0.36	0.386	0.34	0.297	**0.499**	0.339	0.496	**0.169**	**0.241**
GloVe Wiki	0.287	0.5	0.339	0.298	0.432	0.327	0.508	0.132	0.202
Baseline	**0.696**	0.105	0.182	**0.462**	0.08	0.135	**0.607**	0.126	0.202
With class-weight	0.246	0.49	0.306	0.286	0.415	0.301	0.5	0.157	0.234

Table 5.3: Results of different NLP techniques for extracting features from text

Design choice

The results of this optimization step, as can be observed in Table 5.3, show some improvements of the (balanced) baseline. Generally, the optimized TF-IDF text vectorization yields decent performance. For both the Logistic Regression and LinearSVC model, the F1 score is boosted by about 5 percent point compared to the class-weight rebalanced baseline. These improvements do not hold for the XGBoost where the optimization of TF-IDF did not have effect (although still the second best performning method). This experiment has shown that TF-IDF generally performs the best, or at least close to the best, for all baseline classifiers. Hence, the choice has been made to use TF-IDF in further optimization steps.

5.2.4 Algorithm optimization

The last step in this optimization section is algorithm optimization. This is done in both the choice of algorithm as well as in hyperparameter tuning, to obtain the opti-

mal results per algorithm. Section 5.2.4 covers the optimization of hyperparameters and in Section 5.2.4 different machine learning algorithms are evaluated.

Hyperparameter tuning

As described in more detail in Section 4.2.4, prediction algorithms often need to be tweaked to the data to perform optimally. This is done by exhaustively evaluating different parameter settings by means of a grid search. This step will be integrated with the analysis of Section 5.2.4, and results can be found in Table 5.4.

Choice of algorithm

Countless different machine learning algorithms are available for prediction tasks. Besides from the 3 baseline algorithms used in earlier analysis (Logistic Regression, LinearSVC and XGBoost), 8 additional algorithms will be analysed and tuned. Also, a voting classifier combining prediction results though majority voting will be tested, consisting of optimized KNN, AdaBoost, SGDC, SVC and LR classifiers.

Classifier	**PRE**	**REC**	**F1**
AdaBoost-DT	0.307	0.13	0.168
AdaBoost-LogReg	0.26	0.469	0.312
Decision Tree	0.241	0.309	0.245
KNN	0.**536**	0.25	0.315
Random Forrest	0.177	0.51	0.248
SGDC	0.38	0.512	0.354
SVC	0.229	0.583	0.306
Neural Network	0.333	0.553	0.351
Voting Classifier	0.442	0.375	**0.366**
Baseline (balanced, TF-IDF)			
Logistic Regression	0.268	**0.597**	0.35
LinearSVC	0.33	0.498	0.346
XGBoost	0.482	0.159	0.234

Table 5.4: Results of different tuned machine learning algorithms on EPS performance

5.2.5 Final Exploit Prediction System

Table 5.4 shows the final results of the exploit prediction system optimization effort in this thesis. This optimization has been achieved by handling class-imbalance, improving features and lastly by leveraging different (tuned) machine learning algorithms. The biggest advancements in performance was reached in by handling imbalance and the featurization of textual data. The last step where different algorithms have been tested has also proved itself useful, however the advancements were less dramatic and multiple different algorithms yield results close to that of the best algorithm. The optimal exploit prediction system found in this section is a VotingClassifier, combining prediction results from the KNN, AdaBoost, SGDC, SVC and LR classifiers, which yielded a F1-score of 0.366. This is a slight improvement over the performance of its separate classifiers. This classifier operated on data which was balanced by using the class weight method, and used TF-IDF for the featurization of text. The baseline models (LR, LinearSVC, XGB) had F1-scores of 0.182, 0.135 and 0.202 respectively. The F1-score for the optimized EPS was 0.366, which is an improvement of the baseline by 200%, 270% and 180% respectively. This drastic improvement highlights the importance of proper handling of class-imbalance and

feature design.

5.3 IMPLICATIONS FOR INDUSTRY

The results in this chapter have shown that EPS are very sensitive to their surroundings. The effects of six methodological assumptions on EPS performance are evaluated in Section 5.1. The results of these experiments demonstrated that the assumptions prior works build their work on have a fundamental impact on the reported performance of EPS. When the most restrictive but realistic assumptions are complied to, only a fraction of the predictive power of the evaluated models is sustained.

Literature review has shown that a lot of EPS in prior work fall victim to at least some unrealistic assumptions, and thereby reporting inflated performance metrics. Some are vague in what assumptions they apply leading to uncertainty on how to interpret their results. Extra uncertainty emerges when closed-source data is used, making it impossible to reproduce and verify the results.

The optimization steps as outlined in Section 5.2, display effective opportunities for improvement of EPS performance under realistic assumptions. Substantial improvements are possible by handling class imbalance, optimizing feature design and leveraging different tuned algorithms. Despite these developments, the performance of the evaluated EPS are insufficient to be able to justify their deployment in the industry.

6 CONCLUSION AND RECOMMENDATIONS

The growing number of vulnerabilities released and requiring attention is posing a challenge to many organisations. Patching these vulnerabilities is a resource intensive practice and only a small fraction of all vulnerabilities ever get exploited in the wild, and actually pose a *real* threat. Prioritization is key in an effective and efficient patching strategy. Current approaches for patch prioritization are failing and are sometimes no better than random guessing. EPS are trying to fill this gap by leveraging machine learning to predict the exploitability of a vulnerability, which then can be used to effectively prioritize vulnerabilities to mitigate.

Exploit prediction is a challenging task with scarce (high-quality) data and heavy class-imbalance. Over the last decade several researchers have attempted to optimize EPS. Their approaches vary in what data is used, which machine learning techniques are used and how the models are evaluated. Another important factor is how several design decisions are influenced by methodological assumptions. Prior works like Bullough et al. [2017] and Reinthal et al. [2018] identify and acknowledge some of these assumptions and their impact on EPS performance.

The first goal of this thesis was to identify all of the critical methodological assumptions in EPS design, their prevalence in prior work, and the magnitude of their effects. Six critical assumptions have been identified in the area of realistic data collection, the correct processing of data, and proper evaluation of the model. Prior works vary greatly in what assumptions they base their prediction models on. Almost all works fall victim to at least one faulty or unrealistic methodological assumption, and thereby report overoptimistic results. Some works don't clearly state which assumptions they make, leading to uncertainty on how to interpret their results. Extra uncertainty emerges when closed-source data is used, making it impossible to reproduce and verify the results. The effect these methodological assumptions have on EPS performance has been tested by means of experiments. In these experiments the most strict/realistic option is set out against the less strict/unrealistic option. Almost all of the realistic choices have a considerable negative impact on the EPS performance. Some of the experiments of Bullough et al. [2017] and Reinthal et al. [2018] are reproduced in this thesis. Comparable results are obtained in most cases. Except for experiment 2 evaluating the exclusion of zero-day exploits. Reinthal et al. [2018] reports opposite behaviour from what is observed in this thesis and in Bullough et al. [2017]. When all realistic choices are made in the process of designing an EPS, only a fraction of the predictive power of the evaluated models is sustained.

These realistic models form the basis of the second phase of this thesis; the optimization of EPS under restricting yet realistic circumstances. The goal of this part was to optimize the realistic baseline models to improve their performance. This optimization is done by exploring different techniques to effectively handle class imbalance, create richer textual features and/or leverage different prediction algorithms. Substantial improvements are achieved in this optimization step, with a final EPS with a F1-score of 0.366. This is a 200%, 270% and 180% increase in performance compared to the baseline models (LR, LinearSVC, XGB). This score is higher than the score reported by Bullough et al. [2017] in the most restrictive setting. While this optimization step made a big difference, the final performance

of the EPS is still too low to be of value in a real world setting. It might be tempting to improve performance at the expense of sound methodological decisions, but this undermines the effort altogether.

The third contribution of this thesis is the implementation and release of a framework that enables easy experimentation with different kind of machine learning techniques for exploit prediction. The Exploit Prediction Framework will be published open-source. Hopefully this will contribute to a more open and transparent community and pave the way for cooperation in the field, instead of reinventing the wheel.

6.1 RECOMMENDATIONS

The combination of overoptimistic reports of EPS performance due to unrealistic methodological assumptions, the inability to objectively verify most EPS systems, and the fact that attaining workable results in EPS is very challenging, leads to the need for extreme caution when using existing EPS in a production environment. Putting your full trust in existing systems would be a risk. The main recommendation for players in the industry, such as Adyen, is to be careful with the the adoption of EPS. With the current level of maturity, exploit prediction could have value as a complementary measure to existing vulnerability prioritization systems. Further improvements and more transparent systems are essential for EPS to be suitable for practical usage.

One of the main challenges that is holding back this improvement is the scarcity of high-quality data. Exploit data is often commercially owned by for example security solution providers, who are not keen on sharing this data. Looking for new ways to obtain more data from higher quality is of vital importance for progress in EPS and would be a valuable direction for further research. Other means of EPS refinement, such as creating richer features, consuming additional information sources, experimenting with the use of other NLP techniques for text processing, and exploration of other classification algorithms would also be useful future work.

Besides better performance, there is a need for openness and transparency in EPS. Systems with a black box nature can not be verified objectively and could possibly lead to a false sense of security. The source code of this work has been published open-source to set an example.

6.2 LIMITATIONS

This study has potential limitations. The results produced in this thesis rely on the used data and the chosen machine learning techniques. The following potential limitations are identified.

- Limitation of ground truth data. Ground truth data for the final EPS is obtained from Symantec's Attack Signatures and Intrusion Protection Signatures. Although considered the best available data for exploit-in-the-wild prediction, it only contains Microsoft and Adobe exploits and might thus be biased. Efforts have been put in mitigating this bias by filtering vendor information from the feature set. Despite this, it might still influence the ability of the EPS to generalize to vulnerabilities of other vendors than Microsoft and Adobe. Another limitation of the ground truth is that non-exploitation can not be guaranteed. A vulnerability is labeled as exploited if a detection in a real system

has occurred. While this gives certainty about a detected vulnerability being exploited, it does not guarantee that vulnerabilities that are not detected, are actually not exploited. The reason for this is that detection method might have missed it. This limitation holds for any EPS in the exploit prediction domain.

- Limitless techniques exist in machine learning which can be used for creation an optimization of EPS. This calls for a clear scoping on which methods to include, and which not. While this has been done with care, it might be that other methods are more suitable for the task, and yield better performance for EPS.

BIBLIOGRAPHY

Adyen (2020). Anual report 2020. Technical report, Adyen.

Allodi, L. and Massacci, F. (2012). A preliminary analysis of vulnerability scores for attacks in wild: The ekits and sym datasets. In *Proceedings of the 2012 ACM Workshop on Building analysis datasets and gathering experience returns for security*, pages 17–24.

Allodi, L. and Massacci, F. (2014). Comparing vulnerability severity and exploits using case-control studies. *ACM Transactions on Information and System Security (TISSEC)*, 17(1):1–20.

Allodi, L., Shim, W., and Massacci, F. (2013). Quantitative assessment of risk reduction with cybercrime black market monitoring. In *2013 IEEE Security and Privacy Workshops*, pages 165–172. IEEE.

Almukaynizi, M., Nunes, E., Dharaiya, K., Senguttuvan, M., Shakarian, J., and Shakarian, P. (2017). Proactive identification of exploits in the wild through vulnerability mentions online. *2017 IEEE International Conference on Cyber Conflict U.S., CyCon U.S. 2017 - Proceedings*, 2017-Decem:82–88.

Almukaynizi, M., Nunes, E., Dharaiya, K., Senguttuvan, M., Shakarian, J., and Shakarian, P. (2019). *Patch Before Exploited: An Approach to Identify Targeted Software Vulnerabilities*. Springer International Publishing.

Alperin, K., Wollaber, A., Ross, D., Trepagnier, P., and Leonard, L. (2019). Risk prioritization by leveraging latent vulnerability features in a contested environment. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 49–57.

Beattie, S., Arnold, S., Cowan, C., Wagle, P., Wright, C., and Shostack, A. (2002). Timing the application of security patches for optimal uptime. In *LISA*, volume 2, pages 233–242.

Bhatt, N., Anand, A., and Yadavalli, V. S. (2021). Exploitability prediction of software vulnerabilities. *Quality and Reliability Engineering International*, 37(2):648–663.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Bozorgi, M., Saul, L. K., Savage, S., and Voelker, G. M. (2010). Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 105–114.

Bullough, B. L., Yanchenko, A. K., Smith, C. L., and Zipkin, J. R. (2017). Predicting exploitation of disclosed software vulnerabilities using open-source data. *IWSPA 2017 - Proceedings of the 3rd ACM International Workshop on Security and Privacy Analytics, co-located with CODASPY 2017*, pages 45–53.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

Chen, H., Liu, R., Park, N., and Subrahmanian, V. S. (2019). Using twitter to predict when vulnerabilities will be exploited. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 3143–3152.

CISA (2015). Enhancing resilience through cyber incident data sharing and analysis.

Dey, D., Lahiri, A., and Zhang, G. (2015). Optimal policies for security patch management. *INFORMS Journal on Computing*, 27(3):462–477.

Edkrantz, M. (2015). Predicting Exploit Likelihood for Cyber Vulnerabilities with Machine Learning. *Undefined*, page 57.

Fang, Y., Liu, Y., Huang, C., and Liu, L. (2020). Fastembed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. *PLoS ONE*, 15(2):1–28.

Hoque, M. S., Jamil, N., Amin, N., and Lam, K.-Y. (2021). An Improved Vulnerability Exploitation Prediction Model Vector Embedding. pages 1–17.

Jacobs, J., Romanosky, S., Adjerid, I., and Baker, W. (2020). Improving vulnerability remediation through better exploit prediction. *Journal of Cybersecurity*, 6(1):1–12.

Jacobs, J., Romanosky, S., Edwards, B., Roytman, M., and Adjerid, I. (2019). Exploit Prediction Scoring System (EPSS). pages 1–25.

Jonathan, J. (2017). "wannacry" ransomware attack losses could reach $4 billion.

Marin, E., Diab, A., and Shakarian, P. (2016). Product offerings in malicious hacker markets. In *2016 IEEE conference on intelligence and security informatics (ISI)*, pages 187–189. IEEE.

Microsoft (2017). Microsoft Patch Tuesday - 14 March 2017 . "https://support.microsoft.com/en-us/topic/march-14-2017-update-for-microsoft-office-89f120a1-3c51-7b0a-ef68-97c7b11cdeea". Accessed: 2022-02-21.

Miranda, L., Vieira, D., De Aguiar, L. P., Menasche, D. S., Bicudo, M. A., Nogueira, M. S., Martins, M., Ventura, L., Senos, L., and Lovat, E. (2021). On the Flow of Software Security Advisories. *IEEE Transactions on Network and Service Management*, 18(2):1305–1320.

MITRE (2022a). CVE Reference Map for Source EXPLOIT-DB. `https://cve.mitre.org/data/refs/refmap/source-EXPLOIT-DB.html`. [offline; accessed Fev-2022].

MITRE (2022b). MITRE CVE statistics . "https://www.cve.org/About/Metrics". Accessed: 2022-02-21.

Mittal, S., Das, P. K., Mulwad, V., Joshi, A., and Finin, T. (2016). Cybertwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 860–867. IEEE.

Nayak, K., Marino, D., Efstathopoulos, P., and Dumitraş, T. (2014). Some vulnerabilities are different than others. In *International Workshop on Recent Advances in Intrusion Detection*, pages 426–446. Springer.

PCI (2018). Payment card industry (pci) - data security standard. Technical report, PCI.

Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Ponemon (2020). Costs and consequences of gaps in vulnerability response. Technical report, Ponemon.

Reinthal, A., Filippakis, E. L., and Almgren, M. (2018). *Data Modelling for Predicting Exploits*, volume 11252 LNCS. Springer International Publishing.

Rodriguez, L. G. A., Trazzi, J. S., Fossaluza, V., Campiolo, R., and Batista, D. M. (2018). Analysis of Vulnerability Disclosure Delays from the National Vulnerability Database. *Workshop de Segurança Cibernética em Dispositivos Conectados (WSCDC$_S$BRC)*, 1.

Sabottke, C., Suciu, O., and Dumitras, T. (2015). Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. *Proceedings of the 24th USENIX Security Symposium*, pages 1041–1056.

Samtani, S., Chinn, K., Larson, C., and Chen, H. (2016). Azsecure hacker assets portal: Cyber threat intelligence and malware analysis. In *2016 IEEE conference on intelligence and security informatics (ISI)*, pages 19–24. Ieee.

Suciu, O., Nelson, C., Lyu, Z., Bao, T., and Dumitras, T. (2021). Expected Exploitability: Predicting the Development of Functional Vulnerability Exploits.

Symantec (2018a). Symantec A-Z Listing of Threats Risks. https://www.symantec.com/security-center/a-z. [offline; accessed Nov-2021 through WaybackMachine].

Symantec (2018b). Symantec Attack Signatures. https://www.symantec.com/security_response/attacksignatures/. [offline; accessed Nov-2021 through WaybackMachine].

Tavabi, N., Goyal, P., Almukaynizi, M., Shakarian, P., and Lerman, K. (2018). DarkEmbed: Exploit prediction with neural language models. *Proceedings of the 30th Innovative Applications of Artificial Intelligence Conference, IAAI 2018*, pages 7849–7854.

Yang, H., Park, S., Yim, K., and Lee, M. (2020). Better not to use vulnerability's reference for exploitability prediction. *Applied Sciences (Switzerland)*, 10(7).

Yin, J., Tang, M., Cao, J., Wang, H., You, M., and Lin, Y. (2021). Vulnerability exploitation time prediction: an integrated framework for dynamic imbalanced learning. *World Wide Web*.

Zhang, F. and Li, Q. (2020). Dynamic Risk-Aware Patch Scheduling. *2020 IEEE Conference on Communications and Network Security, CNS 2020*.

A METHODOLOGICAL ASSUMPTIONS IN RELATED WORKS

Summary table A.1 can be found on the next page in landscape orientation.

Author	Year	Title	Time-frame	ACC	PRE	REC	F1	A1: Ground truth	A2: Order of events	A3: Dist. of test-set	A4: Label leakage	A5: Cross-validation	A6: Parameter-tuning
Bhatt	2021	Exploitability prediction of software vulnerabilities	2012-2015	0.88	0.91	0.85	0.88	EDB	-	Resamp. of test	Prevented	Strat. 10-fold	No optimization
Hoque	2021	An Improved Vulnerability Exploitation Prediction Model Vector Embedding	1997-2020	0.92	0.89	0.98	0.94	EDB, SYM	Not respected	Resamp. of test	Prevented	10-fold	Not separated
Suciu	2021	Expected Exploitability: Predicting the Development of Functional Vulnerability Exploits	1999-2020	-	-	-	-	Commercial	-	No resamp.	-	Temporal	Not separated?
Yin	2021	Vulnerability exploitation time prediction: an integrated framework for dynamic imbalanced learning	1999-2020	-	-	-	0.56	EDB	Not respected	Resamp. of test	Prevented	Temporal	-
Fang	2020	Fastembed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm	2013-2018	0.93	0.62	0.25	0.36	SecFocus, EDB, SYM	Not respected	No resamp. on test	Prevented	Strat. 10-fold	Separated

Table A.1: Summary table with methodological assumptions in related works (1/4)

Author	Year	Title	Time-frame	ACC	PRE	REC	F1	A1: Ground truth	A2: Order of events	A3: Dist. of test-set	A4: Label leakage	A5: Cross-validation	A6: Parameter-tuning
Jacobs	2020	Improving vulnerability remediation through better exploit prediction	2009-2018	-	0.88	0.23	0.45	Commercial	Not respected	No resamp. on test	Prevented	Strat. 5-fold	Separated
Yang	2020	Better not to use vulnerability's reference for exploitability prediction	?	0.83	0.56	0.80	0.65	EDB	-	Resamp. of test	Prevented	Strat. 5-fold	No optimization
Zhang	2020	Dynamic Risk-Aware Patch Scheduling	1990-2019	-	-	-	-	EDB	Not respected	Resamp. of test	Prevented	Temporal	No optimization
Almukaynizi	2019	Patch Before Exploited: An Approach to Identify Targeted Software Vulnerabilities	2015-2016	-	0.45	0.35	0.40	SYM	Experimenting	No resamp. on test	Prevented	Temporal	No optimization
Alperin	2019	Risk prioritization by leveraging latent vulnerability features in a contested environment	1999-2019	-	-	-	-	EDB	-	No resamp.		Strat. 5-fold	No optimization
Chen	2019	Using twitter to predict when vulnerabilities will be exploited	2016-2018	-	0.56	0.68	0.59	EDB, SYM	Not respected	No resamp. on test	Unsure	Temporal	No optimization

Table A.2: Summary table with methodological assumptions in related works (2/4)

Author	Year	Title	Time-frame	ACC	PRE	REC	F1	A1: Ground truth	A2: Order of events	A3: Dist. of test-set	A4: Label leakage	A5: Cross-validation	A6: Parameter-tuning
Jacobs	2019	Exploit Prediction Scoring System (EPSS) Jay	2016-2018	-	-	-	-	Commercial	Not mentioned	No resamp.	Prevented	5-fold & Temporal	-
Reinthall	2018	Data Modelling for Predicting Exploits	2015-2018	-	-	-	-	EDB	Experimenting	No resamp.	Not-prevented	Experimenting	Separated
Tabavi	2018	DarkEmbed: Exploit prediction with neural language models	2010-2017	-	-	-	-	SYM, Metas-ploit	Respected	No resamp.	Not-prevented	-	Not separated?
Almukaynizi	2017	Proactive identification of exploits in the wild through vulnerability mentions online	2015-2016	-	0.67	0.38	0.48	SYM	Not respected	No resamp.	Not-prevented	Temporal	No optimization
Bullough	2017	Predicting exploitation of disclosed software vulnerabilities using open-source data	2009-2015	-	-	-	-	EDB	Experimenting	Experimenting	Prevented	Experimenting	No optimization
Edkrantz	2015	Predicting Exploit Likelihood for Cyber Vulnerabilities with Machine Learning	2005-2014	0.83	0.82	0.84	0.83	EDB	Not respected	Resamp. of test	Prevented	10-fold	Separated

Table A.3: Summary table with methodological assumptions in related works (3/4)

Author	Year	Title	Time-frame	ACC	PRE	REC	F1	A1: Ground truth	A2: Order of events	A3: Dist. of test-set	A4: Label leakage	A5: Cross-validation	A6: Parameter-tuning
Sabottke	2015	Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploit	2014-2015	-	-	-	-	SYM, EDB, OSVDB, Microsoft	Respected	Resamp. of test	Unsure	Strat. 10-fold	Not separated
Bzorgi	2010	Beyond heuristics: Learning to classify vulnerabilities and predict exploits	1991-2007	0.90	-	-	-	OSVDSB	Not respected	Resamp. of test		10-fold	Separated

Table A.4: Summary table with methodological assumptions in related works (4/4)

B EXPLOIT PREDICTION FRAMEWORK

In this section the configuration file of the Exploit Prediction Framework will be elaborated. This configuration file is used to specify all experimental variables. This file should be provided in *json* format and consist of 7 top level parameter categories. These categories will be explained in further detail in the sections of this chapter. In an experiments, two or more experiment scenarios can be provided by concatenating the scenarios in list form.

B.1 GENERAL PARAMETERS

The general parameters are displayed in listing B.1. In this part of the configuration file, the experiment name can be defined by setting `EXP_NAME`, this is also the name of the directory that will be created for the results. Then the period of analysis is chosen by setting the `START_DATE` and `END_DATE`. The `START_BATCH` and `LAG_SIZE` parameters have to do with temporal cross-validation. The start batch is the batch that an initial classifier is trained on before the rolling window is started. Once a classifier is trained on the start batch, the lag size decides from how far back the data will be used. As an example, in most experiments this value is 12, meaning that data until 12 months prior to the prediction date is considered. If this value is set to 0, all available historic data is used. The `LABEL_PERIOD` defines the maximum amount of months the exploitation date of a vulnerability can be after the disclosure date, for it still to be labeled as exploited. The default value used in this thesis is 6 months. So the vulnerabilities that are exploited more than 6 months after the disclosure date are discarded. This rule is used to obtain binary labels, and to be able to say something about the timeframe of a prediction. The `RANDOM_STATE` parameter defines the seed for different machine learning techniques for repeatability concerns. If set to `null` the experiment yield different results every time they are run. If `USE_CACHE` is set, different time consuming elements of the experiment are using cached data instead of rerunning all calculations. This can for example be cleaned data or trained word embedding models. Lastly, the `W2V_PRETRAINED_PATH` parameter, holds the path to pretrained word vector models such as Word2Vec, GloVe or fastText.

```
1  "general": {
2    "EXP_NAME": "1.1_GT_OWN",
3    "START_DATE": "2013-01-01",
4    "END_DATE": "2018-01-01",
5    "START_BATCH": 12,
6    "LAG_SIZE": 0,
7    "LABEL_PERIOD": 6,
8    "RANDOM_STATE": 42,
9    "USE_CACHE": false,
10   "W2V_PRETRAINED_PATH": "../path/to/pretrained.vec"
11 }
```

Listing B.1: General parameters

B.2 DATA PARAMETERS

The data parameters define what data will be used for the experiment, and how it will be preprocessed. The example in listing B.2 shows the basic structure of an experiment with 2 scenario's using different ground truth data. The `INPUT` parameter is used to choose what input data should be used for the experiment. By default data from the NVD is used, but it is also possible to use data from the IBM X-Force ThreatExchange. Then, the `CVSSv` specifies what version of CVSS to use, either 2 or 3. This has a influence on the amount of total samples, because not all vulnerabilities have CVSS version 3 information available. When version 3 is chosen, samples without a CVSS version 3 score are dropped. The `GT` parameter is used to define what ground truth data will be used, either PoC based ground truth from EDB, or exploit in the wild ground truth from Symantec. `FILTER_LEAK`, `FILTER_SYM` and `EXCLUDE_PAST` are parameters that affect the filtering of some datasamples. If `FILTER_LEAK` is set, label leakage is avoided by removing references that possibly leak information about the labels. Another form of label leakage is avoided by setting `FILTER_SYM`. This parameter is explicitly for the SYM ground truth. It filters the textual description of the vulnerability for references for vendor information. Lastly, the `EXCLUDE_PAST` parameter is used to specify if exploits that have occured prior to the disclosure data should be filtered out or not.

```
12  "data": [
13    {
14      "INPUT": "NVD",
15      "CVSSv": 2,
16      "GT": "EDB",
17      "FILTER_LEAK": true,
18      "FILTER_SYM": true,
19      "EXCLUDE_PAST": true
20    },
21    {
22      "INPUT": "NVD",
23      "CVSSv": 2,
24      "GT": "SYM",
25      "FILTER_LEAK": true,
26      "FILTER_SYM": true,
27      "EXCLUDE_PAST": true
28    }
29  ]
```

Listing B.2: Data parameters, example of experiment with 2 different ground truth datasets

B.3 OPTIMIZATION PARAMETERS

The optimization parameters as listed in listing B.3, are used to define the specifics for parameter tuning. The first parameter, `TUNE_PARAMS`, is used when optimization should take place. If it is false, only 1 model should be provided. If set to true, a model should be defined by setting parameter ranges which will be explored. When `GS_RANDOM` is set to true, *sklearn*'s RandomizedGridSearchCV will be used in stead of GridSearchCV. This means that of all possible different parameter settings, only `GS_ITER` are randomly chosen for evaluation. If `GS_NESTED` is set to true, a nested gridsearch will be conducted. The last 3 parameters are parameters for this nested gridsearch. So, firstly `GS_INNER_TEMP` decides if the inner loop of the gridsearch should also be split temporally or not. `CANDIDATE_SCORES` specifies if scores of all candidates in the inner gridsearch should be returned or not (more computation-

ally expensive). Lastly, `OPT_THRESH` is set if also the classification threshold of the classifier should be optimized.

```
30  "optimization": [
31    {
32      "TUNE_PARAMS": false,
33      "GS_RANDOM": false,
34      "GS_ITER": 2,
35      "GS_NESTED": false,
36      "GS_INNER_TEMP": false,
37      "CANDIDATE_SCORES": false,
38      "OPT_THRESH": false
39    }
40  ]
```

Listing B.3: Optimization parameters

B.4 CROSS-VALIDATION PARAMETERS

Listing B.4 displays the cross-validation parameters, used for defining how model evaluation should be conducted. If `TEMPORAL` is set, temporal cross-validation with a rolling window of size `WINDOW_SIZE` is used (procedure as shown in 4.3). If `TEMPORAL` is not set basic k-fold cross-validation is used for evaluation of the experimental models, with k set equal to `FOLDS`. `STRATIFIED`, `SHUFFLE` are options for the k-fold cross-validation. Lastly, the `RESAMPLE_TEST` can be set if both training and testing data should be resampled (this is methodically incorrect, but is used to show the impact in one of the experiments).

```
41  "crossval": [
42    {
43      "TEMPORAL": true,
44      "WINDOW_SIZE": 6,
45      "FOLDS": 5,
46      "STRATIFIED": false,
47      "SHUFFLE": true,
48      "RESAMPLE_TEST": false
49    }
50  ]
```

Listing B.4: Cross-validation parameters

B.5 MODEL PARAMETERS

This section lists the model parameters, consisting of classifier parameters and featurization parameters. Listing B.5 shows the 3 baseline classifiers and some basic parameter ranges for optimization. The classifiers should be described in key value pairs, with the classifier name as a key and its parameters in list form as the value. The classifier parameters should be supplied as a stringified python dicts, so that after evaluation and concatenation, a parameter dictionary as accepted by *sklearn*'s GridSeachCV is the result.

```
51    "LR": [
52      "{'clf': [LogisticRegression()]}",
53      "{'clf__C': np.logspace(-4, 4, 8)}",
54      "{'clf__class_weight': [{0:x, 1:1.0-x} for x in np.linspace(0.
           01,0.2,10)]}"
55    ],
56    "XGB": [
57      "{'clf': [XGBClassifier()]}",
58      "{'clf__eval_metric': ['logloss']}",
59      "{'clf__objective': ['binary:logistic']}",
60      "{'clf__max_depth': [3, 5, 15]}",
61      "{'clf__n_estimators': [100, 300, 600]}",
62      "{'clf__scale_pos_weight': np.linspace(0.8,0.99,10)}"
63    ],
64    "LinearSVC": [
65      "{'clf': [LinearSVC()]}",
66      "{'clf__C': np.logspace(-4, 4, 8)}",
67      "{'clf__class_weight': [{0:x, 1:1.0-x} for x in np.linspace(0.
           01,0.2,10)]}"
68    ]
```

Listing B.5: Classifier parameters

The last part of the configuration consists of the feauturization settings of the model (listing B.6). Here common preprocessing steps can be defined, such as what numerical scaler to use, which textual columns are used and if they should be combined or not. In the specific preprocessing dictionary, different NLP techniques can be defined used for the vectorization of the textual columns. Multiple entries can be supplied in list from to experiment with different NLP techniques.

```
69  "preprocess_common": [
70    "{'preprocessor__num__scaler': [StandardScaler()]}",
71    "{'preprocessor__text__vectorizer__columns': [['description', '
         domain', 'vendor']]}",
72    "{'preprocessor__text__vectorizer__combine_cols': [True]}"
73  ],
74  "preprocess_specific": {
75    "tfidf" : [
76      "{'preprocessor__text__vectorizer':[MultiColumnTfidfVectorizer
           (None)]}",
77      "{'preprocessor__text__vectorizer__max_features': [1000]}",
78      "{'preprocessor__text__vectorizer__ngram_range': [(1,4)]}",
79      "{'preprocessor__text__vectorizer__sublinear_tf': [True]}"
80    ]
81  }
```

Listing B.6: Featurization parameters

C ALL RESULTS

C.1 ASSUMPTION EXPERIMENTS

Experiment 1

Classifier	**Baseline**			**Experiment**			**Diff**
	PRE	REC	F1	PRE	REC	F1	F1
LR	0.258	0.022	0.041	0.818	0.095	0.171	0.13
SVC	0.429	0.004	0.008	0.692	0.032	0.061	0.053
XGB	0.244	0.043	0.073	0.359	0.131	0.192	0.119
Average	0.31	0.023	0.041	0.623	0.086	0.141	0.101

Table C.1: Experiment 1: EDB versus SYM ground truth (OWN)

Experiment 2

Classifier	**Baseline**			**Experiment**			**Diff**
	PRE	REC	F1	PRE	REC	F1	F1
SVC	0.773	0.548	0.641	0.667	0.01	0.02	-0.621
Average	0.773	0.548	0.641	0.667	0.01	0.02	-0.621

Table C.2: Experiment 2: Exclusion of zero-day exploits (BUL)

Classifier	**Baseline**			**Experiment**			**Diff**
	PRE	REC	F1	PRE	REC	F1	F1
XGBOOST	0.962	0.864	0.91	0.709	0.368	0.485	-0.425
Average	0.962	0.864	0.91	0.709	0.368	0.485	-0.425

Table C.3: Experiment 2: Exclusion of zero-day exploits (REIN)

Classifier	Baseline			Experiment			Diff
	PRE	REC	F1	PRE	REC	F1	F1
SVC	0.729	0.615	0.667	0.161	0.467	0.239	-0.428
Average	0.729	0.615	0.667	0.161	0.467	0.239	-0.428

Table C.4: Experiment 2: Exclusion of zero-day exploits with SMOTE (BUL)

Classifier	Baseline			Experiment			Diff
	PRE	REC	F1	PRE	REC	F1	F1
XGBOOST	0.949	0.867	0.906	0.601	0.421	0.495	-0.411
Average	0.949	0.867	0.906	0.601	0.421	0.495	-0.411

Table C.5: Experiment 2: Exclusion of zero-day exploits with SMOTE (REIN)

Experiment 3

Classifier	Baseline			Experiment			Diff
	PRE	REC	F1	PRE	REC	F1	F1
SVC	0.776	0.551	0.644	0.846	0.858	0.852	0.208
Average	0.776	0.551	0.644	0.846	0.858	0.852	0.208

Table C.6: Experiment 3: Resampling of testset (BUL)

Experiment 4

Classifier	Baseline			Experiment			Diff
	PRE	REC	F1	PRE	REC	F1	F1
LR	0.54	0.126	0.204	0.321	0.028	0.052	-0.152
SVC	0.49	0.135	0.211	0.202	0.023	0.041	-0.17
XGB	0.032	1.0	0.062	0.032	1.0	0.062	0.0
Average	0.354	0.42	0.159	0.185	0.35	0.052	-0.107

Table C.7: Experiment 4: Filtering of label leaking references (REIN)

Classifier	Baseline			Experiment			Diff
	PRE	REC	F1	PRE	REC	F1	F1
XGBOOST	0.641	0.394	0.488	0.69	0.114	0.196	-0.292
Average	0.641	0.394	0.488	0.69	0.114	0.196	-0.292

Table C.8: Experiment 4: Filtering of label leaking references (OWN)

Experiment 5

Classifier	k-Fold			Cutoff			Online		
	PRE	REC	F1	PRE	REC	F1	PRE	REC	F1
XGBOOST	0.776	0.55	0.644	0.526	0.226	0.316	0.59	0.479	0.521
Average	0.776	0.55	0.644	0.526	0.226	0.316	0.59	0.479	0.521

Table C.9: Experiment 5: Cross-validation (BUL)

Classifier	k-Fold			Cutoff			Online		
	PRE	REC	F1	PRE	REC	F1	PRE	REC	F1
XGBOOST	0.725	0.509	0.585	0.641	0.394	0.488	0.679	0.492	0.56
Average	0.725	0.509	0.585	0.641	0.394	0.488	0.679	0.492	0.56

Table C.10: Experiment 5: Cross-validation (REIN)

Experiment 6

Classifier	Baseline			Experiment			Diff
	PRE	REC	F1	PRE	REC	F1	F1
LR	0.18	0.108	0.135	0.124	0.204	0.154	0.019
SVC	0.128	0.068	0.089	0.142	0.104	0.12	0.031
XGB	0.032	1.0	0.062	0.032	1.0	0.062	0.0
Average	0.113	0.392	0.095	0.099	0.436	0.112	0.017

Table C.11: Experiment 6: Parameter tuning with regular gridsearch compared to a nested gridsearch

COLOPHON

This document was typeset using LaTeX. The document layout was generated using the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from André Miede.