Reacting to “n-day” vulnerabilities in information systems

Clément Elbaz

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

« Clément ELBAZ »

« Reacting to "N-Day" Vulnerabilities in Information Systems »

Thèse présentée et soutenue à Rennes, le 30 Mars 2021
Unité de recherche : Inria Rennes - Bretagne Atlantique

Rapporteurs avant soutenance :

Hervé Debar	Professeur, Télécom SudParis
Olivier Festor	Professeur, Université de Lorraine

Composition du Jury :

Président :	Sophie Pinchinat	Professeur, Université de Rennes 1
Examinateurs :	Rémi Badonnel	Maître de conférences, Université de Lorraine
	Anaël Beaugnon	Data Scientist, Roche
	Hervé Debar	Professeur, Télécom SudParis
	Olivier Festor	Professeur, Université de Lorraine
	Sophie Pinchinat	Professeur, Université de Rennes 1
	Sasha Romanosky	Senior Policy Researcher, RAND Corporation
Dir. de thèse :	Christine Morin	Directrice de recherche, Inria Rennes – Bretagne Atlantique
Co-dir. de thèse :	Louis Rilling	Ingénieur-chercheur, DGA Maîtrise de l'Information

Invité(s) :

Eddy Caron	Maître de conférences, HDR, ENS Lyon

Pour ma fille Emma, née durant cette thèse. Les intelligences artificielles font pâle figure à côté de te voir grandir, apprendre, et découvrir le monde.

Et pour ma grand-mère Cécile, « Mam ». Durant sa vie de 1927 à 2018, elle a vu les hommes inventer bien des façons de se blesser les uns les autres. J'espère que les travaux décrits dans ces pages aideront à empêcher quelques blessures.

Acknowledgement

Quelle thèse !

Quand j'ai commencé à travailler à Inria en Octobre 2017, le monde était bien différent d'aujourd'hui. Terminer ce doctorat durant la pandémie du siècle, avec un jeune enfant à la maison, fut certainement une épreuve « intéressante ».

Toutefois, du 1er Octobre 2017 au 30 Mars 2021, une constante fut le plaisir que j'ai eu à travailler chaque jour sur mes sujets de recherche, et je le dois beaucoup à mes directeurs de thèse **Christine Morin** et **Louis Rilling** que je remercie chaleureusement. J'ai énormément apprécié l'autonomie et la confiance que vous m'avez accordées durant ces trois années, me permettant même d'emmener cette thèse vers des directions qu'aucun de nous n'avait prévu initialement. Dans le même temps vous avez toujours été présents et attentifs, un jour à relever les limites de ma réflexion du moment, un autre à passer un brouillon d'article au peigne fin pour en faire le meilleur manuscrit possible. En plus de nos intérêts scientifiques communs, nous partageons tous les trois un perfectionnisme de la rédaction et j'ai beaucoup aimé écrire à vos côtés.

Je voudrais également remercier mon jury de thèse. Merci à **Hervé Debar** et **Olivier Festor** pour avoir été les examinateurs de mon manuscrit. J'ai beaucoup apprécié découvrir, à travers vos rapports, un regard neuf sur ce travail que j'ai porté quotidiennement pendant des années. Merci à **Sophie Pinchinat** pour avoir présidé mon jury de thèse. J'ai apprécié chaque minute de la soutenance et cela est notamment dû à ton organisation fluide de ce moment important. Merci à **Rémi Badonnel** et **Eddy Caron** pour avoir participé à mon comité de suivi tout le long de la thèse. Discuter avec vous chaque année fut un plaisir. Merci à **Anaël Beaugnon** pour les nombreux échanges scientifiques que nous avons eus de 2018 à 2021. J'ai beaucoup appris grâce à toi et je te souhaite beaucoup de succès dans tes nouvelles aventures professionnelles. Lastly, I would like to thank **Sasha Romanosky** for inviting me to present my work at the EPSS workgroup. Later I was very honored when you accepted to be part of my thesis jury. Hervé, Olivier, Sophie, Rémi, Anaël, and Sasha, I wish you the best for the future and I hope our paths will cross

again.

While I'm still writing in English, I would like to thank past and present members of the **Myriads** team at Inria, including **Ali**, **Angélique**, **Anne-Cécile**, **Benjamin**, **Caroline**, **Cédric**, **David**, **Dorra**, **Ehsan**, **Genc**, **Guillaume**, **Javier**, **Kamesh**, **Loic**, **Marin**, **Martin**, **Maryse**, **Matthieu**, **Mozhdeh**, **Nikos**, **Pascal**, **Paulo**, and **Yasmina**. A special thanks to **Amir** for introducing me to everybody on my first day, and for the years of camaraderie after that. I wish you all a very bright future.

Pour terminer ces remerciements professionnels, je voudrais remercier toutes les personnes qui ont donné de leur temps pour les expérimentations décrites dans le chapitre 6 de ce manuscrit, ainsi que toute l'équipe de **Grid'5000**.

Je voudrais maintenant adresser quelques mots de remerciements à mes proches.

Merci d'abord à ma compagne **Lila**, pour ton amour et ton soutien. Cette thèse, c'est aussi la tienne à bien des égards. Parmi d'autres choses, je voudrais te remercier du temps que tu m'as dégagé pour terminer mon article à la conférence ARES durant le mois d'avril 2020. Cette publication (chapitre 5 de ce manuscrit) fut l'une des plus belle réussites de cette thèse, et la soumettre en plein contexte pandémique avec un jeune enfant à la maison n'a été possible que grâce au soutien total que tu m'as témoigné. Je t'en suis redevable.

Merci à ma mère **Anne** et ma sœur **Nina** pour votre amour indéfectible (et votre super chanson !). Une pensée bien sûr pour mon père **Gilles**. Le temps passe depuis que tu nous as quittés, mais il parait que tu aurais adoré m'appeler « Docteur Clément ». Merci à ma belle-mère **Michèle**. Et bien sûr, à tout le reste de la famille, les petits comme les grands. Je vous embrasse fort.

Merci à tous les amis que j'ai la chance d'avoir dans ma vie. En particulier **Florian** et **Floriane** bien sûr, mais aussi **Gaël**, **Fanny** et vos enfants, **Arnaud**, **Azéline**, **Benjamin**, **Mistral**, **Doriane**, **Damien**, **Marie-Sophie**, **France**, **Jessica**, **Mathieu**, **Anne-Catherine**.

Pour terminer ces remerciements, je voudrais rendre hommage à **Gaël Sourimant**, ami et lui aussi Docteur Rennes 1 passé par Inria, décédé en août 2019.

Table of Contents

Chapter 1

Introduction

This thesis is about defending *information systems* against *n-day vulnerabilities*. In Section 1.1 we establish how n-day vulnerabilities create a risk for information systems. In Section 1.2 we discuss the defenses currently used against them, and how they are inadequate in a lot of cases. We present our thesis objective in Section 1.3, our contributions in Section 1.4, and the outline of this document in Section 1.5.

1.1 Specific Risks Created By N-Day Vulnerabilities for Information Systems

In this section we define what is an information system in a modern sense, what are n-day vulnerabilities, and highlight the risk they create against information systems. To this end we discuss the characteristics of modern information systems in Section 1.1.1. In Section 1.1.2 we show how high the stakes of information security have become in the last decade. In Section 1.1.3 we discuss a specific part of the software and hardware vulnerability lifecycle: newly disclosed vulnerabilities or n-day. In Section 1.1.4 we discuss three past n-day vulnerabilities that had wide real-world impact. In Section 1.1.5 we highlight that this real-word impact requires information security to be a continuous activity.

1.1.1 The Modern Information System

Picolli et al define an information system as "*a formal, sociotechnical, organizational system designed to collect, process, store, and distribute information*" [155]. Another way to formulate this is to describe information systems as comprising the people, hardware, software, and organizational processes allowing an organization to make decisions. While information systems are older than computer science, modern information systems have several recurring characteristics. First, they have *multiple stakeholders*. The organization

using the information system is not necessarily the same as the organization maintaining it, the one securing it, or the one developing the software the information system is made of. Second, modern information systems are *composite*: they use heterogeneous software and hardware provided by many vendors. Some of this software is public, some of it is open-source, some of it is neither. Third, many modern information systems are *complex systems*, for which exhaustively charting their numerous components and the interaction between them is challenging.

1.1.2 The Rise of Cyber-Insecurity

Cyber-security is one of the most important technical and societal topic of the 21st century. The more domains are automated through software, the more impactful information security becomes. All observable metrics point towards the same conclusion: the stakes are becoming higher every day. The number of known software and hardware vulnerabilities, the number of recorded cyber-attacks, the cyber-defense investments by public and private organizations have all been growing for over a decade [128, 160, 200].

In the last decade two trends contributed to make cyber-attacks considerably more dangerous than before. The first is the rise of *cyber-warfare*, with many nations developing cyber-offense capabilities in order to target foreign governments and private organizations for geopolitical reasons. Some of these state-sponsored attacks cost billions of dollars in damage to their victims [205], and some led to lethal retaliation [101]. The second trend is the rise of financially motivated cyber-attacks by the organized crime, through the use of new types of malware such as *ransomware* [234] and *cryptominers* [222]. Ransomware in particular, which encrypt files on infected computers to ask a ransom in exchange for the decryption key, have already cost billions of dollars in damage to thousands of affected organizations across the world, and recently led to a fatal casualty following the paralysis of a German hospital [159].

1.1.3 The Double-Edged Sword of Reuse: the N-Day Vulnerability

As said in Section 1.1.1, information systems are *composite*: while some of their components might be custom-made, most are on-the-shelf software and hardware products. While beneficial on many aspects, one overlooked drawback of this fact is the opportunity

for attackers to compromise an information system by finding and exploiting vulnerabilities in the public software and hardware the system is made of.

Vulnerabilities in public software and hardware follow a *life cycle*. They are initially introduced through a design or implementation error, then stay *undiscovered* for some time. Once a vulnerability has been privately discovered, it becomes a *zero-day* vulnerability. In some cases the discoverers might keep the vulnerability existence to themselves, but in other cases they will coordinate a *disclosure process*, making the vulnerability eventually *public*. However, a vulnerability disclosure increases the short-term risk posed by the vulnerability, as it becomes a new tool in the arsenal of attackers while organizations may not have had time to mitigate the vulnerability yet. We call *n-day* recently disclosed vulnerabilities that are still in this critical phase of their life cycle. Once the vulnerability becomes common knowledge for both attackers and defenders, it becomes a *well-known* vulnerability and the associated risk diminishes.

1.1.4 Seminal N-Day Vulnerabilities

Heartbleed, *Shellshock*, and *EternalBlue* are three infamous n-day vulnerabilities that had wide real-world consequences.

Heartbleed was an OpenSSL vulnerability that let attackers obtain the SSL private keys of most public-facing websites on the Internet. It required immediate action for thousands of system administrators in order to secure their systems. A study of the timeline and impact of Heartbleed can be found in Appendix A.1.

Shellshock was a bash vulnerability affecting every bash shell installation from 1989 to 2014. It could be exploited remotely through CGI scripts, and was so easy to use that attacks were recorded within an hour of disclosure. It led to distributed denial of service attacks against the Akamai CDN and the American Department of Defense (DoD) networks. A study of Shellshock can be found in Appendix A.2.

EternalBlue was a confidential NSA software exploiting a vulnerability in the Microsoft implementation of the SMB protocol. This vulnerability affected all versions of Windows from Windows 3.1 to Windows 10 and could be exploited remotely. The public leak of EternalBlue led to the launch of the *WannaCry* ransomware attack and the *NotPetya* cyber-warfare attack against Ukraine, which both created multiple billions of dollars in damage and paralyzed many hospitals across the world, requiring patients to be diverted to other destinations. A study of EternalBlue can be found in Appendix A.3.

These examples show that the risk created by n-day vulnerabilities is not theoretical.

Some future n-day vulnerabilities are likely to have similar or even higher real-world impact. As dozens of vulnerabilities are disclosed every day, they add a risk factor to information systems that is *dynamic over time*.

1.1.5 Information Security over Time

An information system can have vastly different risk profiles at two distinct points in time, even if no modifications have been applied to it in the interval. The system might be adequately protected against all publicly known vulnerabilities at some point, then becomes at risk because of a dangerous vulnerability that has just been disclosed and is not yet mitigated.

This means that information system security is not a one-time activity: security risk must be measured continuously and defensive actions must be taken regularly for information systems to stay secure.

1.2 Motivation

The previous section established the risk posed by n-day vulnerabilities against information systems. In this section we justify our belief that further research is needed to adequately defend against them. In Section 1.2.1 we highlight that n-day vulnerabilities are the product of trends that will not go away in the future. In Section 1.2.2 we review the current security practices to defend against n-day vulnerabilities and show why they are inadequate. In Section 1.2.3 we review the open challenges of defense against n-day vulnerabilities.

1.2.1 N-Day Vulnerabilities are Here to Stay

The real-world impact of n-day vulnerabilities comes from three trends: software reuse, public disclosure of vulnerabilities in public software and hardware, and usage of software in an ever-growing number of use cases, including life-critical contexts. None of these trends seem likely to lose relevance in the short and medium term. The software engineering community widely acknowledges software reuse as a best-practice yielding many benefits, particularly through free and open-source software. Public and coordinated vulnerability disclosure is considered a best-practice by the security community, for reasons we explain in Section 2.1.2. Last, incorporating software and network connectivity in new

objects and places is a tenet of multiple billion dollar industries, such as the *Internet of Things* (IoT). This means it is very likely that the risk posed by n-day vulnerabilities will not diminish in the future and is likely to grow.

1.2.2 Current Security Practices are Inadequate To Defend Against N-Day Vulnerabilities

Having established that n-day vulnerabilities will likely become more and more problematic as time goes on, we now review current defenses against them and highlight their limitations. These defenses are studied further in depth in Section 2.3.

Software Updates

Keeping software up-to-date is a vital step of any n-day mitigation strategy. However, a patch might not be available in the first minutes or hours following a vulnerability disclosure. Moreover, a software update might be available, but also contain a bug or a backward compatibility problem preventing a timely patch deployment. As critical n-day vulnerabilities such as Shellshock were exploited within one hour of disclosure, diligent software updates are certainly necessary but not sufficient.

Signature-based Intrusion Detection Systems

Intrusion Detection Systems (IDSs) monitor computing activity and attempt to separate *malicious* from *legitimate* behavior. Multiple approaches exist for intrusion detection (which we detail in Section 2.3.3) but at least one variant already had a large impact against n-day vulnerabilities: *Web Application Firewalls* (WAFs) is a cloud industry term to designate *Network-based*, *signature-based*, Intrusion *Prevention* Systems. A cloud provider providing a WAF offering monitors incoming traffic on behalf of its clients, and blocks traffic matching *signatures* manually crafted to detect malicious activity.

While very useful to defend against known vulnerabilities, WAFs and other signature-based techniques are not well suited for defense against n-day vulnerabilities. Creating a detection signature for attacks exploiting a vulnerability requires time, effort, expertise and a deep knowledge of the targeted vulnerability. It is therefore very difficult to author a signature in the first hours after disclosure in a systematic way for all vulnerabilities.

Security by Design and Defense in Depth

Security by design is a broad class of software design and implementation techniques aiming to incorporate security as a first-class requirement from the inception of a software project onward. While substantial security gains can be made through these approaches, they are also inadequate for defending entire information systems against n-day vulnerabilities for two reasons. First, these approaches aim to limit the number of vulnerabilities in a piece of software, but cannot claim to remove them completely. Second, information systems are composite. While newer components of a system may have been built with sound security principles in mind, the whole system is very likely to include legacy hardware and software components that are not as secure.

It is possible to design an entire information system using security by design practices, and making it more secure than its individual and possibly insecure components. To this end, a widely used defense pattern for designing security-critical information systems is the concept of *defense in depth*. Defense in depth is the practice of relying on multiple defense mechanisms simultaneously, so that when one layer fails at mitigating an attack, other layers might succeed. For n-day vulnerabilities mitigation, a defense in depth strategy could include a timely software update deployment *and* the use of intrusion detection techniques *and* the use of secure-by-design components. Defense in depth is a crucial part of many defense strategies, but it has limits. In particular, vulnerability disclosures are public events, informing both the attacker and the defender that a layer of the defense can be bypassed. However, a persistent attacker may have secretly found a way to bypass all the remaining layers, leaving the system briefly vulnerable without the defender knowing it.

Air Gap

An *air gap* is the practice of physically isolating an information system from the outside, including removing any network link with the global Internet. This is a common practice for sensitive networks of many countries national armed forces and intelligence agencies. While certainly useful for defending against n-day vulnerabilities, trading internet access for security is a compromise most organizations are not ready to make.

Moreover, there have been cases of air gap networks being breached by highly motivated attackers. The most spectacular example is Stuxnet, a joint US and Israeli cyberwarfare operation that included a self-replicating worm corrupting USB flash drives, en-

abling it to break into an air gap network of the Iranian nuclear program, eventually destroying many Iranian nuclear centrifuges [111].

1.2.3 Challenges when Defending Against N-Day Vulnerabilities

Defense against n-day vulnerabilities is a *race against the clock*: acting slower than attackers is dangerous. But so is acting hastily, as an improper mitigation can be incomplete or even damage the information system. Mitigating n-day vulnerabilities safely yet timely is preferable, but more expensive. As proactive mitigation techniques, such as security by design and defense in depth, are necessary but not always sufficient, organizations considering the security of their information systems as critical look for shorter and shorter reactive mitigation times. However, all current reactive mitigation techniques are limited by human reaction times, such as patch development by a software vendor or the authoring of an IDS signature by security experts. Therefore a promising way to significantly improve the security of information systems against n-day vulnerabilities is to propose novel reactive mitigation techniques that do not require human intervention.

1.3 Thesis Objective

The goal of this thesis is to propose novel tools to automatically analyze and react to the risk posed by n-day vulnerabilities at disclosure. Our ultimate goal is for an information system to exhibit *constant security over time*, with daily vulnerability disclosures not affecting the overall risk it faces.

A more concrete objective is to automatically and immediately gather, for any newly disclosed vulnerability, information that is usually gathered by human experts after several days. In the first moments after a vulnerability disclosure, a lot of information about the vulnerability is either missing, fragmented, or not machine-readable. This includes intrinsic information about a vulnerability such as the software or hardware it affects, which is provided as machine-readable data days or weeks after disclosure, delaying the use of any vulnerability management process. It also includes the severity analysis of the vulnerability, usually provided in the form of a CVSS vector [42], which encodes this analysis in a machine-readable format but is not available at disclosure either. CVSS vectors are authored manually by security experts days or weeks after disclosure, again delaying

the vulnerability management processes. Last, a context-specific analysis is also needed yet difficult to obtain promptly. This includes how much risk a vulnerability disclosure creates for a specific information system.

Gathering all of this data automatically is a necessary condition for faster vulnerability management, yet automation is always a risk of becoming a black box that cannot be explained by anyone. We argue that automated defense systems can only be trusted if they can be understood and that decision explicability should be a cornerstone of any automated security process we devise.

1.4 Contributions

In this thesis we present four contributions to reach our objectives:

— A proposal for an end-to-end defense strategy against n-day vulnerabilities. This strategy consists in gathering automatically more information about a vulnerability in seconds after its disclosure, performing an automated risk analysis of the vulnerability in the context of a specific information system, then if warranted, triggering an automated reaction mitigating the risk created by the vulnerability. All our other contributions implement parts of this overall strategy, in an explicable way.

— We propose a process enabling the extraction of the name of the affected component of a vulnerability immediately at disclosure. To our knowledge this is the first one to be both automated and explicable.

— We propose a prediction process for vulnerability CVSS vectors, that can be used immediately after disclosure. To our knowledge this is the first one to be both automated and explicable.

— Building on top of these contributions, we propose a risk analysis process for newly disclosed vulnerabilities in the context of a specific information system. It uses active learning to extract the conscious and unconscious knowledge of the security team overseeing the information system to decide the appropriate alert level for a new vulnerability disclosure in the context of the defended system. It is a preliminary yet promising step towards automated reaction mitigating new vulnerability disclosures. To our knowledge this is the first process of its kind to be both automated and explicable.

1.5 Thesis Outline

In Chapter 2, we present the state of the art in the research and industry fields we build upon: Security as a Service (SECaaS) and why it is important in the context of modern, multi-stakeholders information systems, the life cycle of software and hardware vulnerabilities, n-day vulnerabilities and the current defenses against them, and machine learning for security alerting which is an important tool to fulfill our objectives.

In Chapter 3, we outline the global picture of this thesis, and present our first contribution: an end-to-end strategy for defending information systems against n-day vulnerabilities. We lay out how all our contributions come together towards fulfilling this strategy.

In Chapter 4, we present our second contribution: automatically determining the software or hardware affected by a new vulnerability using only the free-form text description available at disclosure.

In Chapter 5, we present our third contribution: assessing the severity of a vulnerability at disclosure through automated prediction of its CVSS vector based on the vulnerability text description.

In Chapter 6, we present our fourth contribution: automating the decision process of raising an alert or not regarding a new vulnerability disclosure in the context of a specific information system.

In Chapter 7 we conclude, and present our perspectives for the short, medium, and long term.

Chapter 2

State of the Art

In this chapter, we establish the state of the art in four domains this thesis builds upon. In Section 2.1, we explore the security of information systems and the concept of security as a service. In Section 2.2, we review the life cycle of software and hardware vulnerabilities. In Section 2.3, we study a specific part in this life cycle: *n-day vulnerabilities*. In Section 2.4, we explore machine learning for security alerting, which is an important tool used in our contributions. We conclude in Section 2.5.

2.1 Security as a Service

In this section, we explore how information system security practices evolved (Section 2.1.1) to acknowledge that security risk evolves over time (Section 2.1.2). We study how this led to the trends of *security as a service* and information security outsourcing (Section 2.1.3).

2.1.1 Information Systems and their Stakeholders

Information systems have been studied and implemented for a long time in both industry and academic research.

Definition 1 *Information systems comprise the people, hardware, software, and organizational processes allowing an organization to make decisions.*

As soon as 1991, it was known that information systems are inherently multi-stakeholders systems [170]. These stakeholders are often divided between the following parties:

Definition 2 *The owner of an information system is a physical or legal person legally responsible for the proper functioning of the system.*

Definition 3 *The users of an information system interact with the system day to day to accomplish their daily tasks.*

Definition 4 *The developers of an information system build and modify the system.*

Definition 5 *The administrators of an information system ensure the system is working as intended on a day to day basis.*

The taxonomy may vary: for instance, the advent of *cloud computing* divided the administration of many information systems between two parties: the information system administrators keep the responsibility for the application layer of the system, while the cloud provider now usually takes the responsibility for the technical infrastructure underneath. It also implies that many core components of information systems are not located within an organization boundaries anymore, but in a cloud provider infrastructure instead.

Going back to 1991, it was also known that verifying that an information system is working properly over time is not trivial, nor is it to prove it to all stakeholders. *Service-Level Agreements* (SLA) have been shown to be a relevant tool for contractualizing what is an acceptable level of operation for a given system [167].

Definition 6 *A Service-Level Agreement or SLA is a contract between a service provider and a client to ensure that a service is provided as intended. An SLA is based on an observable metric, an acceptable threshold for this metric, a time period, and a penalty (financial or otherwise) for the service provider which occurs if the metric crosses the threshold during the time period.*

When implementing an SLA a stakeholder (such as the information system owner) becomes a client and another stakeholder (such as a system administration company) becomes a service provider. A typical metric is the amount of uptime of a computer machine: typical thresholds can be 99 % (7 hours of allowed downtime per month for the machine), 99.9 % (44 minutes), or 99.99 % (4 minutes).

Over the last decades, many processes related to information systems turned out to be better handled continuously than as one-off tasks. For instance, it is worthwhile to note that information system development and planning mostly went from static *waterfall* models to dynamic *agile* methods, with the system design never being frozen.

Information Security has been gradually accepted as a cornerstone of a well functioning information system. Just as its development or its administration, security of an information system is best viewed as a continuous process with multiple stakeholders, as seen in the next sections.

2.1.2 Evolution of Security Risk over Time

In this section, we examine the difference between proactive and reactive security, and the causes leading the risk profile of an information system to change over time.

Proactive Security vs Reactive Security

There is an ongoing debate among the security community about whether information security should be *proactive* or *reactive*. Proactive security considers security as a paramount quality of systems, software and hardware from their inception, resulting in architectures that are deemed *secure by design*. Reactive security, on the other hand, takes a system, software or hardware as it currently is and only reacts to actual security incidents, fixing them as they occur.

It is indisputable that proactive security efforts allowed the software community to dwarf entire classes of security problems. For instance, Microsoft has disclosed that 70 % of vulnerabilities they fix in their products are due to memory-safety problem [119], a class of bugs that has been fixed in modern, memory-safe languages such as Java, Go or Rust.

However, reactive security is still necessary for several reasons. First, software reuse is a cornerstone of modern software development practices. While this has a lot of benefits, any security problem in a software dependency of a system (directly or transitively) becomes a security problem in the system itself [47]. The massive increase in open-source software in organizations [190] allows both attackers and defenders of a system to study its source code, for opposite motivations. Second, proactive security does not offer guarantees about new attack classes that were not part of the original security model. For instance, side-channel attacks on software/hardware interfaces such as Spectre [109] or Meltdown [115] were not anticipated in most otherwise secure software architectures.

Reactive security is here to stay. Some even theorize that under certain conditions, reactive security is an optimal game-theoretical strategy as it allows defenders to focus their efforts on the areas most susceptible to attacks [15].

Causes of Risk Evolution over Time

There are two main causes of change for the risk profile of an information system over time. First, attackers motivations can change. Systems that have been secure to past threats can become overwhelmed by more competent (or more numerous) attackers. A

common reason for attacker motivation evolution is geopolitics. For example, the 2007 tensions between Estonia and Russia around the relocation of a Soviet-era statue in the city of Tallinn led to an unprecedented wave of cyberattacks targeting both the Estonian governement and private organizations, paralyzing parts of the country for 22 days [151].

There have been proposals, such as Caldara et al's [30], for a global geopolitical risk index aiming to help both public and private sectors detect heightened periods of risk for their countries or organizations because of recent world events.

While the evolution of attackers motivations (for both geopolitical and non-geopolitical reasons) is an important part of the evolution of security risk over time, we chose not to focus on this topic in this thesis.

Second, latent vulnerabilities in existing software are routinely discovered and disclosed publicly. As most information systems are composed of myriads of software and hardware components, a vulnerability in any of these components can create a security risk for the whole system. While the vulnerability has been there since the deployment of the vulnerable component, before disclosure the vulnerability is only known to a limited number of people (possibly zero). The vulnerability disclosure informs all parties, benevolent or malicious, of the presence of the vulnerability, creating additional risk for all vulnerable systems.

This is a major topic of this thesis and we investigate this topic further in depth in Section 2.2.

2.1.3 Organizations and their Security Policies

In this section, we review how organizations adapted to the dynamicity of security risk, through the creation of recurring security services. We then highlight that on the one hand these services are currently too expensive to operate internally for the majority of organizations needing them, but on the other hand outsourcing security to a third-party has unique challenges that are still unsolved.

Recurring Security Services

The evolution of cybersecurity threats led organizations to pursue continuous protection through the creation of *Security Operation Centers* (SOCs) dedicated to ensuring their ongoing safety.

SOCs aim to protect organizations and information from cyber attacks in real time,

through collection, gathering, correlation and analysis of numerous real-time, heterogeneous information sources regarding the system to be protected [169]. These information sources can include host or network logs, IDSs or antivirus alerts, vulnerability disclosures, and many others.

In practice, even though a SOC might be partially automated, human *security analysts* still play a critical role in attack detection and mitigation. A lot of SOCs employ multiple analysts who work in a 24/7 rotation from a dedicated secure room [191].

Most Organizations Cannot Afford Real-Time Defense

Ideally, every organization would benefit from being protected by a SOC, either internal or outsourced. However, this is a challenging goal in practice.

Creating an internal SOC is difficult because security analysts are expensive and in a short supply. According to [200] the median US salary for analysts was $76,000 per year in 2019, with the 9th decile at $117,000 per year. Meanwhile, the cybersecurity unemployment rate has stayed at zero per cent since 2011 and a shortage of 3.5 millions unfulfilled cybersecurity jobs has been predicted for 2021 [202]. This makes the goal of hiring cyber analysts for an internal SOC out of reach for many organizations.

An organization could try to mitigate this problem through contracting protection from an *outsourced SOC*. In this configuration, an organization forwards all its real-time security data to a dedicated cybersecurity organization which protects multiple clients at the same time. Outsourcing a SOC does avoid upfront hiring costs for an organization, and it lets security analysts protect multiple organizations at a time, allowing for more expertise and specialization. However, it does so at the expense of analysts having an intimate familiarity with the protected system and imply storing sensitive security data outside the organization [169]. Moreover an outsourced SOC still faces the global shortage of security analysts and this reflects in the price it charges its clients. Finally, making security analysts monitor multiple organizations has limits as there is a documented history of elevated stress and burnout in SOC analysts [62, 191, 192, 145]. In particular the US Air Force studied the stress and burnout risks of 500 members of its cyberwarfare workforce and found higher stress rates among cyberwarfare workers than other positions inside the US Air Force [34].

Therefore, at a global level it is neither possible to hire more security analysts nor to give them much more work to do. These facts taken together mean the human component of cybersecurity defense is globally saturated, and the only way to meet the ever-increasing

demand for real-time cybersecurity defense is through automation.

However automation also has challenges. First, many SOCs have idiosyncratic processes that are formed around the available data sources for the entity to protect. A lot of analysts have trouble integrating on-the-shelf solutions into their daily analysis workflows [191], often requiring automation to be tailored for a specific SOC. Second, even though automating low-level tasks does work well, automating higher-level security analysis tasks has proven more challenging. As we see in Section 2.4, machine learning for security alerting exhibits a number of challenges that are exacerbated the more complex and critical the decision becomes. SOCs analysts tend to work in a hierarchy [192] with many entry-level analysts analyzing the bulk of potential alerts then escalating potential risks to higher-level, more experienced analysts for deeper analysis. While automation lets us consider automating part of the workload of the cheaper, easier to hire entry-level analysts, it leaves the workload of the more experienced and expensive higher-level analysts intact.

Nevertheless both outsourcing and automation are part of the solution to affordable real-time cybersecurity defense for all organizations.

Outsourcing and Security

There is a fundamental problem with outsourcing security critical parts of an information system, such as outsourcing a SOC or simply deploying an information system in a cloud provider infrastructure: it is difficult to prove to both parties that the resulting system is secure. The service provider can hide or even ignore that its infrastructure is vulnerable to attacks. A successful covert attack can impact confidentiality or integrity of a system without neither the service provider nor the client noticing it.

Security researchers, security practitioners, governments and hardware manufacturers all tried to solve this issue through different means, arguably all unsuccessfully.

A part of the security research community spent the last decade trying to pinpoint the concept of *Security SLA*, or Sec-SLA, especially in the context of cloud computing [35, 154, 199, 209]. The concept of a sec-SLA is to take the components of an SLA (a time period, a metric, a threshold, and a penalty) and apply it to the security properties of the system. While such a system can be part of a solution, it cannot be complete as important security properties of a system are unobservable by the client (for example, the physical security of a cloud provider's datacenters), and some of these properties are unobservable by neither the client nor the service provider (for example, the presence of undiscovered

zero-day vulnerabilities in the technical infrastructure of the cloud provider).

Meanwhile, most of the security industry denied the problem, resulting in a phenomenon called *security theater* [175]. Security theater is a situation where measures are applied with the goal of providing the *feeling* of security instead of *actual* security. In particular, multiple SOC analysts have expressed frustration at their hierarchy being seemingly more focused on justifying the SOC budget to decision-makers than making the protected systems actually safer [192].

Governments approached the problem through regulation and legislation of sensitive digital services. For instance, credit card information manipulation is regulated using the PCI DSS standard [153] world wide, while the European Union regulates electronic identification and transactions through the eIDAS regulation [59]. These regulations can include thorough *obligation of means*, usually evaluated using organizational and technical audits. However they do not require any *obligation of results*. While these regulations are effective at filtering out careless actors, they do not prove that a qualified actor is really secure.

Hardware manufacturer Intel proposed a technical solution named *Software Guard Extensions* (Intel SGX) [99], which introduces the concept of *secure enclave*. Once an enclave is created through a specific set of CPU instructions, the code and data residing in an enclave stay encrypted and cannot be read or modified by the rest of the system, including the OS. The CPU can sign a *proof of enclave* allowing two enclaves on two separate machines to acknowledge the existence of each other. SGX is managed through a public key infrastructure whose root of trust is Intel. However SGX is only useful in the rather peculiar threat model where a client would fully trust Intel but not its own service provider. Moreover, SGX has a history of breaches [177, 82, 25, 28, 176] that makes such a trust in SGX arguably misplaced.

In the future, a possible way out of this problem could come from cryptography, namely *homomorphic encryption*, first described by Rivest et al in 1978 [165] and first successfully implemented in 2009 by Gentry et al [76]. Homomorphic encryption allows an untrusted party to achieve computation on data that is *still encrypted*, on behalf of another party that *did not disclose the decryption keys*. For instance, this allows a client to encrypt her sensitive data, send the encrypted version of the data to a cloud provider that would then make computations on the data without being able to decrypt it at all. However, homomorphic encryption comes at a tremendous performance cost: current state of the art techniques [36] evaluating less than a hundred binary gates (NAND, XOR,

AND, etc.) per second per CPU core, a slow-down of approximate magnitude 10^7. This performance cost makes this approach currently unpractical for general computing tasks. Nevertheless a homomorphic encryption scheme with acceptable performance would allow a breakthrough in secure computation outsourcing.

2.2 The Vulnerability Life Cycle

The software vulnerability life cycle is an important source of security risk evolution over time and a major topic of this thesis. In Section 2.2.1 we propose a fictional example of a typical vulnerability life cycle in order to get more familiar with the related concepts. In Section 2.2.2 we describe the data sources and frameworks used in the research and industry security communities to model the life cycle of software vulnerabilities. In Section 2.2.3 we study the security discoveries made by the security community about the life cycle of software vulnerabilities at scale.

2.2.1 A Typical Vulnerability Life

CVE-PLATYPUS-0001: Story of a Fictional Vulnerability

For a brief time let us forget about the physical computers of the real world and venture into the fictional, legendary Kingdom of the Mighty Platypus.

In the Kingdom of the Mighty Platypus, plebians are very keen in keeping their treasures secure and they sleep better at night when these treasures are stored into a personal safe box. However they are aware that the security of their treasures can only be as good as the security of their safe, and they are always on the look for the best safe manufacturer in the Kingdom.

It happens that the best safes in the Kingdom are made by a brave smith named Smith. Smith the smith has the reputation of remedying any weakness in his safes timely and thoroughly. Alas, he is only one man and the Kingdom is vast.

After years of hard work, Smith's reputation now precedes him and most of the plebians are equipped with his safes, even the ones living far away from the small town where Smith resides.

It is in one of these remote parts of the Kingdom that a vile thief named Thierry made a terrible discovery: every safe ever manufactured by Smith the smith could be opened

without the key by knocking three times and a half on the right panel of the safe, then pronouncing loudly the word "polyglot".

Thierry the thief realized he had a very valuable secret in his hands. During the next few weeks he proceeded to discreetly dispossess a few wealthy citizens of their belongings, but remained tight-lipped about the mean he used to break into the safes.

But even the vilest thief in the Kingdom makes a mistake sometimes. On a night with the fullest moon the Kingdom had ever seen, Thierry the thief was busy with his larceny du jour and did not notice how the moon was shedding light on all his very moves.

From the opposite side of the alley, a virtuous plebian named Virgil was watching and listening. Fearing for his life, he let Thierry the thief go away with his loot without bothering him. But Virgil the virtuous did learn Thierry's secret, and after trying to knock three times and a half on the right panel of his own safe while saying "polyglot", he realized he too could open any safe he wanted. But Virgil the virtuous was virtuous, and instead of using the finding to live a life of sin, he decided to embark on a journey to find Smith the smith who only could devise a solution to repair all the safes in the Kingdom of the Mighty Platypus.

But the journey to Smith the smith was long. After a long walk throughout the Kingdom, Virgil the virtuous became thirsty and did a well-deserved stop at a tavern half-way. There he fraternized with some fine folks, and in a lapse of judgment (possibly induced by alcohol), he revealed to everybody in the tavern the hidden motive behind his adventure. He then paid for everybody's drinks and proceeded to complete his expedition.

Smith the smith welcomed the news of the safe's weakness with stoicism and a sense of urgency. After two days and one sleepless night of tireless work, he had a breakthrough: by putting a leaf of sarsaparilla on the inner right side of the safe while pronouncing the word "salted butter caramel", the safes would be safe.

Sarsaparilla was abundant in the Kingdom of the Mighty Platypus. This made for an easy repair, and Smith the smith and Virgil the virtuous started spreading the word around them. Soon half the Kingdom was frantically looking for sarsaparilla and pronouncing "salted butter caramel !" repeatedly, repairing many safes along the way.

However, back in the tavern a folk had told his mother who told her nephew who told his neighbor who told his father-in-law about how to break into the safes, and soon enough all the thieves in the Kingdom, from the best to the worst, started knocking three times and a half while shouting "Polyglot!" at every safe they could put their hands on.

For some time, chaos ensued. In parts of the Kingdom that first learnt about the

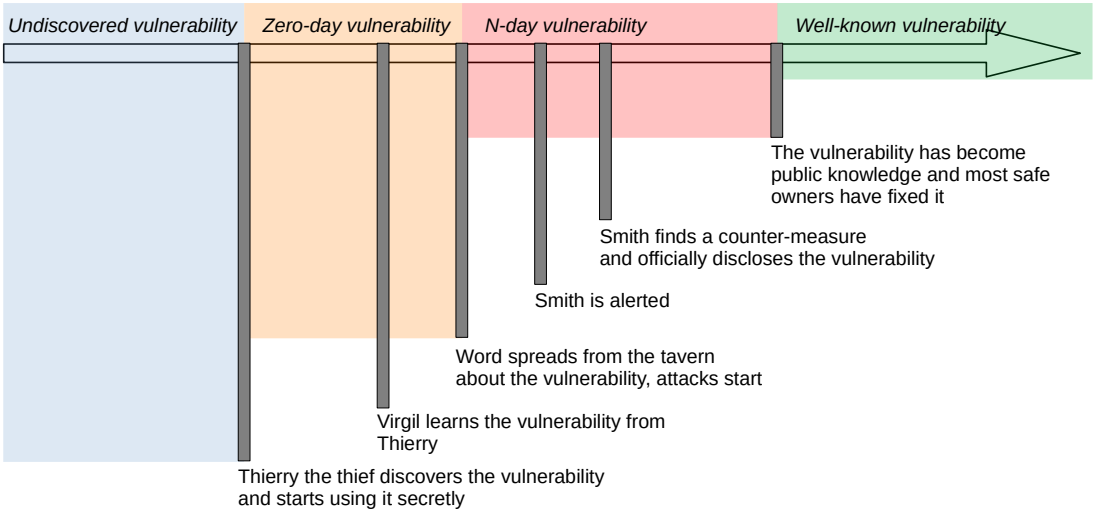

Figure 2.1 – The life cycle of the fictional vulnerability CVE-PLATYPUS-0001.

sarsaparilla, order prevailed. Thieves came and went frustrated, as knocking and shouting "polyglot" had no effect whatsoever. However in other parts of the kingdom whole villages were looted because honest folks did not learn about the weakness in time to fix it.

After some time, almost everybody in the Kingdom had learnt both how to break into vulnerable safes, and how to repair their own. A few careless folks did neglect to ensure their safes were safe, to the delight of the mischievous kids in their neighborhood. But all in all, peace was restored in the land of the Mighty Platypus.

CVE-PLATYPUS-0001: A Fictional But All Too Real Vulnerability

Let us now go back to our real world and computer systems. While the Kingdom of the Mighty Platypus might be imaginary, the outline of the events depicted here is not.

The safe built by Smith the smith is a *component* and the fact that it could be opened without the key is a *vulnerability*, which we call *CVE-PLATYPUS-0001* from now on. Fixing the problem through the use of sarsaparilla is a *counter-measure*.

Definition 7 *A software or hardware vulnerability is a bug or defect in the design or implementation of a software or hardware component, creating an unforeseen security issue.*

Definition 8 *The exploitation of a vulnerability is the act of using the vulnerability to compromise a target.*

Definition 9 *A counter-measure is a course of action preventing a vulnerability from being exploited.*

Before discovery the vulnerability was already present as an *undiscovered vulnerability*. Once Thierry the thief became aware of it and started using it secretly, CVE-PLATYPUS-0001 became a *zero-day* vulnerability *exploited in the wild*.

Definition 10 *An undiscovered vulnerability is an existing vulnerability that no one is aware of, neither secretly nor publicly.*

Definition 11 *A zero-day vulnerability is secretly known to a limited number of parties but unknown to the general public.*

Definition 12 *A vulnerability exploited in the wild is actively used by attackers to compromise unwilling targets.*

Once the word started to spread, the vulnerability was *disclosed* and a race against the clock started to emerge between the *exploitation* and *mitigation* of the vulnerability. During this troublesome period just after disclosure, the vulnerability was a *n-day* vulnerability.

Definition 13 *An n-day vulnerability is a de facto public vulnerability (officially disclosed or not) that not all affected parties are familiar with yet and not all are ready to defend against, creating a heightened risk for a temporary period of time.*

Definition 14 *Mitigation is reducing the risk induced by a vulnerability to a negligible amount. This can include deploying a counter-measure or simply verifying that a vulnerability is not exploitable in practice.*

Definition 15 *The disclosure of a vulnerability is the event marking the transition of the vulnerability from secret to public knowledge, in other words from zero-day to n-day.*

Once both the vulnerability and its counter-measure become common knowledge and things settle, the vulnerability becomes a *well-known* vulnerability. The *life cycle* of this fictional vulnerability is summarized in Figure 2.1.

Definition 16 *A well-known vulnerability is a public vulnerability that the vast majority of affected parties are familiar with and ready to defend against.*

Definition 17 *The life cycle of a vulnerability comprises all the stages of a vulnerability timeline: from being introduced as an undiscovered vulnerability, to becoming a zero-day vulnerability once first discovered, then an n-day vulnerability just after its disclosure, then a well-known vulnerability once firmly in the public knowledge.*

CVE-PLATYPUS-0001 shares many similarities with real-world vulnerabilites. Knocking three times and a half to open the safe is inspired by CVE-2017-13872 [143], an actual macOS High Sierra vulnerability that allowed attackers to log as root by leaving the password field blank then clicking on the login button multiple times [50] (however, there was no need to pronounce "polyglot" while doing so). The race between attacks and mitigation throughout the Kingdom is inspired by CVE-2014-0160 [140] also known as Heartbleed [61] and CVE-2014-6271 [141] also known as Shellshock [38], which we describe in detail in Appendix A.

Hardware vulnerabilities, such as Spectre[109] or Meltdown [115], work in a similar way as software vulnerabilities and in this thesis we treat them the same. They emerge because of a flaw in the design or manufacturing process of a hardware component, just as software vulnerabilities emerge because of a flaw in the design or implementation process of a software component.

It should be highlighted that the fact that a component is vulnerable or not is independent of the context in which the component is used (although a specific context may be required for the vulnerability to be *exploited*). In that regard, there is a distinction to be made between a *component* and a *system*. While the fictional safe made vulnerable by CVE-PLATYPUS-0001 is a physical component of limited value in itself, it is part of a larger system aiming to secure the belongings of many citizens. In this case, the safe is vulnerable no matter what is inside it, but whether a citizen's wealth is vulnerable or not is a matter of if and how the citizen is using the vulnerable safe. Likewise, a software or hardware vulnerability can lead to an *information system vulnerability* when the vulnerable component is used in a way where it can be exploited to compromise the entire information system.

Definition 18 *An information system vulnerability is a technical or organizational flaw allowing an information system to get compromised. It can emerge from a vulnerability*

affecting a piece of software or hardware used in the information system if the vulnerability can actually be exploited in the context of the information system. However information system vulnerabilities can also happen for other reasons, such as configuration errors or human negligence.

In this thesis, the word *vulnerability* always refers to software or hardware vulnerabilities. Information system vulnerabilities are specifically mentioned as such. Another distinction to be made is the difference between a vulnerability *severity*, the *threats* against an information system, and the *risk* an information system faces.

Definition 19 *The severity of a software or hardware vulnerability is a measurement of its potential impact, outside of a specific context.*

Definition 20 *The threats faced by an information system or an organization are the attackers attempting to compromise it, with varying degrees of skills, motivation, and resources.*

Definition 21 *The assets of an organization are the sensitive resources that the organization is willing to protect, possibly through the use of an information system. This can include proprietary data, monetary resources, means of production, and many others.*

Definition 22 *The risk faced by an information system is the likelihood of an asset in this information system to be compromised. This risk can be estimated through the combination of the threats faced by the system, the vulnerabilities it is affected by, and the value of the assets residing in the system.*

In the case of CVE-PLATYPUS-0001, the severity of the vulnerability is solely defined by the fact that an attacker can open the safe without the key and not the context in which the safe is used. The threats faced by a citizen are the thieves trying to dispossess them. The risk faced by a citizen is the combination of how much of a threat the thieves are, and whether the citizen is using a vulnerable safe to store important belongings.

The more vulnerable an information system is, the likelier it is for even a modest threat (such as low-skilled and lowly motivated attackers) to result in a compromise. Conversely, organizations facing strong threats can limit their risk by heavily investing in defense and counter-measures.

Vulnerability discoverers find and disclose dozens to hundreds of software vulnerabilities every day, some benign and some critical. This creates a lot of time-sensitive

information to analyze and act upon for information system administrators and SOC analysts. The security community has settled on a few de facto standards to present and organize software vulnerability intelligence, which we detail in the next section.

2.2.2 Data Models and Databases

In this section we present widely used databases and data formats related to the vulnerability life cycle: *CVE*, *NVD*, *CVSS*, *CPE*, and *CWE*.

CVE

The *Common Vulnerabilities and Exposures* (CVE) database is described as "*a list of entries —each containing an identification number, a description, and at least one public reference— for publicly known cybersecurity vulnerabilities*" [41]. Since its launch in 1999 CVE has become the de facto standard for public vulnerability disclosure. CVE assists in four important points of the security disclosure process:

— Identifying whether the reported problem is actually a public vulnerability. Some security findings are not actual vulnerabilities and must not be reported as such. Others are legitimate vulnerabilities but affect only private components used internally by organizations and are therefore outside the scope of CVE.

— Identifying possible duplicates and ensuring every unique public vulnerability is associated with a unique identifier. This identifier is in the form CVE-YYYY-XXXX, with the first part being the year of the beginning of the disclosure process, and the second part an incrementing integer.

— Gathering preliminary information about the vulnerability, in the form of at least one public reference (for instance, the vulnerability announcement from the software publisher) and a short free-form English description that includes a brief description of the problem, including the affected software and versions.

— Coordinating with the original vulnerability discoverer and the software publisher for an orderly yet timely disclosure process. Deciding the disclosure time can be difficult as there can be pressure in both directions [180]. Premature disclosure can lead to the software publisher not being able to provide a timely counter-measure for the vulnerability. On the other hand the longer a discovered vulnerability stays undisclosed the more likely it is to become exploited as a zero-day, either through a leak of the still-confidential disclosure process or through independent rediscovery

of the vulnerability [156].

CVE is overseen by the not-for-profit organization Mitre [203] but the actual vulnerability disclosure process is world-wide and decentralized. More than a hundred organizations act as *CVE Numbering Authority* or CNA [48]. All of them can lead a vulnerability disclosure process independently under Mitre's supervision. Most CNAs have a limited *scope* and are only concerned with vulnerabilities in a specific set of software products. Some software publishers, such as Apple and Microsoft, are also CNAs for their own products, allowing them to assign a CVE number to their products vulnerabilities without requiring third-party assistance. In the case individual CNAs would misbehave, Mitre still has the final word into deciding what is a legitimate CVE vulnerability or not. An example of CVE vulnerability can be found in Table 2.2.

NVD

Once a vulnerability has been disclosed by CVE, it is public. However at this stage the information about the vulnerability is only present in a raw form that is appropriate for security experts to get a quick understanding of the vulnerability, but not suited to automated processes.

Created in 2005, the NIST *National Vulnerability Database* (NVD) [128] aims to refine vulnerability information provided by CVE into *machine readable metadata* that can be consumed by automated vulnerability management processes, using various data formats such as SCAP [178]. This makes NVD the de facto vulnerability data source for *consumers* of vulnerability data.

The main information provided by NVD compared to CVE is a machine readable analysis of the *vulnerability severity* in the form of a *CVSS vector and score* (see below) and a machine readable enumeration of the *affected software and versions* in the form of a *CPE URI* (see below). This is done through a manual analysis by NVD security experts, often leading NVD data to be published days or even weeks after the original CVE (see more details in Chapter 3).

The relationship between CVE and NVD has evolved over time. Historically, CVE and NVD activities have been strictly separated [51] but in June 2020 NVD announced the CVMAP initiative [144], that aims to bring CVE's CNAs and NVD closer and notably letting CNAs publishing their own metadata information in the NVD database.

Base Vector		**Temporal Vector**	
CVSS V2	**CVSS V3**	**CVSS V2**	**CVSS V3**
Access Vector (AV)	Attack Vector (AV)	Exploitability (E)	Exploit Code Maturity (E)
Access Complexity (AC)	Attack Complexity (AC)	Remediation Level (RL)	Remediation Level (RL)
Authentication (Au)	Privileges Required (PR)	Report Confidence (RC)	Report Confidence (RC)
	User Interaction (UI)	**Environmental Vector**	
Confidentiality Impact (C)	Confidentiality (C)	**CVSS V2**	**CVSS V3**
Integrity Impact (I)	Integrity (I)	Collateral Damage Potential (CDP)	Modified Base Metrics (M*)
Availability Impact (A)	Availability (A)	Target Distribution (TD)	
	Scope (S)	Confidentiality Requirement (CR)	Confidentiality Requirement (CR)
		Integrity Requirement (IR)	Integrity Requirement (IR)
		Availability Requirement (AR)	Availability Requirement (AR)

Table 2.1 – Fields for Base, Temporal, and Environmental CVSS vectors in V2 and V3.

Description			
The HTTP/2 implementation in Apache Tomcat 9.0.0.M1 to 9.0.14 and 8.5.0 to 8.5.37 accepted streams with excessive numbers of SETTINGS frames and also permitted clients to keep streams open without reading/writing request/response data. By keeping streams open for requests that utilised the Servlet API's blocking I/O, clients were able to cause server-side threads to block eventually leading to thread exhaustion and a DoS.			
CVSS V3 Base Vector			
CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H			
CVSS V2 Base Vector			
AV:N/AC:L/Au:N/C:N/I:N/A:P			
CVSS V3 Base Score	7.5	**CVSS V2 Base Score**	5.0

Table 2.2 – Description (from CVE), CVSS V2 and V3 base vectors and scores (from NVD) for vulnerability CVE-2019-0199.

CVSS

The *Common Vulnerability Scoring System* (CVSS) is described as providing "*a way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity*" [42]. CVSS is a syntax one can use to answer standardized multiple-choice questions about a vulnerability. These questions can be answered objectively by someone who has knowledge of the inner workings of the vulnerability. Once the questions have been answered, the answers (or *fields*) can be compiled into a machine readable format called *CVSS Vector* and a vulnerability *severity score* between 0.0 and 10.0 can be computed using the *CVSS score formula*. The score formula is non-linear and small changes in the fields values can greatly impact the resulting severity score.

As of 2020 three versions of the CVSS standard are currently in widespread use: CVSS V2 [1], CVSS V3 [43], and CVSS V3.1 [44]. CVSS V2 and V3 have markedly different standardized questionnaires and formula, while CVSS V3 and V3.1 differ only by a minor change in the severity formula.

In all versions of CVSS the fields are divided into three *groups* or *subvectors*:

— The *base vector* describes the inherent properties of the vulnerability that do not change neither with time or context.

— The *temporal vector* describes properties of the vulnerability that change over time, such as the availability of a counter-measure, or a confirmed exploitation of the vulnerability in the wild.

— The *environmental vector* describes properties of the vulnerability in the context of a specific organization to protect.

Therefore the base vector is common to all people at all time, the temporal vector is common to all people at some point in time, and the environmental vector is specific to an organization at some point in time. Table 2.1 lists the existing fields for CVSS V2 and V3.

The CVSS base vector (in both V2 and V3) has been adopted as the de facto standard for measuring vulnerability severity, and is the format adopted by NVD for providing severity analysis of a vulnerability. In comparison the temporal and environmental vectors have been far less adopted as they require vastly more resources to be kept up to date and are rarely used in practice. Table 2.2 shows the description (provided by CVE) and the CVSS base vectors and score (provided by NVD) for vulnerability CVE-2019-0199 affecting Apache Tomcat.

An example of a CVSS field is the *Attack Vector* field (AV) from the base CVSS V3

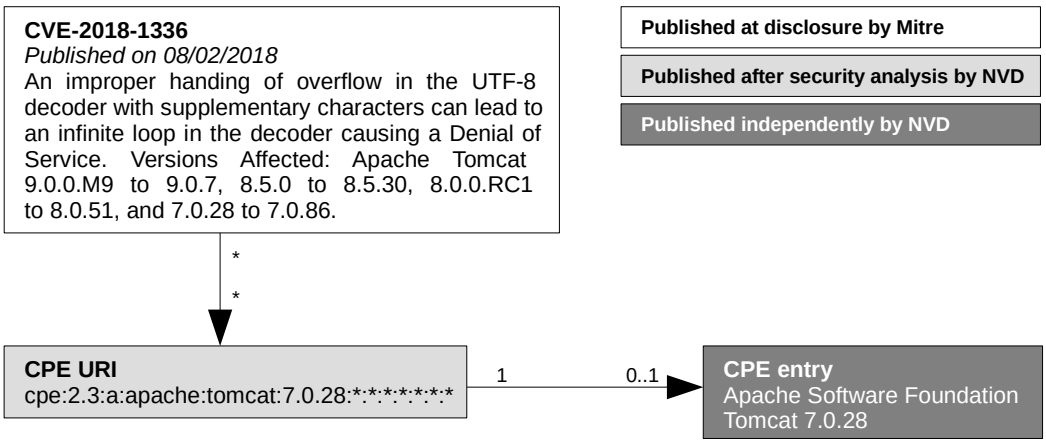

Figure 2.2 – The relationships between a CVE vulnerability and its associated CPE URI and entries. The vulnerability, its metadata, and the CPE entries have three different publication processes.

vector, which can be set to *Network* (AV:N, the vulnerability can be exploited remotely), *Adjacent* (AV:A, the vulnerability can be used from an adjacent network such as the same Wifi network or a local IP subnet), *Local* (AV:L, the vulnerability can only be used through a local access to the vulnerable system), or *Physical* (AV:P, the vulnerability requires physical manipulation of the vulnerable device).

There have been critics of CVSS and in particular of its severity score formulas, which have been argued as not reflecting expert consensus on a vulnerability severity [92], and having no correlation with the real-world impact of a vulnerability [8] [9]. Despite these critics CVSS has been the only vulnerability severity measurement framework to gain widespread adoption.

CPE

The *Common Platform Enumeration* (CPE) [132] is a notation to describe the naming and versioning of software components affected by vulnerabilities. The *CPE Dictionary* [139] is a database maintained by NVD about every piece of software ever affected by a vulnerability. CPE is separated into *URIs* and *entries* as depicted in Figure 2.2. The URIs act as references to entries located inside the CPE dictionary.

The CPE URIs include fields such as the affected software name in short form, the affected version, the software vendor name and the target software (the name of the software platform the affected software runs on: for instance an Android application's target software is Android). The complete CPE entry includes all these fields as well as additional information such as the affected software name in long form and sometimes in multiple languages.

It is worth noting that URIs and dictionary entries have independent publication processes. For every analyzed CVE vulnerability, NVD experts provide a list of CPE URIs that they consider to be associated with the vulnerability, without necessarily ensuring that a corresponding CPE entry actually exists in the dictionary. Therefore, the information about the affected software in a vulnerability comes in three independent steps: in raw form at disclosure in the text description of the CVE vulnerability, as a CPE URI when the vulnerability is analyzed by NVD, and as a CPE entry in the CPE dictionary at some point in time (before or after the publication of the related CPE URI). This lack of coordination impacts how CPE URIs and the CPE dictionary can be used in practice, as we show in Chapter 4.

CPE has been criticized for being a theoretical database authored manually by security experts, which by design cannot map perfectly to actual software binaries and packages in a production system [173, 204]. This situation creates a lot of discrepancies when doing analysis, such as associating a CVE vulnerability to incorrect or inexisting CPE entries. In particular, some software dependencies are difficult to establish, such as static inclusion of software librairies in a pre-compiled binary. As a concrete example, Bitcoin Core released an emergency patch [22] for the OpenSSL vulnerability Heartbleed [61] because it included a copy of OpenSSL compiled statically. However the CPE URIs proposed by NVD for Heartbleed never included Bitcoin Core as an affected software [140]. In Section 2.3.2 we discuss some attempts by the research community to remedy these shortcomings.

CWE

The Common Weakness Enumeration (CWE) [52] is described as "*a community-developed list of software and hardware weakness types*". It is maintained by Mitre. It aims to be the reference documentation of common *vulnerability patterns* that can be found across a wide range of software products and technologies. A few examples of CWE weaknesses have been selected in Table 2.3. For each CVE vulnerability they analyze, the security experts from NVD provide a reference to a related CWE entry in order to classify

CWE ID	Title	Description
CWE-287	Improper Authentication	When an actor claims to have a given identity, the software does not prove or insufficiently proves that the claim is correct.
CWE-620	Unverified Password Change	When setting a new password for a user, the product does not require knowledge of the original password, or using another form of authentication.
CWE-295	Improper Certificate Validation	The software does not validate, or incorrectly validates, a certificate.

Table 2.3 – Examples of CWE Weaknesses.

vulnerabilities by type of weakness.

2.2.3 Analyzing and Predicting the Vulnerability Life Cycle

CVE was started in 1999 and NVD in 2005. As of July 2020, more than 139 000 vulnerabilities have been disclosed by CVE and analyzed by NVD. This has allowed the research community to explore the vulnerability life cycle at scale instead of settling for anecdotal evidence as was the norm before.

The seminal work in vulnerability life cycle research was published by Frei et al in 2006 [73]. For the first time, an empirical examination of vulnerabilities was done at scale (14000 vulnerabilities were studied at the time), revealing global trends in the relationship between the time of discovery, disclosure, exploit, and patch of the vulnerabilities.

Multiple works followed [37, 183, 74] with the community gradually converging on three insights. First, the relationship between attacks and vulnerabilities follows a power-law, with a minority of vulnerabilities being exploited in a majority of attacks, and most vulnerabilities never being exploited at all [129, 7]. The actual rate of vulnerability exploitation has been subject to discussion, with figures going from 15 % [129] to 1.3 % [172] having been reported in the literature. Second, too many vulnerabilities are disclosed at any time for them to be treated with an equal level of urgency. More than 18000 vulnerabilities were disclosed by CVE [41] in 2018 and 2019, around 50 vulnerabilities per day on average. Prioritization is required. Third, usage of a vulnerability in the wild can increase as high as *five orders of magnitude* between before and after vulnerability disclosure [21].

These three facts taken together led the community to a conclusion: *vulnerability exploitation in the wild*, both in the present and the future, is the most important property

of a vulnerability to predict. By properly predicting exploitation in the wild, one can efficiently prioritize limited mitigation resources while still preventing most attacks.

Starting from 2010, most efforts in the community were devoted to finding accurate predictors of exploitation. As CVSS was gaining widespread use in the security community at the time, multiple teams attempted to use CVSS vectors for vulnerabilities as input of vulnerability exploitation models. None of these efforts led to satisfactory results [93, 8, 9], creating an entrenched divide in the community regarding the usefulness of CVSS [92], and even regarding the mere feasibility of unbiaised vulnerability statistics [29]. During the rest of the decade multiple other sources of information were proposed, with better success.

In 2010 Bozorgi et al [24] tried to predict future vulnerability exploitation using vulnerability metadata published in databases such as NVD, CVE, and OSVDB [150] (a now defunct vulnerability database predating NVD, with mostly similar content). Assuming the availability of this metadata, they were able to predict the exploitation of a vulnerability within two days with 85 % accuracy.

Allodi et al [9] compared future vulnerability exploitation prediction using CVSS vectors, presence of a public proof-of-concept exploit, and availability of a weaponized exploit for sale in vulnerability black-markets. They found the availability of the exploit in vulnerability black-markets to be the best predictor of exploitation in the wild.

As social networks such as Twitter became important venues for discussing vulnerability disclosure, multiple works attempted to use tweets as a data source for predicting current and future vulnerability exploitation [172, 125]. In 2015 Sabottke et al [172] used this technique and were able to detect vulnerability exploitation two days prior all other techniques and datasets available at the time. More recently, Tavabi et al [196] used dark web forums discussions to predict future vulnerability exploitation.

One of the most singular work was proposed in 2018 by Xiao et al [229], who correlated botnet activity and patch deployment activity at the ISP level to detect present vulnerability exploitation in the wild. They achieve a detection accuracy of 90 % and can detect exploitation earlier than previous state of the art techniques. This is the current state of the art in *present* vulnerability exploitation detection.

These prediction efforts culminated with the release in 2019 of the *Exploit Predicting Scoring System* (EPSS) [102], which uses many data sources to train a model answering a simple question: what is the probability of a vulnerability being exploited in the wild in the twelve months following its disclosure? The data sources included public sources such

as the NVD database, ExploitDB [67], Rapid7's metasploit framework modules [122], D2 Security's Elliot Framework [54], the Canvas Exploitation Framework [31], as well private data sources from organizations such as Proofpoint [53], Fortinet [71], AlienVault [5] and GreyNoise [83]. From all these sources they trained a regression model attempting to make explicable predictions. This is the current state of the art in *future* vulnerability exploitation prediction.

One interesting thing to note is that to our knowledge all of these models, including EPSS, are based on data usually generated after disclosure, such as human discussion or exploit trading happening after the vulnerability has already gone public for some time. While some of these models can provide reasonable results days or even hours after the disclosure, the ability to predict the danger posed by a vulnerability in the minutes or seconds after its disclosure has not improved significantly. This is an important problem as, as we can see in the next section, some critical *n-day* vulnerabilities have been exploited in very short time frames after their disclosure.

2.3 N-Day Vulnerabilities and Current Defenses Against Them

In this thesis we focus on a specific part of the vulnerability life cycle: *n-day* vulnerabilities, which are recently disclosed vulnerabilities still in the critical part of their life cycle. We first propose a summary of the real-world impact of n-day vulnerabilities in Section 2.3.1 then explore current mitigation best practices from the research and practitioners security communities: these include software management and patch deployment (Section 2.3.2), intrusion detection and prevention (Section 2.3.3), and information sharing (Section 2.3.4).

2.3.1 Real-World Impact of N-Day Vulnerabilities

In Appendix A we propose the case studies of several n-day vulnerabilities with major real-world impact, including *Heartbleed* [61], *Shellshock* [56], and *EternalBlue* [137, 220, 205]. These case studies include the complete timeline of each vulnerability from the introduction of the vulnerable code up to the widespread dissemination of the mitigation, marking the transition of the vulnerability from n-day to *well-known*. It also includes an assessment of the real-world damages and consequences of each vulnerability.

From these case studies we can draw a few observations. First, n-day vulnerabilities can have tremendous consequences: EternalBlue caused more than 14 billions dollars of damages, in part through its inclusion in the *WannaCry* ransomware that paralyzed thousands of information systems across the world, and also because of its use in *NotPetya*, the most devastating cyber-warfare attack in history, that destroyed the content of 1 in 10 computers across Ukraine. During the very last days of work on this thesis, the n-day vulnerability CVE-2019-19781 [159] led to a confirmed fatal casualty, through its inclusion in a ransomware that paralyzed an entire hospital, requiring a patient in critical condition to be diverted elsewhere. The patient passed away during transfer.

Second, dangerous vulnerabilities can stay non-public for a considerable amount of time. Shellshock and EternalBlue stayed non-public for decades, Heartbleed for years. Moreover, it is often difficult to assert with confidence whether a non-public vulnerability has actually stayed undiscovered or was independently discovered (and possibly exploited) as a zero-day by some parties before the publicly known discovery leading to its eventual disclosure. EternalBlue is confirmed as having been used as a zero-day before disclosure; this is still a topic of debate for Heartbleed and Shellshock. Third, once a vulnerability is disclosed, attacks can happen in a very short time: Shellshock led systems to be compromised just one hour after disclosure. Finally, having a proper patch at disclosure is not a guarantee: it took an entire week for Shellshock to be properly patched, all while being under active exploitation.

These observations show in particular that n-day vulnerabilities already have had widespread impact (including lethal consequences), sometimes in a very short amount of time after disclosure, and that it could happen again in the future. For further discussion we refer the interested reader to the case studies in Appendix A. We now describe the techniques currently proposed by the research and practitioners communities to defend against n-day vulnerabilities.

2.3.2 Software Management and Patch Deployment

After having established the risk posed by n-day vulnerabilities, in the rest of this section we study the current state of the art for mitigating them, from both the research and practitioners security communities. We focus on the case of a single organization willing to defend its own information systems from risks related to n-day vulnerabilities disclosures.

A first obvious yet critical way to diminish the risk posed by n-day vulnerabilities is

the ability to maintain an up-to-date inventory of the software and hardware used in a complex information system, and being able to quickly apply a pending security update related to any component used in the system. However this is not easy and does not fully solve the problem.

Software Updates are not Seamless

Nappa et al [127] used Symantec's WINE platform to assess security updates dissemination for various software on 8.4 million hosts that used the Symantec antivirus software. They found that on average, when a public exploit was released for a vulnerability, only 14 % of the hosts had deployed a patch mitigating the vulnerability. They also found that some software (such as Adobe Flash) were installed multiple times for more than 50 % of the hosts, with one installation patched but not the others, creating a false feeling of security for users that applied security updates but did not correctly catalog all copies of software residing on their machine.

Why is keeping software up to date so difficult in practice? We identify three problems: cataloging software and hardware, asserting if a component is vulnerable, and actually deploying a pending security update.

First, cataloging an information system exhaustively is hard. Software has many forms and many locations: while software binaries reside on hard disks, firmware is embedded inside hardware, and uncompiled source code can be interpreted at run time. Software can be downloaded on the fly over the network without touching the disk, such as when browsing the web with Javascript enabled. Hardware can be vulnerable [109][115] and is not easily updated or replaced. While in theory the CPE database [139] (which references all software and hardware ever affected by a vulnerability) can be used to track and monitor all these components, discrepancies between the real world and CPE entries (which are authored manually) quickly accumulate to the point that some have argued that proper vulnerability management is a myth [204]. This problem also impacts software versioning: as an example the Debian security team [57] is known to backport security fixes into debian packages without waiting for official releases of the upstream software, creating a debian-specific version number while doing so. While this practice should be lauded for improving the timeliness of Debian security updates, it adds confusion about how a given version of a software is supposed to behave. Also, when Nappa et al [127] analyzed the software installed in 8.4 millions hosts they remarked that "*identifying all vulnerabilities affecting a host is challenging due to the various vendor approaches for*

maintaining program and file versions".

Second, evaluating if a just-disclosed vulnerability affects a given piece of software or hardware is not straightforward. While in theory the CVE [41] database (which references all public vulnerabilities) and the CPE database are linked together through CPE URIs metadata annotating vulnerabilities, discrepancies are numerous here as well, as noted by Sanguino et al [173]. Several works have been proposed to improve the quality and the timeliness of the mapping between NVD's CVE metadata and CPE: Glanz et al [79] proposed to enhance software versions existing in NVD vulnerability metadata by extracting additional information from the vulnerability CVE description. In this thesis we present a contribution on automated prediction of affected software of a vulnerability at disclosure in Chapter 4 and Wåreus et al [221] have since built upon our work by improving the accuracy of the prediction at the cost of its explicability.

Part of the difficulty of evaluating if a software is affected by a vulnerability is *accidental*: due to the way the disclosure process has been historically separated between CVE and NVD (as discussed in Section 2.2.2), the timing of publication of vulnerability information is not optimal. While initiatives such as NVD CVMAP [144] have the potential to improve the base quality and timeliness of most disclosures, another part of the difficulty is *essential*: Shellshock's initial incomplete patch [56] meant no one in the vulnerability disclosure team (including long-time bash maintainer Chet Ramey) was able to fully grasp the extent of the vulnerability at the time of its disclosure. For automated systems and human experts alike, evaluating if and how a newly-disclosed vulnerability affects a given piece of software can only be answered with some degree of uncertainty.

Third, even when a newly-disclosed vulnerability is correctly recognized as creating a risk on a system, a security update is pending and is ready to be deployed, applying the patch safely is not always straightforward. Software updates can exhibit unforeseen backward-compatibility problems, which put system administrators in a dilemma: should they apply security updates quickly at the risk of breaking critical business processes, or should they leave a known vulnerability unpatched [152]? This problem is particularly acute in safety-critical systems such as health-care or avionics which are often relying on legacy software [80], creating life-threatening situations when the wrong choice is made.

Moreover the cloud has heightened the stakes of this problem: Zhang et al [235] pointed the trend of cloud clients to rely on pre-existing OS images to build their cloud virtual machines and how this trend created a concentrated risk when a vulnerability is detected in widely-used images. This concentration creates opportunities for both attack and defense

at scale.

The way software is actually updated, and how this affects updates dissemination timeliness has also been subject of study. Duebendorfer et al [60] studied how web browsers' update mechanisms impacted the timeliness of security updates deployment. They found that Google Chrome's completely automated update mechanism allowed it to have to the most up to date fleet of browsers after three weeks (near 100 % of the browsers were up to date), while Firefox's one-click update mechanism (at the time of the study in 2009) led to the quickest burst of updates after five days (near 75 %).

Software Updates are Necessary but not Sufficient

As said before, complete, correct and timely software patching is critical to defend against n-day vulnerabilities but is not a complete defense against them. Sometimes there is simply no viable patch to be deployed when the vulnerability starts to be exploited, either because none has been prepared in time by the software publisher, or because deploying the available patch would create unacceptable problems for the information system. Patch authoring timeliness did improve in the last few years: the Google Project Zero team [11], which looks for vulnerabilities in widely used closed and open-source software, has a strict disclosure delay policy: 90 days after contacting a publisher regarding a vulnerability they discovered, they publicly disclose the vulnerability regardless of the availability of a mitigation. They remarked that at the time of the team inception (in the aftermath of Heartbleed in 2014) it was common for vulnerabilities to take up to six months to be mitigated, while in 2019 97.7 % of their reports were fixed before the 90 days delay [156].

Still, it is inevitable that there will be n-day vulnerabilities exploited before being fully patched (which has already happened with Shellshock) or information systems that have legitimate reasons not to apply the available patch. In the next subsections, we explore n-day vulnerabilities mitigation beyond patch management.

2.3.3 Intrusion Detection and Prevention

Intrusion detection and prevention is an important field with regard to n-day vulnerability mitigation. IDSs raise alerts regarding activity they consider malicious, creating the possibility of *false positives* and *false negatives*.

Definition 23 *Intrusion Detection aims to analyze computer activity to determine if this*

activity is legitimate or malicious. A security system doing intrusion detection is called an Intrusion Detection System (IDS).

Definition 24 *A false positive is an alert being emitted superfluously while the situation does not warrant it.*

Definition 25 *A false negative is the erroneous absence of an alert while the situation warrants the emission of an alert.*

Types of Intrusion Detection Systems

IDSs are usually classified depending on *what* activity they analyze and *how* they analyze it.

Definition 26 *Host-based IDSs (HIDSs) reside on a single computer system and evaluate if the behavior of the various processes executed on the computer system are reflecting legitimate or nefarious activity.*

Unlike an antivirus software which usually focus on the content of files, HIDSs analyze runtime behavior, such as the sequence of system calls made by a given process in its lifetime [26].

Definition 27 *Network-based IDSs (NIDSs) reside on a computer network: they are given a copy of every network packet traveling inside the protected network and assess if the packet is legitimate or part of malicious behavior.*

Unlike HIDSs, NIDSs can monitor entire computer networks at a time [110]. Moreover, IDSs are also separated according to how they make their decisions.

Definition 28 *Signature-based (sometimes called rules-based) IDSs are based on expert knowledge, gathered in the form of* signature databases.

These signatures use simple rules (such as string matching) to detect *known, specific* attacks [195]. However they are useless against never-seen-before attack techniques [75], for which *anomaly-based IDSs* are (in theory) better suited.

Definition 29 *Anomaly-based IDSs use statistics techniques or machine learning to separate normal and anomalous activity.*

While not all anomalous activity is malicious, all malicious activity is by definition anomalous. However, neither signature-based nor anomaly-based IDSs can prevent an attack from succeeding, they merely report it. Blocking malicious activity is the task of *Intrusion Prevention Systems (IPSs)*.

Definition 30 *Intrusion Prevention Systems (IPSs) behave like an IDS, but after deciding that an activity is malicious, they block it.*

Network-based IPSs drop packets that are considered malicious before they reach their destination [27][117][75], and host-based IPSs block unauthorized actions before they are executed by the OS [228]. IPSs have additional constraints compared to regular IDSs, mostly because they have to make an analysis decision in real time [232].

Signature-Based IDSs

Signature-based IDSs and IPSs already play an operational role in the mitigation of n-day vulnerabilities. The cloud industry has coined a domain-specific term for signature-based network IPSs: *Web Application Firewall* (WAF), with many commercial offerings such as Amazon Web Services WAF [13], Google Cloud Armor [81], or the Cloudflare Web Application Firewall [40], while other actors sometimes use the term *Next-Generation Firewall* (NGFW) [130]. There are also widely used open-source IDSs such as Snort [195] and Suricata [193] that can be used with free and commercial signature databases.

However signature-based intrusion detection and prevention share many of the same shortcomings as patch management: a human expert needs to have enough time to understand how the vulnerability is exploitable to create signatures detecting the associated attacks. The signatures need to be disseminated to all IDSs that protect an instance of the vulnerable software. Large-scale cloud IPSs or WAFs make this last part easier, as a single signature deployment can protect millions of information systems at once. For instance, Cloudflare has an history of adding new rules and signatures to its WAF offering in a timely manner [39], and as seen in Appendix A.2 they were able to deploy a signature mitigating Shellshock three hours after its disclosure. But while this timeliness was praiseworthy, it was still not enough as the first recorded Shellshock attack occured less than one hour after disclosure.

Network-Based, Anomaly-Based IDSs

Anomaly-based IDSs and IPSs, by contrast, can protect computer systems against new attacks and should in theory be better suited to mitigate n-day and zero-day vulnerabilities. In particular, network-based, anomaly-based IDSs (NAIDSs and NAIPSs) could be the Holy Grail of n-day vulnerability mitigation for information systems, as they can protect a whole network at a time against unknown attacks, including n-day vulnerabilities for which no patch nor signature is available yet.

NAIDSs have been a research topic for several decades, and a complete state of the art of the field is outside the scope of this thesis (we can refer the interested reader to surveys by Buckzak et al [27], Bhuyan et al [117], and Garcia et al [75] for going deeper). Instead we focus on why these considerable research efforts did not lead to security products that are actually used by security practitioners.

A first problem is that anomaly-based intrusion detection shares all the challenges of machine learning for security alerting for which we dedicate the complete Section 2.4. However there are also some difficulties specific to intrusion detection.

The biggest challenge is that properly evaluating an anomaly-based IDS is still an open problem. In this context, intrusion detection is a prediction problem, and prediction techniques are typically evaluated by maintaining an *evaluation dataset* made of annotated data: each piece of data is fed as an *input* to the prediction system. This input is annotated with a *ground-truth* and an *evaluator* compares the prediction made from the input and the ground-truth value that should be predicted. Some prediction techniques require some form of *training data* (this is called *supervised learning*, see Section 2.4.3) and some do not (*unsupervised learning*, also Section 2.4.3) but in all cases we need an annotated evaluation dataset to see how well a given prediction technique actually works compared to another.

All NAIDS evaluation datasets have serious shortcomings. Viegas et al [217] and Hindy et al [88] identified several properties required for NAIDS evaluation datasets to be relevant.

First, the dataset should be *realistic* and *generalizable*. This is a problem because an IDS can be deployed in many contexts: information systems big and small, domestic LANs and Internet backbones are all valid situations for which one would wish to use an IDS yet they have very different definitions of normal traffic. Some datasets [211] are based on actual traffic captured in a real network while others are more artificial: either a made-up computer network was created from scratch and the resulting network activity was captured [233], or the traffic data was entirely generated by simulation [6]. None of these

approaches are a panacea: real traffic is filled with personal data that makes it difficult to share publicly, while generated traffic risks not being realistic enough.

Second, the dataset should be *valid*. This means the base-rate (the ratio between malicious and non-malicious traffic) should be realistic, and all interesting classes of attacks should be covered. This is a problem because the base-rate is usually very low (a vast majority of the traffic is legitimate), which implies that an exhaustive dataset must be very big to stay realistic. Moreover, the state of the art in attacks is not public and constantly evolving, making it challenging to assert the completeness of the coverage. Hindy et al [88] identified a taxonomy of network threats and concluded that only 33.3 % of these threats were covered by existing IDSs datasets.

Third, the dataset should be *correctly annotated*. This is particularly challenging for datasets based on actual traffic as human authoring of ground-truth annotations is untractable for large amount of traffic data. Moreover, for actual traffic no oracle (human or IDS) is able to assess with 100 % certainty that a given piece of traffic is legitimate or part of a covert attack, because of the non-public state of the art in attacks.

Fourth, the dataset should be *current* and *updatable*, as both normal and attacker traffic evolve over time. Even a perfect dataset on all fronts would become outdated after some years if not updated regularly. This makes proper IDS evaluation an ongoing process, which is even more challenging than traditional scientific reproducibility (which is already difficult in a lot of scientific fields).

Fifth, the dataset should be *public*, or at least *shareable*, to allow other researchers to compare their IDSs against existing ones. This is particularly difficult for datasets based on real traffic comprising personal or confidential data.

As soon as 2010 Tavallaee et al [197] and Sommer et al [187] concluded that past NAIDS research had profound methodology issues challenging the validity of most research papers in the field. Overall NAIDSs have not been adopted by the security industry and to our knowledge there is only one open-source NAIDS available, Hogzilla [90], which has close to no adoption. Since 2010 the evaluation of NAIDS datasets themselves has become a growing topic of research [164, 78], but the ideal dataset or evaluation protocol remains as elusive as ever. Therefore even recent NAIDS works [233] still use standard IDS evaluation datasets, such as NSL-KDD [138] which is based on the KDD-99 dataset [107] which was itself based on the DARPA98 dataset [55]. Whatever the merits of DARPA98 when it was released, typical network traffic in 2020 has nothing to do with typical network traffic in 1998, which inevitably leads these evaluations to lack realism.

Other recent works [124] included the creation of dedicated, more realistic evaluation environments, which is laudable but raises questions regarding the validity and generalizability of their evaluation protocol, as these dedicated evaluation protocols may have shortcomings themselves. Moreover this makes comparison between multiple IDSs challenging as each of these works use a different evaluation protocol from the others.

These difficulties prevent NAIDSs from becoming a silver bullet to defend against n-day vulnerabilities. This may change in the future if a breakthrough happened either in NAIDS scientific evaluation or NAIDS real-world applicability.

2.3.4 Information Sharing

As seen in the previous section, cyber defense can sometimes be more of an art than a science as the true state of the art of attackers is non-public, which makes it difficult to reliably assert the realism and relevance of an evaluation protocol. Therefore it can be helpful to take a step back and look at which techniques security practitioners have converged towards when dealing with actual threats in the real world.

The common thread of all these practices is information gathering and sharing. Over the years practitioners have found that the more information one gathers into one place and has access to, the more equipped one is to connect the dots and realize that an attack is taking place. A first success story of information sharing is the vulnerability disclosure community, which we detail in Section 2.2. In this section we present four other information sharing endeavors: *Endpoint Detection and Response* (EDR), *Security Information and Event Management* (SIEM), *Unified Threat Management* (UTM), and *Cyber Threat Intelligence* (CTI).

Endpoint Detection and Response

Endpoint Detection and Response (EDR) [214] is a set of software and practices aiming to protect the *endpoints* of an information system, e.g. the computer hosts manipulated by users on a day to day basis. EDR systems gather multiple security metrics from endpoints and send them to a central coordination entity, which can then trigger remote reactions to be executed by the endpoint. Common security metrics are process activity (such as disk writing and reading, or network communication) while common reactions include network isolation or even computer shutdown. EDR systems are also a great fit with IDSs as the combination of both allows the analysis of the host and network data of an entire

information system in a single process, facilitating the detection of unusual activity. A typical scenario for EDR systems is the rapid detection and mitigation of a ransomware attack propagating across an organization network [223].

Security Information and Event Management

The quantity of data gathered for security analysis (such as metrics collected by EDR agents or network IDSs) makes it challenging to navigate and use meaningfully. The industry solution to this problem is called *Security Information and Event Management software* (SIEMs) [213]. SIEMs aim to centralize all available security information in any form and make it available for security analysts to use in their investigations. Analysts can then automate parts of their tasks by creating SIEM rules triggering security alerts according to patterns in the incoming data. According to Bhatt et al [19], a major challenge for SIEMs is the ever-growing volume of data to be processed, which manifests in several ways: analysts have difficulties creating alert rules that do not generate too many false positives and negatives (see Section 2.4), and the scalability requirements for SIEM hardware and software make them more and more difficult to engineer and maintain.

Unified Threat Management

Unified Threat Management (UTM) [215] is the trend of packing multiple security functions inside a single piece of software or hardware. For instance, a UTM device can combine a network firewall with a network IDS and IPS, and include other features such as a Virtual Private Network (VPN) or even a SIEM. UTM is inherently a tradeoff as it provides ease of use and natural integration of otherwise disjoint elements, but this creates the risk of a single point of failure and is contrary to the doctrine of defense in depth.

Cyber Threat Intelligence

Cyber Threat Intelligence (CTI) is the practice of sharing knowledge about attacks and attackers between multiple organizations targeted by common threats, in an effort to get a better picture of the attackers, their motivations, and the *tactics, techniques and procedures* (TTPs) they use. CTI starts with collecting and sharing forensic evidence of an attack (such as computer logs showing an anomalous behavior in an information system) and attempts to reconstruct the *kill chain* of the attack: which procedure the attackers

used to intrude the system and which resources they targeted once they were inside. The end-goal of this process is *attack attribution*: determining the identity and motivation of the attackers from the evidence they left when they attacked.

While information sharing between organizations is a laudable effort, it creates the challenge of agreeing on a common vocabulary and standardizing the way attack knowledge is encoded and transferred.

Multiple standardization efforts for CTI have been proposed, with the *Structured Threat Information eXpression* (STIX) [14] being the closest to an industry standard. Like CVE, STIX is proposed by Mitre [203] and is an XML Schema aiming to represent CTI expert knowledge. However CTI as a field has been the subject of criticism. Mavroeidis et al [120] argued that no current standard including STIX are readily available for use for CTI, as each existing one has some shortcomings. Oosthoek et al [147] argued that CTI is a product without a process, with the field lacking results for multiple reasons such as methodology deficiencies, lack of sharing, low quality of shared knowledge, bias of CTI actors, and difficulty of attack attributions. Some of these problems are acknowledged by the French *Agence Nationale de la Sécurité des Systèmes d'Information* (ANSSI) and motivated the creation of OpenCTI [148], an open-source CTI platform. They were also the motivation for TTPDrill [97], an automated system extracting threat actions from unstructured textual CTI sources.

It should be noted that these tools and practices (EDR, SIEMs, UTM, and CTI) have had little scientific validation from the research community so far. This can make it difficult sometimes to separate valuable tradecraft from security theater. Whatever the merits of the current state of the industry, current practices did not prevent catastrophic n-day vulnerabilities such as EternalBlue from occuring. Both researchers and practitioners need to innovate to better protect information systems and their users in the future.

2.4 Machine Learning for Security Alerting

The recent progress in artificial intelligence (AI) and machine learning has had a profound impact on many topics of Computer Science and society in general. Information security is no exception. In 2019 there have been more than 100 machine learning research papers published *every day for the whole year* [207]. It is therefore largely beyond the scope of this thesis to provide an exhaustive state of the art of the field.

Instead we focus on the unique challenges of applying machine learning to security,

and more specifically to *security alerting*: deciding whether a perceived situation presents a risk or not, with the possibility of both false positives and false negatives.

Security alerting is a critical topic for improving the accuracy of many security tools such as IDSs and SIEMs. In Section 2.4.1 we study the reasons to use (and not to use) machine learning for security alerting. In Section 2.4.2 we review the difference between machine learning, data mining, and information retrieval. In Section 2.4.3 we review the different types of machine learning tasks, such as supervised and unsupervised learning, and others. In Section 2.4.4 we study the most common machine learning challenges, and review commonly used solutions to them in the context of machine learning for security alerting. In Section 2.4.5 we study several machine learning techniques we use in the contribution of this thesis. Finally, in Section 2.4.6 we review how machine learning was used to make progress on the vulnerability life cycle (studied in Section 2.2), with a specific focus on decision explicability.

2.4.1 Why Using Machine Learning for Security?

As both the software industry and the research community rush to apply machine learning to many open problems without always considering if it is appropriate, it can be helpful to assert the goals and non-goals of applying machine learning to critical security decisions such as security alerting.

There are two main properties one may wish to improve by applying machine learning in this context: *decision cost* and *decision timeliness*.

Definition 31 *Decision cost is the monetary and human cost of making a decision.*

Definition 32 *Decision timeliness is the time necessary to make a decision from a given input.*

Currently a lot of security decisions are made by human security experts, which are expensive and in a short supply. As seen in Section 2.1.3, there is a shortage of SOC analysts on the job market and current SOC analysts face an elevated risk of burnout. The present monetary and social cost of security decisions is therefore very high.

Even with unlimited access to security experts, humans have a slow reaction time while attacks can spread in milliseconds. Therefore the time to make a security decision, or decision timeliness, is the second property that could be vastly improved by using machine learning.

However even if improving decision cost and timeliness is valuable, machine learning also has an impact on other properties, such as *decision accuracy*, *decision explicability*, and *decision reliability over time*. These are properties one may wish to preserve.

Definition 33 *Decision accuracy, in the context of a prediction system, is the average rate of correct decisions according to an evaluation protocol proposed by the designers of the prediction system. Incorrect decisions can be either false positives or false negatives.*

Definition 34 *Decision explicability, in the context of a prediction system, is the ability for a third party to evaluate and understand how and why the prediction system made a decision (correct or not).*

Definition 35 *Decision reliability over time is the measure of whether measuring decision accuracy at a given point time can help predict decision accuracy at a later point in time.*

If diminishing the cost and time required to make security decisions leads these decisions to become less accurate, one has to make a subjective assessment regarding whether the trade-off is worth it. Monitoring the accuracy of past security decisions is therefore critical to make a balanced judgment.

Some machine learning techniques are known as *black-boxes* where the reasons that led a predictor to make a decision are not *explicable* [231], neither by a domain expert nor a machine learning expert. This is a problem for life-critical and business-critical decisions: a lack of decision explicability can erode trust in the prediction, especially when the odds of making an inaccurate prediction are not negligible.

Last but not least, the more complex the prediction system the more difficult it is to track its potential *drifts* over time, for instance because the production environment the system is used in differs more and more from the assumptions taken during training and evaluation.

Definition 36 *Accuracy drift is a gradual divergence between the initial evaluation environment of a training system and the production environment it is actually used in, leading to a lack of decision reliability over time.*

For these reasons, evaluating both the metrics we want to improve and the ones we want to preserve is critical when applying machine learning in the context of information security. But as seen in Sections 2.1.3 and 2.3, many security problems are difficult to

evaluate properly. For instance SOCs are idiosyncratic environments and IDS evaluation is challenging. As proper evaluation is simultaneously required yet difficult to achieve in practice, caution is advised when applying machine learning methods to information security.

Explicability plays a key role in all the contributions of this thesis and is an important criterion when comparing the machine learning techniques we discuss in the next sections.

2.4.2 Machine Learning, Data Mining, and Information Retrieval

Machine learning, *data mining* and *information retrieval* are three separate yet related research domains. The common thread of all three domains is the use of data to achieve a goal.

— Machine learning aims to use data to train a predictor to emulate the behavior of a function, while not having access to the function itself but only to examples of its use. Starting from a corpus of all past CVE vulnerabilities, a typical machine learning task would be to automate the generation of a convincing vulnerability description.

— Data mining aims to extract new insights from a data collection. Starting from a corpus of all past CVE vulnerabilities, a typical data mining task would be to identify the software and hardware most often affected by vulnerabilities.

— Information retrieval aims to select the most relevant documents in a corpus according to a user query. Starting from a corpus of all past CVE vulnerabilities, a typical information retrieval task would be to list all vulnerabilities affecting a specific piece of software or hardware, provided as a user query.

While each research domain has a different objective and community, there is some overlap in the techniques they use and in this thesis we made the choice to present the three of them as a whole.

2.4.3 Learning: Supervised, Unsupervised and Beyond

A cornerstone of machine learning is the use of data for *training*. Machine learning techniques are often separated depending on the way they use training data to achieve the learning goal. In this section we cover five classes of machine learning tasks: *supervised learning*, *unsupervised learning*, *semi-supervised learning*, *active learning*, and *reinforcement learning*.

Supervised Learning

As mentioned above, the goal of machine learning is to learn the behavior of a function from a set of examples of its use. *Supervised learning* [171] is used when we have access to both the input (the *sample*) and the corresponding output (the *label* or *annotation*) for the function. A supervised learner then constructs a *model* that emulates the behavior learned from the labeled samples: given an input in the same format as the trained data, the model outputs a prediction based on what it has learned of the hidden behavior of the function.

Supervised learners can be evaluated using another distinct set of *evaluation data* also made of labeled samples from the same function. Model evaluation is important to assess the ability of the model to *generalize*, e.g. to provide reasonable results on inputs it has not seen during its training. *Over-fitting* is the opposite of generalization: an over-fitted model performs very well on its own training data but poorly on evaluation or production data.

Examples of supervised learning applied to security alerting includes *DIADEM* [16], a supervised learning framework proposed by Beaugnon et al for security tasks such as malicious PDF files detection. Another example is the many intrusion detection works [27, 117, 75] based on fully annotated datasets such as DARPA98 [55] and its derivatives.

The requirement of having access to *labeled data* makes supervised learning difficult to apply in some contexts. For instance an IDS based on supervised learning would require labeling network packets from the training data as legitimate or malicious, a costly and difficult proposition. This requirement can be relaxed using other learning techniques, which we detail next.

Unsupervised Learning

Unsupervised learning [32] is a class of learning tasks that do not require the training data to be labeled. Common types of unsupervised learning tasks include *clustering* (grouping samples into subgroups depending on their similarity), *anomaly detection* (finding outliers among samples), and *ranking* (compare one sample to another according to a metric). In particular, anomaly-based IDSs can be implemented using anomaly detection techniques applied to the problem of intrusion detection, such as in the works of Bivens et al [23] and Leung et al [113].

One problem of unsupervised learning is that although data annotation is not required

for *training*, some ground-truth is still required for *evaluation*, which often implies having to label the evaluation data in practice. When proper labeling is impractical, unsupervised learning tends to become as difficult to implement as supervised learning in practice.

Semi-Supervised Learning

Semi-supervised learning [237] is a hybrid class of learning where a small part of the training data is annotated but the vast majority is not. Semi-supervised techniques can be considered when properly labeling data is costly but possible. They often take assumptions regarding the data, such as similar samples tending to share the same label. Moreover, labeling the evaluation data is still important for proper evaluation.

An example of semi-supervised learning applied to security is *EASEAndroid* [218], a framework to automate policy analysis and refinement for SEAndroid [179].

Active Learning

Active learning [181] is another hybrid class of learning where training is an interactive process between an automated learner and an oracle such as a human expert. The learner starts with a set of unlabeled data. It then has to solve two learning problems: the first is to select the next sample to be submitted to the oracle for labeling, and then to use both the labeled and unlabeled data to make predictions.

The main difference between semi-supervised learning and active learning is interactivity: each label provided by the oracle should impact which sample should be submitted next.

Active learning is an interesting choice for information security as the same channel between the learner and the oracle can be used for both training and evaluation: once the system is trained, random unlabeled samples can be sent to both the learner and the oracle to compare their answers. The oracle, the learner, the data and its annotations can all remain in one security context, avoiding the need to share sensitive datasets between multiple security contexts.

Active learning has two main challenges: the first is *oracle availability*. For instance security experts may have limited time to answer annotations from an active learner and their time is very expensive. This can be controlled by allotting an *annotation budget* to the learner, but this raises the question of the feasibility for the learner to reach an acceptable level of accuracy within the given annotation budget. This problem is particularly acute when there is a vast number of hyperparameters (see Section 2.4.4) to set before training:

a bad choice of hyperparameter settings can simultaneously waste the training budget by submitting superfluous samples to be annotated to the expert, while limiting the ability of the model to infer correct decisions from the training data.

The second challenge is *generalization*. While active learning is helpful to learn the context of an idiosyncratic environment, the learner may not perform as well in a different context or when evaluated against another oracle (such as a different security expert from the one the system was trained with).

An example of active learning for security is *ILAB* [17], an active learning for security framework designed by Beaugnon et al aiming to facilitate the annotation of intrusion detection datasets. We make use of an active learning technique proposed by Settles et al [182] in a contribution detailed in Chapter 6.

Reinforcement Learning

Reinforcement learning [105] does not require a dataset. Instead the learner is put inside an *environment* and provided with a *utility function* to maximize. A reinforcement learner has to find a balance between *exploration* and *exploitation*: when exploring, the learner tries to discover new things about its environment. When exploiting, the learner maximizes the utility function based on what it already knows.

Reinforcement learning is appropriate when there exists an environment where a learner can make mistakes without excessive consequences. Training can therefore take place in simulations, test-beds, or low-stake production environments as long as the training environment is close enough to evaluation and high-stake production environments.

Utility function design is another important consideration. For example there are numerous examples of learners respecting the letter of their utility function while ignoring the spirit behind it, a phenomenon called *specification gaming* [189]. An example of reinforcement learning applied to security is the work of Xiao et al [230] on mobile edge caching in 5G networks to prevent jamming attacks.

2.4.4 Common Machine Learning Challenges

In this section we describe a selection of common problems encountered in most machine learning endeavors.

Document	**Content**		
A	Little Red Riding Hood		
B	Little Red Corvette		
C	Little By Little		
Word	**Doc A**	**Doc B**	**Doc C**
By	0	0	1
Corvette	0	1	0
Hood	1	0	0
Little	1	1	2
Red	1	1	0
Riding	1	0	0

Table 2.4 – Examples of bag-of-words applied to a corpus.

Featurization

Most of the data useful for machine learning is unstructured. Meanwhile, the vast majority of machine learning techniques take a vector of real numbers as input.

Definition 37 *Featurization is the process of translating data from an arbitrary format to a set of vectors of real numbers.*

Featurization is an important topic of machine learning applied to security, as the right or wrong selection of features can deeply impact the accuracy and explicability of a machine learning system. Automated featurization from raw data is an active research domain and in 2018 Beaugnon et al [16] concluded that the current state of the art is not satisfactory enough for security decisions.

While featurization has to be solved on a case-by-case basis, the problem is ubiquitous enough to have spawned many general-purpose solutions.

In particular, text featurization is the process of converting a corpus of documents into set of vectors of real numbers. Corpus of documents are a very common type of input data in machine learning pipelines, and text featurization is necessary for all the contributions of this thesis. Text featurization has historically been handled through an algorithm called *bag-of-words*, proposed by Harris et al in 1954 [85]. As shown in Table 2.4, bag-of-words is used on a corpus of documents and treats the word index of the entire corpus as a multi-dimensional space, each dimension representing an individual word. Individual documents are then treated as vectors in this space, with each word count populating a dimension (zero indicating the absence of the word in the document).

Bag-of-words is simple, efficient, and easily explicable. It is still widely used in the research community (including in two contributions of our own in Chapter 4 and 5) but has some drawbacks. The main one is that it does not preserve grammar and word order inside the document, which creates a loss of information during the featurization stage. Other text-to-vector featurization schemes have been proposed, such as paragraph vectors [112] which use backpropagation training to encode paragraph semantics in the vector, at the cost of simplicity and explicability.

Dimension Reduction

Once training samples have been converted into real vectors, various machine learning algorithms can be applied to them. However, most of these algorithms have superlinear time or space complexity in the number of dimensions of the data. This creates a phenomenon called *curse of dimensionality* where some algorithms become intractable on highly dimensional data. Some featurization schemes can exacerbate this. For instance bag-of-words adds a new dimension for every new word in the index of the corpus. The index commonly grows linearly with the size of the corpus, leading some techniques to break down after the training data becomes too large. This situation creates the need for *dimension reduction* [216], a way to keep the number of dimensions of the input data manageable.

Definition 38 *Dimension reduction is the process of managing the number of dimensions of an input space to ensure that the input space is tractable in the context of a chosen machine learning technique and available computing resources.*

Dimension reduction can be done through feature *selection* and feature *projection*. Feature selection [103] is about retaining the original dimensional space but discarding individual dimensions that are considered irrelevant. Explicability preservation is the main advantage of feature selection, as a dimension in the reduced dimensional space is still the same dimension as it was in the original space (for instance, a bag-of-words followed by feature selection results in a smaller dimensional space where each dimension still represent the count of occurrence of a specific word in a document). However the feasibility of feature selection is often data-specific, making it difficult to generalize.

On the contrary feature projection [216] is about creating a new space with a lower number of dimensions while creating a *projection* from the original space to the new space. Feature projection is an active research topic but older approaches are widely used,

including *Linear Discriminant Analysis* (LDA), *Principal Component Analysis* (PCA) and *Locality Sensitive Hashing* (LSH) [98]. However the new dimensional space created during feature projection is meaningless, which breaks the explicability chain from the input data throughout the machine learning system.

For the sake of explicability we make use of feature selection in all the contributions of this thesis.

Similarity Measurement

Once input data has been translated into real vectors of a reasonable size, a common question is how *similar* a pair of samples is compared to another pair. Similarity measurement is an integral part of many unsupervised learning techniques such as clustering or anomaly detection.

The concept of similarity is sometimes expressed as a construction on top of the concept of *distance*. As an example, the *Radial Basis Function Kernel* (RBF Kernel) [33], described in Equation 2.1 is a similarity measurement based on the euclidean distance between two vectors $\mathbf{x}$ and $\mathbf{x}'$ of n dimensions, with the similarity decreasing exponentially the bigger the distance between them gets. The exponential decay rate is adjusted using the hyperparameter σ.

$$\text{RBF Kernel}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right), \tag{2.1}$$

An overlooked aspect of distance-based similarity measurement is that similarity between vectors that have few or no dimensions in common is computed from the norm of each vector. This can tend to exacerbate the similarity between vectors of lower norms, as they are close to the origin of the dimensional space.

In some cases it can be desirable to only consider similarity from the shared dimensions of two vectors. This can be accomplished using similarity measurements that are not based on distance, such as the *cosine similarity* [185], described in Equation 2.2. As the cosine similarity is based on the dot product of the vectors, vectors that do not share any dimensions have a similarity of zero. We make use of cosine similarity in Chapter 6.

$$\text{cosine similarity}(\mathbf{x}, \mathbf{x}') = \frac{\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}'_i}{\|\mathbf{x}\| \|\mathbf{x}'\|}, \tag{2.2}$$

Hyperparameters Tuning

A *hyperparameter* is a parameter that has to be chosen before training occurs, has a direct correlation with model performance (the computing and time resources required for the model to be trained and used), but has a non-obvious impact on decision accuracy.

Definition 39 *A hyperparameter is a configuration parameter of the training process of a machine learning pipeline.*

A typical example of hyperparameter is the *stopping condition* of iterative training methods [91, 236], which can be a number of iterations, an acceptable error threshold, or both. Many machine learning techniques require additional hyperparameters specific to their inner architecture (for instance, deep learning which we cover in Section 2.4.5 requires many hyperparameters to define the neural network topology).

A given combination of settings for all the hyperparameters in a machine learning pipeline is called a *hyperstate*. The space of all possible hyperstates for a given pipeline is called a *hyperstate space* and grows exponentially with the number of hyperparameters. The optimal choice of hyperparameters settings depends very much on the training and evaluation data, which creates a risk of over-fitting.

Moreover, as hyperparameters must be set before the beginning of training (which can be lengthy), a highly dimensional hyperstate space cannot be fully explored repeatedly. As the accuracy impact of the hyperparameters is difficult to reason about and a thorough evaluation of all possible hyperparameters is often impossible, hyperparameters tuning is often done in a rather empirical manner. An excessive number of hyperparameters is therefore an obstacle to prediction explicability, as a large hyperstate space becomes a black box. In the next section we review several machine learning techniques, notably discussing how many hyperparameters they require.

2.4.5 Machine Learning Techniques

In this section we describe several machine learning techniques as a background for the rest of this thesis. We first focus on two techniques we use in the contributions of this thesis: regression analysis and ranking functions. We then briefly cover other noteworthy machine learning techniques we do not use, such as deep learning.

Regression Analysis

Regression analysis [72] is perhaps the earliest form of machine learning, actually predating computer science by more than a century. Despite its age it is still a very relevant tool in many security alerting problems.

There are several forms of regression analysis. In particular, the well-known *linear regression* is based on finding a *linear combination* of the input vectors that properly fits a desired output, which is itself a real number. The process of fitting a linear regression consists in finding the appropriate linear coefficients for every dimensions in the input space. Equation 2.3 shows how a linear model β of $n+1$ dimensions and an input vector x_i of n dimensions can approximate an output y_i with error ε_i. We make use of linear regression in Chapter 5.

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_n x_{in} + \varepsilon_i, \qquad i = 1, \ldots, n, \tag{2.3}$$

Sometimes y_i is better approximated by a non-linear function of x_i. In these cases *non-linear regression* [72] can be applied by finding a function f such that $y_i = f(\beta \cdot x_i) + \varepsilon$.

Linear regression has two very interesting properties for security alerting: it is very explicable, as the importance of every dimension in the input is reflected in the linear coefficient applied to it, and it can be trained analytically without any hyperparameter using the *Ordinary Least Squares* (OLS) method [72].

Logistic regression [72] can be applied when the output is not a real number but an enumeration of categories, to compute the probability of getting a given output category given an input vector. *Binary regression* is logistic regression with two possible outputs while *multinomial logistic regression* is logistic regression with more than two outputs. If the categories are ordered, *ordinal logistic regression* can be applied.

However, unlike linear regression, logistic regression models can not be computed analytically. Instead they require iterative methods such as Iterative Reweighted Least Squares (IRLS) [91] or gradient-based solvers such as L-BFGS-B [236]. Iterative methods require hyperparameters for their stopping conditions, which diminishes the explicability of the model and raises the risk of over-fitting.

Ranking Functions

Ranking functions are a family of algorithms used in information retrieval to order documents belonging to a corpus according to their relevance to a query. This query can

be a word or a list of words.

The most well-known ranking function is *TF-IDF* [104]. TF-IDF is a numerical statistic reflecting the importance of a word to a document, in the context of a corpus. TF-IDF is defined in Equation 2.4 where t is a word and d is a document belonging to a corpus of documents D.

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D), \tag{2.4}$$

TF is based on the number of occurrences of the word in the document. Several formulas have been proposed, such as using the raw count directly or treating the presence or absence of a word as a boolean event. Applying a logarithm on top of the raw count of occurrences has been argued [198] to better reflect the diminishing returns of repeating the same term several times.

IDF is defined in Equation 2.5:

$$\text{IDF}(t, D) = \log \frac{|D|}{|d \in D : t \in d|}, \tag{2.5}$$

where $|D|$ is the number of documents in the corpus, and $|d \in D : t \in d|$ is the number of documents of the corpus containing the word t. TF-IDF therefore allows more specific words to have a bigger impact on the mapping than common words.

TF-IDF has spread beyond the information retrieval community and is widely used in many text analysis tasks across the machine learning community. It is often used in conjunction with bag-of-words at the featurization stage of a machine learning pipeline to transform a corpus of text documents into a set of real vectors with specific words weighted heavier than common words (e.g. *"the"*, *"a"*, etc.). We make use of TF-IDF in Chapters 4 and 6.

However, while TF-IDF is ubiquitous in machine learning, the information retrieval community argues [226] that it has been made obsolete by another algorithm named Okapi BM25 [166]. While TF-IDF has been found to work reasonably well in practice, it lacks theoretical justifications for its formula. Meanwhile Okapi BM25's formula is more complex but based on information theory. However Okapi BM25 requires two hyperparameters to be set, unlike TF-IDF which requires none. Okapi BM25 is used in TTPDrill [97].

Choosing between TF-IDF and Okapi BM25 in the context of decision explicability creates an interesting dilemma, as TF-IDF is simpler, more well-known, and has no need for hyperparameters, while Okapi BM25 has better theoretical justifications. This raises the question about whether decision explicability is about making scientifically sound

decisions even if they are complex to explain, or about keeping the decision process simple enough to be understandable by a layman.

Deep Learning and Other Advanced Techniques

None of the machine learning techniques presented here are particularly new. Most of the machine learning research community today is focused on *deep learning*: the use of *artificial neural networks* (ANNs) [171] to solve learning problems. While ANNs have existed since the 1950s, the abundance of today datasets and computing resources allowed modern ANNs to unlock a degree of accuracy well above any machine learning technique that existed before. This has revolutionized multiple research fields and industries in the last decade.

However there are two major problems when using deep learning in critical contexts such as security alerting. The first one is the inherent lack of explicability of large neural networks. Deep learning explicability is a very active topic of research [231, 100] but overall it is still an unsolved problem.

The second problem is that ANNs are vulnerable to *adversarial attacks*.

Definition 40 *Adversarial attacks happen when an attacker deliberately crafts a malicious input with the goal of misleading the model towards the wrong output.*

Since their initial demonstration in 2014 by Szegedy et al [194], mitigation of adversarial attacks against ANNs has been a very active research topic, with promising results [118] but no complete solution for the time being. While adversarial attacks are only a minor inconvenience in some fields, they create a very real risk in computer security contexts as motivated attackers will definitely exploit this type of weakness in defense systems [219].

These two problems lead us to a major decision for this thesis: as of 2020, we believe that deep learning is not yet suited for security alerting. However both deep learning explicability and adversarial attacks mitigation are very active research topics and it is conceivable this may change in the next decade.

Many more machine learning techniques (Support Vector Machines (SVMs) [46], Decision Trees [168], Random Forests [106] etc.) are more explicable than deep learning, but less than regression analysis. Some of them, such as SVMs, are also vulnerable to adversarial attacks [20]. As we do not use these techniques in this thesis, we omit them for the sake of brevity.

2.4.6 Machine Learning and the Vulnerability Life Cycle

To close this chapter, we combine two of our topics of interest, to investigate how machine learning has been used in the vulnerability life cycle research community to predict properties of vulnerabilities such as severity and probability of exploitation. For the reasons described in Section 2.4.1 we view explicability as an important property of robust security alerting and review the literature with this criterion in mind.

Explicability was an explicit design goal of Jacobs et al [102] when designing the EPSS prediction model, which attempts to give the probability of a given vulnerability to be exploited in the twelve months following its disclosure. To achieve explicability they settled on using logistic regression, which lets them provide a very transparent prediction model. However they made the choice to use non-public data sources for model training, which probably improves decision accuracy but hampers decision justification: for instance, according to the EPSS model the heaviest factor of vulnerability exploitation risk is for the software vendor to be either Microsoft or IBM. This is certainly an explanation on a superficial level, but the more interesting question of *why* were these two vendors weighted so heavily in the model is a black-box (Jacobs et al did make an effort to justify this aspect of the EPSS model with the following comment in their paper: "*The Microsoft variable is likely a reflection of the ubiquitousness of Microsoft products (operation systems, and desktop and servers applications), as well as a long history of being targeted for exploitation. The IBM variable appears to be related to a handful of exploited and widely used products being led by their flagship Websphere application*").

Neither Borzorgi et al [24] (who also tried to predict vulnerability exploitation prior to EPSS) or Khazaei et al [108] (who tried to predict CVSS scores from vulnerability descriptions) made any mention of explicability in their design goals. Bozorgi et al used SVMs while Khazaei et al compared multiple machine learning techniques such as SVMs, random forests, and fuzzy systems. None of these techniques exhibit strong explicability properties. Though Khazaei et al used a bag-of-words followed by TF-IDF for featurization of the vulnerability descriptions, they also used non-explicable techniques such as LDA and PCA for dimension reduction.

Some works have pushed exploitation prediction beyond software and hardware vulnerabilities, by trying to predict exploitation of entire websites and information systems at a time. Soska et al [188] used a custom classification algorithm and web crawler to monitor the entire world wide web, to predict in advance the probability of a given website to become malicious in the future. Liu et al [116] monitored publicly visible configura-

tion issues of many organizations (including open recursive resolvers, DNS source port randomization, BGP misconfiguration, untrusted TLS certificates, open SMTP relays, etc.) and used random forests to predict the probability of these organizations to suffer a security breach in the future. Neither Soska et al nor Liu et al made any mention of explicability as a research goal.

2.5 Conclusion

In this chapter we presented the state of the art of four distinct domains. Section 2.1 outlined how the security risk of an information system keeps evolving over time, making current cyber-defense approaches unsustainable. Section 2.2 illustrated the complete life cycle of software and hardware vulnerabilities, from the introduction of the vulnerability, to its disclosure, to the dissemination of a mitigation. Section 2.3 focused on a specific part of this life cycle: n-day vulnerabilities, which are newly disclosed vulnerabilities with heightened risk as they are not well known yet. Finally, in Section 2.4 we reviewed machine learning tools to be used for n-day vulnerability mitigation, with a specific focus on decision explicability.

The current state of the art in n-day vulnerability mitigation leaves us with a sense of urgency. The next major n-day vulnerability will likely cause billions of dollars in damage and may conceivably lead to fatal casualties as well, two things that have already happened in the past. Properly managing the constant flow of new vulnerability disclosures is a very hard problem: most of these vulnerabilities are benign, but any of them can be a major source of risk. A lot of organizations are vulnerable and have no meaningful ways to defend themselves. Outsourcing defense requires trust and this trust is undermined by the difficulty of evaluating and explaining many of the defense mechanisms proposed today by the practitioners and research communities. The cyberdefense community is facing a shortage of qualified workers leaving a lot of jobs unfilled and many workers overworked. Still, automation and machine learning are no silver bullets and misusing one or the other can easily degrade our defense mechanisms instead of improving them.

While the scale of the challenge is far beyond the work of one PhD thesis, in Chapter 3 we outline how our multiple contributions align together to help improving the current situation. First we propose contributions to improve the understanding of newly disclosed vulnerabilities through better and faster prediction of their inherent properties. Second, we enable security experts to train a prediction system to evaluate the risk and the appro-

priate response for future vulnerability disclosures in the context of a specific information system.

Chapter 3

Defending Information Systems Against N-Day Vulnerabilities at Disclosure

N-Day vulnerabilities create important risks for information systems, and have already caused fatal casualties and billion of dollars in damage in the past. The goal of this thesis is to better defend information systems against them. In this chapter we outline our strategy: quickly extracting new information regarding n-day vulnerabilities at their disclosure, then assessing if the vulnerability creates a risk for an information system to be protected. The quicker this information can be gathered, the faster the risk created by the vulnerability can be addressed. The free-form text description of a new vulnerability is the only data we reliably have access to at its disclosure. This means faster vulnerability management must start with automated analysis of the description using machine learning methods. This automated analysis should conclude by making a decision about whether to initiate a counter-measure against the vulnerability, depending on the estimation of the risk it creates to the protected information system.

As described in Section 2.1.1, many information systems have multiple stakeholders. This means that any solution we devise should work in the context of a Security as a Service (SECaaS) offering, where the security of the information system is ensured by a different organization than the one owning it. Therefore the transparency and explicability of all of our automated decision processes is paramount.

In Section 3.1 we show how better defenses against n-day vulnerabilities make economical sense for both information system owners and security providers, in the context of multi-stakeholder information systems in a SECaaS arrangement. In Section 3.2 we describe the overall architecture of our proposed analysis and reaction pipeline aiming to assess and mitigate the risk created by newly disclosed vulnerabilities. We conclude in Section 3.3.

3.1 Economical Relevance of Guaranteeing the Security of Information Systems over Time

As seen in Section 2.1 the need for information security far exceeds the number of available skilled cybersecurity workers. This means hiring dedicated security staff is out of reach of many organizations that are still seeking some sort of information security. Therefore providing security to as many organizations as possible can only come in the form of an outsourced service, with one organization acting as a *security service provider* and another one as a *client*. This has several implications.

First, in order for two organizations to enter a business relationship, they must agree on a *contract*. This contract usually stipulates some action to be taken (or result to be achieved) by one party in exchange for a retribution by the other party. If one party deems that the other has not fulfilled its obligations, it can challenge it in an appropriate judicial system. However this implies that both parties (as well as a third-party such as a judicial court) should be able to evaluate what is and what is not a fulfilled obligation according to the contract. Therefore security practices in a SECaaS context must inherently be more transparent than in internal contexts: it is not enough for a provider to keep its clients secure. A provider should also ensure that it can convincingly confirm it to either the client or a court.

Another implication is that two parties will not enter a business relationship if one party sees no benefit to it. Organizations will not pay for a SECaaS offering if they feel that is not worth the money. Security providers will not propose a SECaaS offering that does not make economical sense for them. Therefore for our solutions to be relevant in the real world, they must take into account the legal and economical context of the organizations willing to use them.

In Section 3.1.1 we propose a new type of Service-Level Agreement (SLA) aiming to encourage security providers to defend their clients against n-day vulnerabilities. In Section 3.1.2 we show that this SLA makes economical sense for all parties, and incentivizes security providers to improve the defense mechanisms they provide against n-day vulnerabilities.

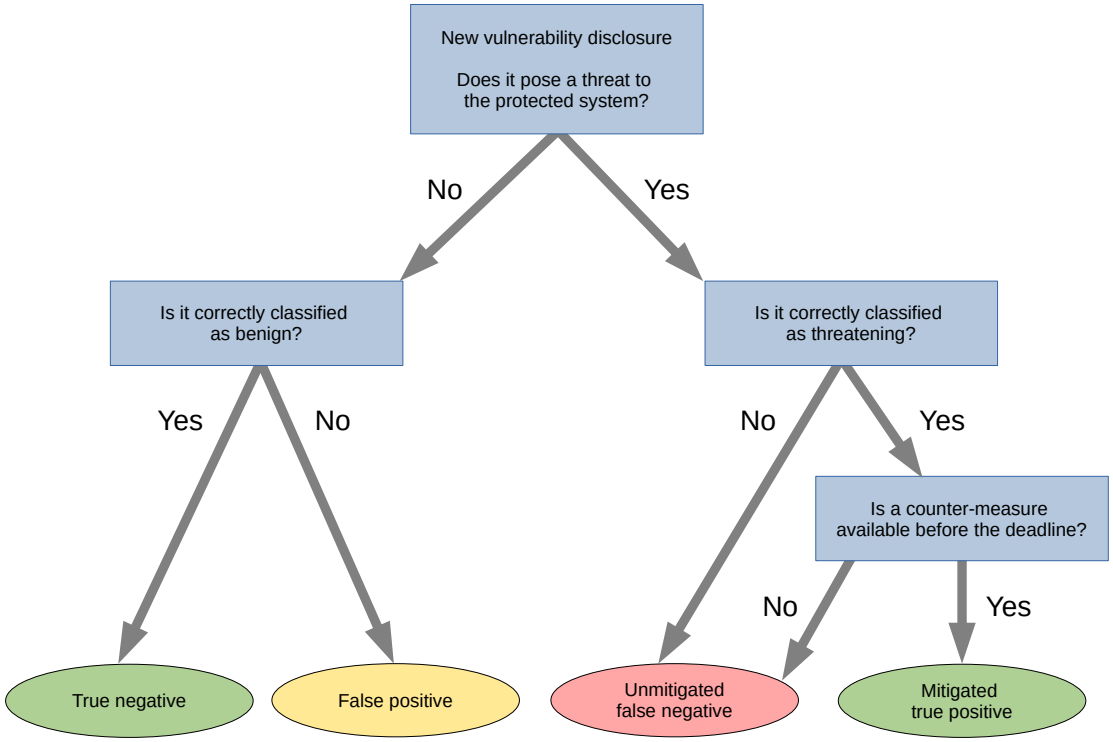

Figure 3.1 – The structure of our proposed SLA between a security provider and a client.

3.1.1 A Service-Level Agreement on Vulnerability Mitigation for Information Systems

We propose a new type of Service-Level Agreement (SLA) incentivizing security providers to provide mitigation plans for n-day vulnerabilities creating a risk for their clients, by incurring a financial penalty if they fail to provide one. The security provider must then analyze the financial risk of not being able to provide a mitigation plan under the deadline even when having correctly classified a vulnerability as threatening. The structure of our SLA allows the security provider to cover this cost in a probabilistic way.

Example of an SLA for N-Day Vulnerability Mitigation

As described in Section 2.1, the IT industry has long reconciled the business and technical aspects of IT services through the notion of *Service-Level Agreement* (SLA).

Therefore we first propose a very simple SLA for n-day vulnerabilities mitigation.

This SLA can be included in a contract between a security provider and a client which is responsible for an information system.

The security provider monitors all newly disclosed CVE vulnerabilities. Based on the knowledge it has of the client's information system, it estimates which newly-disclosed vulnerabilities the client should be concerned with, and has to propose a mitigation plan in a given delay after the disclosure date of the vulnerability.

For this service, the client pays a periodic charge (such as a monthly fee). However, if the provider misses the deadline and does not propose a mitigation plan for a vulnerability creating a risk (a *false negative*), it has to reimburse back a *penalty* fee. Assuming the provider's knowledge of the information system is imperfect, it is also possible for the provider to send a superfluous mitigation plan to the client (a *false positive*) which can also warrant a small penalty fee to prevent mitigation plan spamming. Figure 3.1 shows how such a contract and SLA might work.

It should be highlighted that the security provider is not tasked with *executing* the mitigation plan but merely devising it and notifying the client about it. If a client is notified about an n-day vulnerability and deliberately chooses not to execute the associated mitigation plan (for good or bad reasons), the security provider's responsibility is not engaged.

We also assume that the notifications and mitigation plans cannot be *repudiated*: neither party can forge an antedated notification or deny the existence of a legitimate one in court.

This setup is appropriate for a SECaaS context because it checks all the requirements of a good-quality SLA: it is based on a metric that is accessible and measurable by both parties (a public vulnerability disclosure event), the crossing of a threshold (a missed deadline for a mitigation plan) is an objective event witnessed by both parties, and evidence of a breach of contract can be reliably presented to an arbitrator if necessary.

Existing Counter-Measures for Vulnerability Mitigation

Nothing prevents such an SLA to be put in place today. A security provider willing to propose such an SLA could include a number of counter-measures in its n-day mitigation plans:

— The deployment of a corrective patch for the vulnerable software,

— The deployment of a new IDS signature detecting attacks exploiting the vulnerability,

— A workaround specific to the information system to be protected, such as a configuration change, the shutdown or network isolation of non-critical parts of the information system, etc.

However as seen in Section 2.3, none of these counter-measures are ideal. Corrective patches and IDS signatures may not be available within the deadline agreed upon by the provider and the client, and specific workarounds might not be feasible, or deemed too risky as they usually require an urgent yet manual operation on the production system. In the next section we see how this SLA incentivizes security providers to develop better counter-measures for n-day vulnerabilities through a financial reward.

3.1.2 Financial Impact of a Vulnerability Mitigation SLA for Information Systems Stakeholders

Even assuming that the service provides indisputable security value to the client, appropriate contract parameters still need to be chosen such that both parties are financially incentivized to sign it. We propose an evaluation strategy based on Fermi estimation [10] to evaluate the expected monetary value of this SLA for security providers, in two scenarios: by using currently available types of counter-measure, and by assuming the existence of better defense mechanisms. A variation of this estimation was presented at the French national conference *ComPas'18* [66].

Mitigation with Existing Counter-Measures

We can use Fermi estimation to devise a simple thought experiment. This experiment helps us to estimate the order of magnitude of the expected monetary value for a security provider proposing such a contract, assuming plausible parameters. We assume that:

— The only counter-measures proposed by the security provider are software patches.

— A deadline of 30 days after disclosure for the security provider to provide a mitigation plan to the client before enduring a financial penalty. Frei et al [74] estimated in 2010 that roughly 70 % of vulnerabilities had a patched available 30 days after disclosure.

— 100 vulnerabilities are disclosed every day. This is slightly higher compared to current levels, as more than 18000 vulnerabilities were disclosed by CVE [41] in 2018 and 2019, around 50 vulnerabilities per day on average. However the number is growing every year and 100 vulnerabilities per day will likely become the norm

	Scenario 1	**Scenario 2**
Parameters		
Number of Vulnerabilities in Period	3000	
Revenue for Period	1	
Vulnerability Threat Rate	0.1 %	
Classification False Positive Rate	0.1 %	0.5 %
Classification False Negative Rate	0.1 %	0.5 %
Counter-Measure Availability Rate	70 %	95 %
Penalty for Unmitigated Vulnerability	100 %	
Penalty for False Positive	2 %	
Results		
Expected Number of Dangerous Vulnerabilities	3	
Expected Number of Benign Vulnerabilities	2997	
Expected Number of False Positive	2.997	14.985
Expected Number of False Negative	0.003	0.015
Expected Number of Missing Counter-Measures	0.899	0.149
Expected Penalty for False Positive	0.06	0.30
Expected Penalty for Unmitigated Vulnerabilities	0.902	0.164
Expected Monetary Value for Period	**0.04**	**0.54**

Table 3.1 – Expected monetary value of the proposed SLA assuming plausible parameters.

in a few years.

— 0.1 % of these vulnerabilities pose a risk to the client's information system (3 per month on average). This hypothesis is rather arbitrary as the actual number highly depends on the size and heterogeneity of the information system to defend.

— Patch delay and the fact that the vulnerability creates a risk on the information system are two independent probabilistic events. This is likely a worst-case scenario compared to reality, which probably makes our estimation conservative.

— The security provider hires skilled human security experts to monitor all new CVE vulnerabilities, who can assert the risk posed by a vulnerability to the client's system with a false positive rate and false negative rate of 0.1 % each. This hypothesis is rather arbitrary as to our knowledge there is no literature on the accuracy of human security experts assessing the risk created by newly disclosed vulnerabilities.

— The client pays a monthly fee for the service.

— The security provider reimburses a month worth of fee for every false negative, and 1/50 of a month worth of fee for every false positive.

These propositions and their effects are summarized in Scenario 1 in Table 3.1. Under

these assumptions, the expected monetary value of the service for the security provider is actually positive. However more than 96 % of the security provider's revenues will get reimbursed back to the client as penalties, mostly because of false negatives.

One could wonder why the security provider should get a financial penalty because of a third-party software vendor failing to provide a patch for a security issue. This is by design, as this aligns the financial perspective of the security provider with the security of its client. It incentivizes the security provider to get creative with vulnerabilities when no counter-measure is readily available. For instance providers could fix issues in open-source software themselves, passing deals with software vendors to guarantee timely patches, or exploring alternatives mitigation solutions, which we evaluate in the next section.

Development of New Counter-Measures

Suppose our security provider proposes the same contract as above to its client, but through a fully automated solution. We now assume that the security provider uses a new mitigation technology allowing 95 % of vulnerabilities to be mitigated within deadline once being classified as dangerous (compared to 70 % previously). However, we assume that the automated threat analysis system is less accurate than the previous human expert judgment, and the false positive and negative rates have increased by a factor of five, to 0.5 %. This is Scenario 2 in Table 3.1.

Even with this degradation, the expected value for the security provider is vastly increased, with the projected reimbursment penalty fees going from 96 % of revenues to 46 %. The majority of the penalty fees is now due to false positives instead of false negatives, implying the client's system is now more secure than previously. This Fermi estimation shows that it is possible to align the financial incentives of a security provider with the actual security of its clients against n-day vulnerabilities, creating a positive feedback loop benefiting everyone.

3.2 Real-Time Assessment of Vulnerability Risk at Disclosure

In this section we detail the technical and research problems needing to be solved in order to estimate and react to the risk created by new vulnerability disclosures. In Section 3.2.1 we explain the impact of a vulnerability disclosure on the threat and risk

faced by an information system. In Section 3.2.2 we study the estimation of the risk created by the disclosure of a new vulnerability. In Section 3.2.3 we discuss possible real-time reactions to this risk, and in Section 3.2.4 we present the open problems implementing these reactions implies.

3.2.1 Impact of Vulnerability Disclosure on Threat and Risk

As seen in Section 2.2.1, the threats faced by an information system are the attackers attempting to compromise it, with varying degrees of skills, motivation, and resources. The risk faced by the information system is the combination of the threats it faces, the vulnerabilities it is affected by, and the value of the assets it contains.

While not every organization will face *dedicated* attackers, there are many attackers attempting to *indiscriminately* compromise systems visible from the Internet. Historically a recurring motivation has been the constitution of *botnet*s (networks of compromised hosts that can be used for dedicated attacks such as distributed denial of service). More recently attackers have also been motivated by financial gain through the use of *ransomware* [234] (a type of malware which encrypts the files of its host then asks for a crypto-currency ransom in order to get the decryption key), or *crypto-miner malware* [222] (where the attacker uses the computing resources of the compromised system to gain new crypto-currency through mining). Therefore any system connected to the Internet faces a baseline of threat even in the absence of dedicated attackers.

The disclosure of a software or hardware vulnerability does not create additional information systems vulnerabilities. After all, the latent vulnerability was already there before the disclosure. The value of the assets in the information system does not change either.

However, this vulnerability disclosure can still cause an evolution of the threats faced by the system for two reasons. First, it informs existing attackers of new ways to potentially compromise the system. Finding and exploiting zero-day vulnerabilities is a difficult and expensive task for all but the most sophisticated attackers, but many n-day vulnerabilities can be used rather easily on unprotected systems once they are publicly disclosed. Therefore even unsophisticated attackers can become dangerous if they are persistent, through the timely use of a recent n-day vulnerability.

Second, it can motivate new attackers to act for opportunistic reasons, such as financial gain (through ransomware or crypto-miners for example) or even mere vandalism. This creates new threats one has to defend against.

As risk is the combination of threat and vulnerabilities, an evolution of threat does lead

to an evolution of risk. Therefore new vulnerability disclosures do increase the risk faced by a system. Applying immediately and automatically counter-measures to dangerous newly disclosed vulnerabilities would effectively smooth out this risk towards a stable baseline level. However as outlined in Section 2.2 and 2.3, vulnerability information at disclosure is unstructured and incomplete, making it difficult to properly assess the risk. Therefore we need automated information extraction techniques in order to gather and structure as much intelligence as we can regarding the vulnerability, in order to make a decision about the risk it creates to the systems to defend.

In this thesis our goal is to help security providers of a given information system answer several questions about a newly disclosed vulnerability and its relationship with the information system:

— What is the severity of the vulnerability?
— Which software or hardware component does this vulnerability affect?
— Is this component part of the information system?
— Does this vulnerability increase the risk on the information system?
— Does this vulnerability need to be mitigated?
— What are the relevant counter-measures to mitigate this vulnerability?

In Section 3.2.2 we go further in depth regarding these questions and the way they can be automatically answered at disclosure.

Once the security provider has gathered enough information and asserted the existence of a risk, it can react to the disclosure by deploying a counter-measure. As seen previously, multiple counter-measures are possible. They include automatically deploying a corrective patch, dynamically adjusting the threshold factor of an IPS, and many other possibilities. We discuss all these counter-measures in Section 3.2.3.

3.2.2 Risk Evaluation at Vulnerability Disclosure

In this section we outline how to better evaluate the risk created by vulnerability disclosures by making an exhaustive list of all the unmitigated n-day vulnerabilities affecting an information system. We then describe two open research problems that must be solved in order to make progress on this task.

Listing Unmitigated N-Day Vulnerabilities as a Measure of Threat and Risk Level Evolution

In order to defend an information system, one could aim to list every public vulnerability related to the system that have not been mitigated yet (either because they have not been analyzed yet or because an analysis confirmed the presence of a risk). This list would give a relevant picture of the evolution of threat and risk for the system, though incomplete as threats can evolve for other reasons as we described in Section 2.1.2. This list is very difficult to maintain over time as many vulnerabilities are disclosed every day (as seen in Section 2.2.3). Automating this task is therefore desirable.

Unsolved Problem: Analyzing Vulnerabilities at Disclosure

As seen in Section 2.2, the state of the art in vulnerability management relies on human analysis. Nearly all vulnerability management systems are based on vulnerability metadata such as CVSS vectors or CPE URIs. This metadata is machine-readable yet written by humans. Therefore a large number of security experts are needed to author this metadata as analyzing a single vulnerability requires minutes, hours or even days.

We argue that human analysis of new vulnerabilities at disclosure is unsustainable. Figure 3.2 shows the number of vulnerability disclosures by year since 2007. We can see that in 2017 the number of annual vulnerability disclosures more than doubled compared to 2016 and the number keeps growing every year. It seems that the trend will continue in 2020: during the first semester 9810 vulnerabilities have been disclosed compared to 7338 on the same period in 2019, a 34 % increase.

NVD, which is the only organization attempting to provide a publicly available analysis for every CVE vulnerability, has arguably been caught off-guard by this increase. Figure 3.3 shows the evolution of the NVD analysis delay: the number of days between the vulnerability disclosure by CVE and the publication of its analysis by NVD. We can see that the median and 9th decile analysis delay have considerably increased in 2017 and 2018, with a median analysis duration of 35 days in 2018 (compared to 0 days from 2007 to 2015) and a 9th decile of 63 days (compared to 2 days from 2007 to 2012). While NVD should be credited with reversing the trend in 2019, the delays are still considerably higher than in the earlier years of its existence. This analysis delay makes many current vulnerability analysis techniques obsolete as they assume the immediate availability of vulnerability metadata, which cannot be taken for granted anymore.

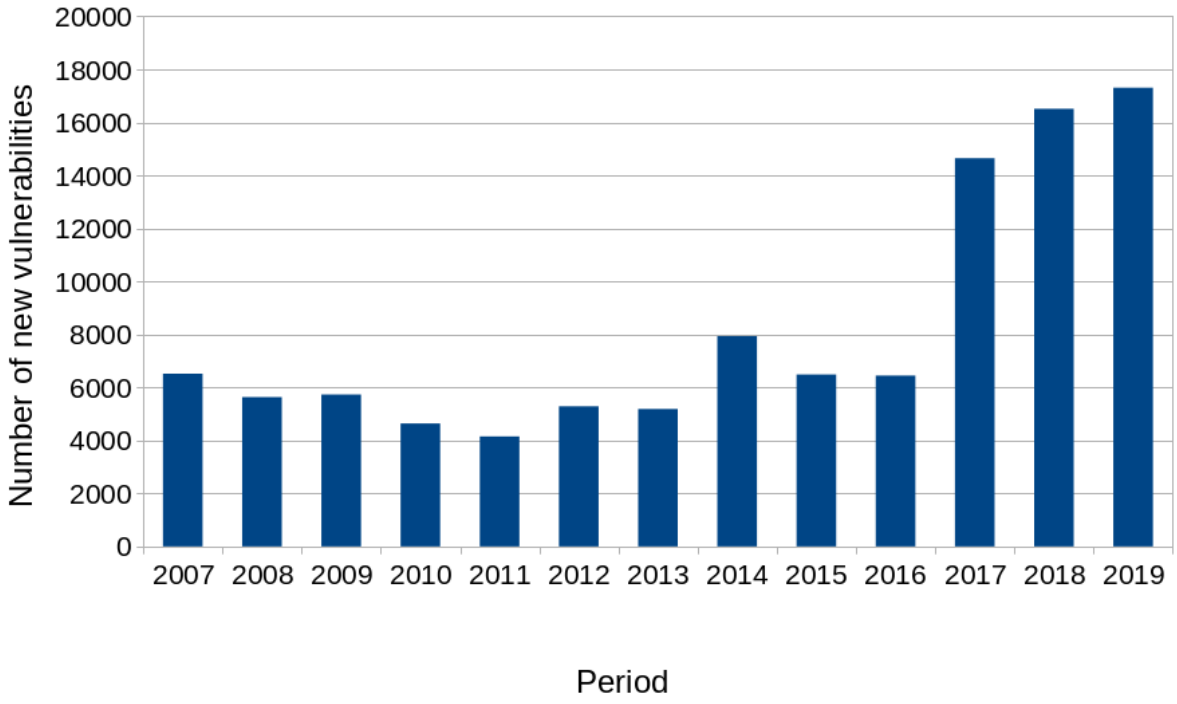

Figure 3.2 – Number of CVE vulnerability disclosures from 2007 to 2019.

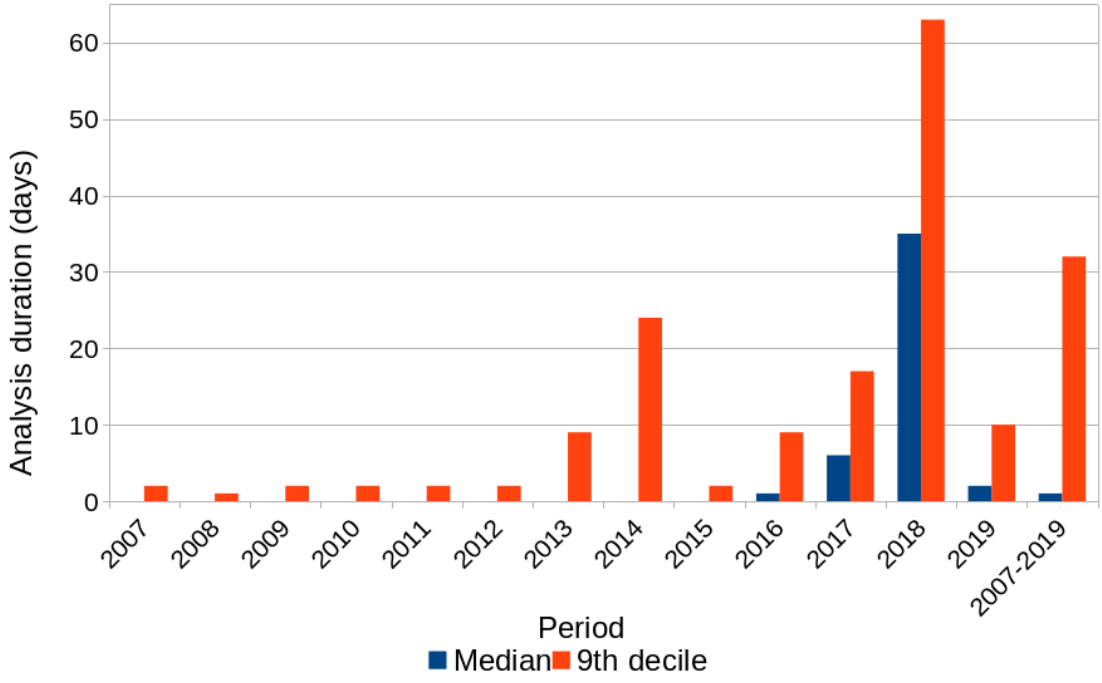

Figure 3.3 – Delay between vulnerability disclosure and analysis availability in NVD from 2007 to 2019.

Description			
Stack-based buffer overflow in the brcmf_cfg80211_start_ap function in drivers/net/wireless/broadcom/brcm80211/brcmfmac/cfg80211.c in the Linux kernel before 4.7.5 allows local users to cause a denial of service (system crash) or possibly have unspecified other impact via a long SSID Information Element in a command to a Netlink socket.			
CVSS V3 Base Vector			
CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:H			
CVSS V2 Base Vector			
AV:L/AC:L/Au:N/C:N/I:P/A:C			
CVSS V3 Base Score	6.1	**CVSS V2 Base Score**	5.6

Table 3.2 – Description, CVSS V2 and V3 base vectors and scores for vulnerability CVE-2016-8658, about a not-fully characterized buffer-overflow in the Linux Kernel.

One could understandably wonder if this is a research problem or an organizational problem between Mitre (which oversees CVE) and NVD. As seen in Section 2.2.2, CVE vulnerability disclosures are prepared by security experts working for CVE Number Authorities (CNAs) and these experts are well-placed to provide vulnerability metadata along the free-form text description they currently write. And indeed, the NVD CVMAP initiative [144] announced in June 2020 aims to let CNAs experts directly provide vulnerability metadata themselves in the NVD database if they have authored it while working on the CVE disclosure. So part of the problem complexity is *accidental* and will eventually be solved at the organizational stage.

However another part of the complexity is *essential* and in our view justifies to consider the problem as a research problem. First, many vulnerabilities are not fully understood at disclosure even by their discoverers: some are discovered through source code analysis (manually or automatically) and many teams managing critical software (such as the Linux Kernel team) treat any suspect buffer-overflow as a potential vulnerability by default. They then fix it and disclose it as soon as possible, even if the actual impact of the vulnerability has not been clearly defined yet. CVE-2016-8658 shown in Table 3.2 is a typical example of this phenomenon. A second aspect of this problem is that not all disclosures are executed smoothly. While CVMAP will soon encourage CNAs to share vulnerability metadata at disclosure, they will not be obligated to do so. Therefore many CVEs will still be disclosed without metadata. Even if all CVEs were properly disclosed, sometimes a vulnerability discoverer bypasses the CVE network and brutally discloses a zero-day vulnerability publicly. In other cases a security researcher will detect active ex-

ploitation in the wild of a previously unknown zero-day vulnerability, requiring immediate disclosure and mitigation even though the exact impact of the vulnerability has not been properly assessed yet.

This means that both in the short term and long term, working with incomplete vulnerability data at disclosure is a necessity. When analyzing a newly disclosed vulnerability, the only information we can reliably count on is the English free-form description of the vulnerability. However for past vulnerabilities we do have access to both their descriptions and their metadata, opening the door to machine learning-based predictions of vulnerability information at disclosure. Real-time automated analysis is therefore a promising candidate tool to characterize the inherent properties of a vulnerability faster and cheaper than using human experts.

We propose two contributions that use text mining techniques to reconstruct machine-readable vulnerability information from its English description. The first contribution, detailed in Chapter 4, is focused on assessing which software is affected by a newly disclosed vulnerability. The second contribution, detailed in Chapter 5, is focused on assessing the severity of the vulnerability through automated prediction of its CVSS vector.

Unsolved Problem: Automated Threat and Risk Evaluation for a Vulnerability and a Specific System

So far we discussed about predicting the *inherent* properties of a vulnerability such as its severity, which is independent of the context. However organizations are more concerned with the *risk* the vulnerability creates for them, which is only loosely related to its absolute severity. Automatically assessing the impact of a vulnerability disclosure on the threat and risk level of an organization requires solving two problems.

First, the automated evaluation system needs to automatically assess if a vulnerable version of the affected component is present in the information system. This requires knowing which hardware and software (including version number) are present in the information system, and having a ground-truth enumerating all vulnerable versions of a component.

Second, if the evaluation system is confident that the vulnerable component is present in the system, it needs to evaluate if the vulnerable component results in an *information system vulnerability*, allowing attackers to actually compromise the system. This is dependent on the way the component is used in the information system, the way the vulnerability can be exploited, and the impact it has once exploited.

If these questions cannot be answered in an absolute way, there are other related questions that may be easier to answer yet providing a useful approximation: for instance, is it possible to predict the reaction of a human security operator tasked with protecting the same system if she was faced with this vulnerability? Would she ignore the vulnerability or try to deploy a counter-measure against it?

This is important as the first two problems are difficult. As seen in Section 2.3, correctly determining which software components are installed in a complex information system is non-trivial. Listing existing binaries on every host is not enough as there are many runtime-based software platforms such as web browsers, the Java Virtual Machine, etc. Getting the proper version number of all this software is difficult, and this information can become obsolete at any moment as software is frequently upgraded without notice. Moreover this requires a correct ground-truth for the affected versions of the vulnerable component, which is a part of the vulnerability properties that are not available at disclosure in a machine-readable format. Meanwhile, the version information given in the raw text description of the vulnerability is often ambiguous or even incorrect (see Section 2.3.2).

For these reasons it is usually neither possible to have a complete picture of an information system to protect, nor possible to list all public vulnerabilities present in the system exhaustively, neither at disclosure nor after. And even if it is possible, the way a vulnerable software component is actually used can make the vulnerability critical or benign independently of the vulnerability inherent severity. Therefore analyzing the risk created by a vulnerability disclosure on an information system is very difficult, even for human experts.

We argue that getting rid of human experts completely for this risk analysis is both impractical and risky. However we can use machine learning methods such as active learning (see Section 2.4.3) to allow an automated system to imitate the decisions taken by a human expert responsible for evaluating the risk created by new vulnerability disclosures. This would allow for a real-time protection of an information system even when no human experts are on-call. We propose a contribution on this topic in Chapter 6.

3.2.3 Real-Time Reaction to Vulnerabilities Disclosure

Once an automated risk analysis pipeline has determined that a new vulnerability disclosure creates a risk for the system to protect, a reaction to this information is needed so that the risk can be mitigated. In this paragraph we outline several appropriate reactions,

from the least to the most automated one: raising an alert, triggering out-of-band security updates, automatically elevating log levels, switching into a degraded mode, and using a dynamic IPS to balance confidentiality, integrity and availability.

Raising an Alert

A non-automated yet relevant reaction is simply to alert an on-call human security operator. We propose a contribution in this regard in Chapter 6. While such an approach does not allow faster-than-human reaction time, it still allows a human to react in hours or even minutes, instead of days or weeks (or never) which is common.

There are two main limitations to this approach. First, it still leaves a significant workload to human security operators, who are already overworked. It also assumes that there is an on-call operator available for the system, which is not always the case. Second, this approach cannot decrease the reaction time beyond human capabilities. As seen in Section 2.3.1 and Appendix A.2, the Shellshock vulnerability made most Unix system with bash shell vulnerable, and the first recorded attack occurred less than an hour after disclosure. Simply raising an alert is not enough against such impactful n-day vulnerabilities. Therefore we now explore more automated reactions.

Automated Security Updates

A possible automated reaction is to lean on a software package manager (such as *apt* in Debian-based Linux distributions) to automatically install security patches for all computer hosts of the information system. While it is common for such security updates to be applied regularly (either manually or automatically), a possible reaction to a vulnerability disclosure is to accelerate the update check frequency, in order to install a security patch as soon as it becomes available. This strategy is also applicable to signature-based IDSs that use upgradable signature databases [195].

There are two main limitations to this approach. First, this reaction only makes sense if a patch (or an IDS signature) is rapidly made available for the vulnerability (Frei et al [74] estimated in 2010 that only 50 % of vulnerabilities had a corrective patch available at disclosure). Moreover, the patch needs to be made available in the package repository used by the information system, which can be another source of delay. Second, automated patching of production systems can introduce bugs and regressions in the event of a flawed patch. This can be partially mitigated through excellent continuous deployment practices, but not every administration team has the time or skill to put those in place.

Moreover those practices do not fully eliminate the risk: even cloud providers following best practices have endured outages due to faulty software updates in the past [2].

Automated Log Elevation

Another possible automated reaction to a vulnerability disclosure is the immediate elevation of logging levels for various components of the information system, to record events at a finer granularity for some period of time [12]. This can be instrumental in giving future forensic investigators enough information to reconstruct the kill chain of a potential attack after the fact.

There are two main limitations to this approach. First, obviously this reaction alone does not mitigate the attack itself, it only allows to study it afterwards; though in the presence of on-call security operators, it may also help in detecting an attack in real time. Second, keeping an elevated log level for too long creates many log entries needing to be stored. This may either saturate the storage components of the information system, creating a risk of cascading failure, or if log rotation is enabled, it risks erasing meaningful logs as the available space for log storage was not dimensioned for sustained elevated logging. Moreover too many logs can become an impediment for security operators tasked with analyzing them. Careful timing of the reaction duration is therefore needed.

Automated Switch into Degraded Mode

Information systems with critical safety requirements can sometimes be designed with two modes of operation: a nominal mode and a degraded mode where non-critical components of the system are deactivated, shut down, or isolated from the network. In degraded mode, the attack surface of the system is reduced, limiting the probability of a successful compromise. For instance, a hospital information system could be designed with a degraded mode shutting down every non life-critical computer systems in order to avoid worm attacks such as WannaCry or NotPetya (discussed in Appendix A.3). In this regard, a possible automated reaction to a dangerous vulnerability disclosure is the automated activation of the information system degraded mode. In situations where long-term confidentiality and integrity of the system prevail on its short-term availability, a simple but radical degraded mode is to isolate the entire system from the Internet at the network level.

There are three main limitations to this approach. First, information systems are complex, and complexity is a source of insecurity. Having multiple modes of operation

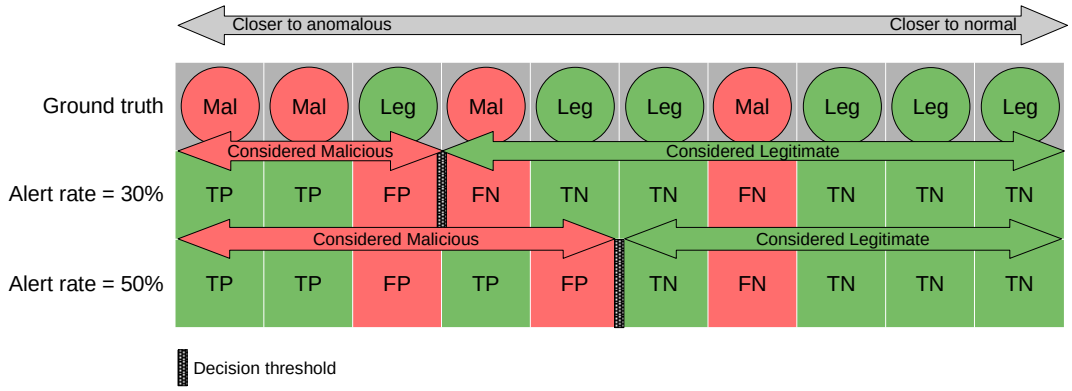

Figure 3.4 – Anomaly-based IDSs need a decision threshold to convert a continuous normalcy evaluation into a binary decision. This decision threshold can be set as an alert rate parameter, with a lower alert rate increasing false negatives and a higher alert rate increasing false positives.

increases this complexity. Moreover, depending on the type of services provided by the system, a degraded mode may not even make sense. Second, determining if the vulnerable component will actually be protected when switching to degraded mode is non-trivial. Therefore switching into degraded mode does not necessarily prevent a successful compromise. However it does hamper the lateral movement ability of an attacker already in the system, which is a valuable defense-in-depth layer. Third, the components remaining online in degraded mode are usually the most critical ones in the entire system, and they remain vulnerable against an attack. In the previous example of a hospital information system, the only computer hosts remaining online during degraded mode would be the machines supporting life-critical functions. Yet these machines are precisely the ones that need to be kept safe the most.

Using an IPS to Balance Integrity, Confidentiality and Availability

As seen in Section 2.3.3, a network-based *Intrusion Detection System (IDS)* analyzes incoming network packets and assesses if those are legitimate or malicious. They can work using detection signatures, or by anomaly detection. When an IDS is given the ability to block the packets it considers malicious, it becomes an *Intrusion Prevention System (IPS)*.

Network-based IPSes offer an interesting duality: a false negative can impact the con-

fidentiality and integrity of the system by letting an attack pass through, while a false positive can impact the availability of the system by blocking legitimate traffic.

Moreover, there is an inherent relationship between the *alert rate* (AR, Equation 3.1) of an IDS or IPS, its *false negative rate* (FNR, Equation 3.2) and its *false positive rate* (FPR, Equation 3.3). This is especially true for anomaly-based IDSes and IPSes, which usually evaluate the normalcy of an incoming packet on a continuous scale, as shown in Figure 3.4.

$$\mathrm{AR} = \frac{\mathrm{TP} + \mathrm{FP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}} \tag{3.1}$$

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}} \tag{3.2}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}} \tag{3.3}$$

IDSes and IPSes have to take a binary decision: an IDS has to decide whether to raise an alert or not, and an IPS has to decide whether to block a packet or not. The higher the AR, the lower the FNR and the higher the FPR. Conversely, the lower the AR, the higher the FNR and the lower the FPR.

When using an IPS, we can treat its alert rate as an availability budget for a given period. This availability budget can be allocated in the form of an SLA. If the risk level evolves over time, because of vulnerability disclosures in our case, this availability budget is better spent by increasing the alert rate during periods of higher risk and decreasing it when the risk is mitigated.

Some anomaly-based IDS and IPS designs do not provide an actual alert rate parameter. However nearly all of them first compute a normalcy score for each network packet then let the administator define a threshold to separate legitimate and malicious behavior (see Figure 3.4). In that case, the alert rate can be dynamically adjusted using a control feedback loop taking the observed alert rate as an input and the normalcy threshold as an output.

3.2.4 Real-Time Reaction: Open Problems

The reactions proposed above require some problems to be solved before becoming practical. These include synthesizing the perceived risk, comparing it to an expected

baseline risk, and an unsolved research problem: evaluating anomaly-based IDSes and IPSes.

Synthesizing the Perceived Risk as a Single Metric

Some reactions, such as raising an alert, can be initiated on a case-by-case basis for each individual vulnerability. Other reactions, such as switching the system into degraded mode, have to be decided globally for the entire system at once. In this case, we have to synthesize the *overall risk* posed by all non-mitigated n-day vulnerabilities taken together. Some reactions require a binary decision (for instance, to switch or not to switch into degraded mode) while others require a numerical metric (such as the alert rate of an IPS).

A boolean decision can be viewed as a numerical value coupled with a threshold. Therefore, computing the overall risk level as a numerical value is necessary in many cases. Non-mitigated n-day vulnerabilities provide a great basis for this numerical value, but it is possible to incorporate other factors into the risk level, such as geopolitical risk indexes we described in Section 2.1.2. However the more metrics are incorporated into the risk level, the more difficult it is to evaluate whether the measured risk level is actually realistic and useful. If not the risk level becomes a security theater metric, bringing a false sense of security uncorrelated with reality.

Current Risk and Expected Baseline Risk

In order to decide whether to launch a global reaction, it is not enough to have a measure of the current overall risk level. We also need a baseline, an expectation of what the overall risk level is supposed to be in the current situation, assuming no new vulnerability disclosure. The decision to react is then taken by comparing the currently measured risk level to the expected baseline risk level.

A baseline is needed for both binary and continuous reactions. For binary decisions, the baseline can be used to compute the threshold for initiating the reaction. For continuous reactions, such as an IPS converting the risk level into an availability budget, the expected risk level is necessary to decide how much the IPS should presently over-spend or under-spend its budget depending on the current circumstances.

Some reactions require a notion of budget per period. For instance our dynamic IPS proposal has a limited number of network packets it is allowed to block, and an alert system may have an alert budget for on-call operators notification. In those cases a pending

question is what to do when the decision process runs out of budget for the given period. We should point out that a budget shortage can happen for both good and bad reasons: the decision system may have successfully detected and mitigated an actual but exceptional risk and kept the information system safe in extraordinary circumstances, or the risk may have been miscalculated and the budget wasted. In both cases, one need to decide if the system should be allowed to temporarily exceed its budget or be shut down for the remaining of the budget period. For reactions such as a dynamic IPS, this creates another dilemma between the long-term confidentiality and integrity of a system and its short-term availability.

Unsolved Problem: Evaluating Anomaly-Based IDSes and IPSes

As outlined in Section 2.3.3, a major obstacle to our dynamic IPS proposal is that evaluating anomaly-based IDSs and IPSs is still an open research problem [217, 88]. As we cannot evaluate how well a regular IDS performs, we have no baseline to compare our dynamic IPS with. Moreover, even if the research community succeeded in authoring a proper IDS evaluation dataset, we would have an additional requirement: having attacks being realistically correlated with vulnerability disclosures. While the relationship between attacks in the wild and vulnerability disclosures have been studied (as seen in Section 2.2.3), the research community is far from having a comprehensive model of how they interact.

We know that the base rate of attacks in typical network traffic is very low and that most vulnerabilities are not exploited at all, which means that a dataset correlating attacks and disclosures would need to cover an extended period of time to include multiple dangerous vulnerability disclosures. It is likely that such a dataset would be of an unpractical size. We do not propose a dynamic IPS design in this thesis, first because of lack of time, second because the obstacles for evaluating such a work would make it difficult to publish as scientific research.

3.3 Summary

In this chapter, we showed a new type of SLA between a security provider and a client that could help limiting the risk created by n-day vulnerabilities against the client's information system, while being profitable for the security provider. By focusing on these two goals at once, we can initiate a positive feedback loop providing sustainable security

for more organizations around the world. We aim to create a risk evaluation system that automatically evaluates the risk created by a new vulnerability disclosure on a given information system. This risk evaluation system can be the basis for many real-time reactions aiming to reduce the risk endured by an information system. In this chapter we proposed several counter-measures to be used as reactions against dangerous vulnerability disclosures.

Before achieving this goal, we first need to solve two open problems. The first problem is to gather better information about a vulnerability in the seconds after its disclosure, before it is proposed as machine readable metadata by security analysts. The second problem is is to analyze how much risk a vulnerability disclosure creates for a given information system, and evaluating if this disclosure warrants a reaction or not.

In this thesis we propose two contributions on the first problem, and one contribution on the second one. In Chapter 4, we propose a solution to automatically determine the software or hardware affected by a new vulnerability using only the free-form text description available at disclosure. In Chapter 5, we use again this text description to assess the severity of a vulnerability at disclosure through automated prediction of its CVSS vector. Once we have gathered enough information about a vulnerability, the risk analysis of how a vulnerability disclosure impacts a given information system can be automated. In Chapter 6, we propose an automated decision process to evaluate the need to raise an alert to an on-call human security operator regarding a new vulnerability disclosure in the context of a specific information system.

Chapter 4

Extracting Relevant Keywords from Vulnerability Disclosure Using TF-IDF

In Chapter 3, we outlined that automated vulnerability analysis at disclosure was an unsolved problem. In this chapter we propose a first contribution towards solving this problem: identifying the affected software of a vulnerability by extracting the relevant keywords from its text description. To this end we use machine learning tools we presented in Chapter 2, such as bag-of-words and TF-IDF. We particularly focus on the reliability of our results in two ways: first by ensuring that our prediction pipeline is explicable, and second by designing a strong evaluation protocol to precisely evaluate how variations of the prediction scheme can impact its accuracy and its failure modes. This chapter content was published in the *2020 IEEE/IFIP Network Operations and Management Symposium* (NOMS 2020) [64].

We detail our objectives for this contribution in Section 4.1. In Section 4.2 we present the keyword extraction pipeline that we propose to fulfill these objectives. In Section 4.3 we present our evaluation protocol and results. We conclude in Section 4.4.

4.1 Objectives

We want to design an automated technique allowing us to identify the affected software of a vulnerability based on its raw text description, without the need for any human-authored metadata. Beyond accuracy we have one major goal for this work: to make the prediction of the affected software reliable enough to be used in a production environment.

In our view reliability comes from answering two questions: first, *how likely is the prediction to fail?* This can be answered by designing a thorough evaluation protocol identifying not only the accuracy of the prediction but its failure modes as well. Second,

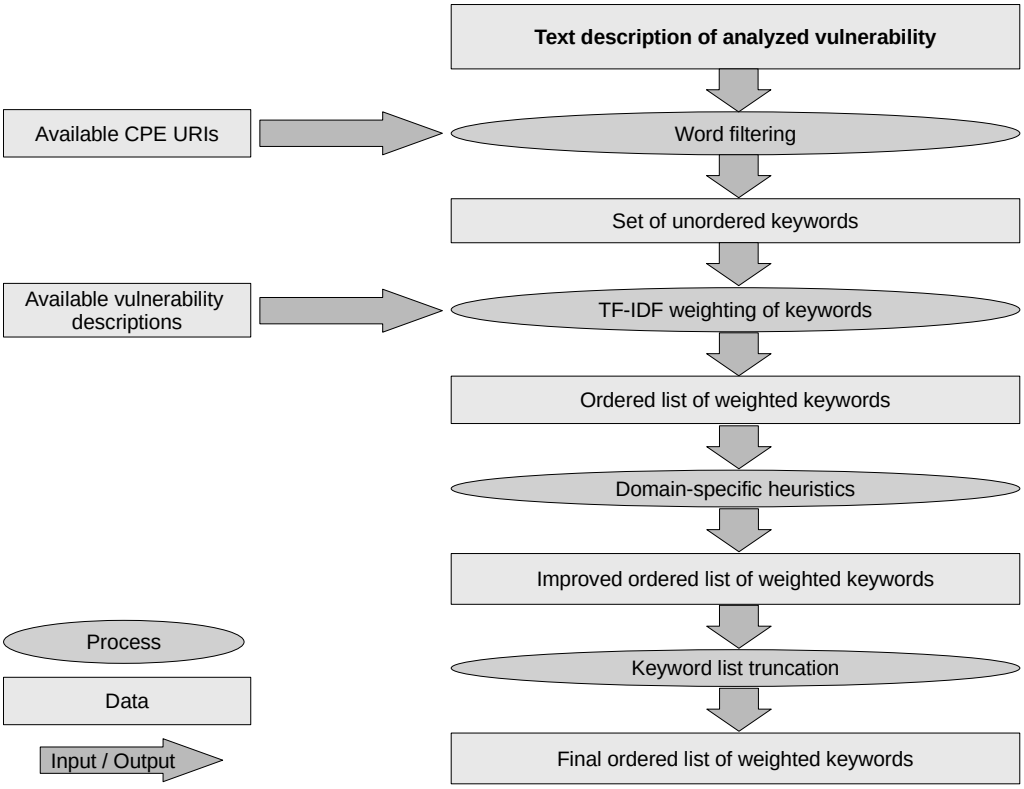

Figure 4.1 – Overview of the vulnerability description processing pipeline.

why did a prediction fail? This can be answered by keeping the analysis pipeline explicable at all stages, which has a profound impact on the choice of the machine learning techniques we can use. Explicability is important for security systems as most organizations require security incidents to be audited and their root cause understood. When such an incident occurs because of an incorrect decision from an inference system, the decision process of the inference system becomes the root cause to be analyzed. If the failure mode of the inference system cannot be understood, the correct course of action to prevent future occurrences of the incident is not to rely on the inference system anymore, often leading to its decommissioning. Therefore, our keyword extraction pipeline must be explicable at all stages, from the input to the output.

In the next section we describe our proposed pipeline to fulfill these two objectives.

CVE ID	**Disclosure date**	**Description**	
CVE-2016-6808	04/12/2017	Buffer overflow in Apache Tomcat Connectors (mod_jk) before 1.2.42.	
		Weight	**Keywords**
		27.54	tomcat connectors
		12.41	mod jk
		11.01	connectors
		8.37	tomcat
CVE-2017-0155	04/12/2017	The Graphics component in the kernel in Microsoft Windows Vista SP2; Windows Server 2008 SP2 and R2 SP1; and Windows 7 SP1 allows local users to gain privileges via a crafted application, aka "Windows Graphics Elevation of Privilege Vulnerability."	
		Weight	**Keywords**
		18.46	windows server 2008
		12.30	windows vista
		12.28	windows 7
		12.18	graphics
		11.84	server 2008
		11.74	windows server
		8.70	windows
		5.95	r2
CVE-2015-3421	07/21/2017	The eshop_checkout function in checkout.php in the Wordpress Eshop plugin 6.3.11 and earlier does not validate variables in the "eshopcart" HTTP cookie, which allows remote attackers to perform cross-site scripting (XSS) attacks, or a path disclosure attack via crafted variables named after target PHP variables.	
		Weight	**Keywords**
		26.29	eshop plugin
		19.10	eshop
		12.33	checkout php
		5.71	wordpress
CVE-2015-5194	07/21/2017	The log_config_command function in ntp_parser.y in ntpd in NTP before 4.2.7p42 allows remote attackers to cause a denial of service (ntpd crash) via crafted logconfig commands.	
		Weight	**Keywords**
		14.69	ntp
		5.07	y
		3.50	parser
		3.44	config

Table 4.1 – A sample of keyword extraction results, using our analysis pipeline (with all heuristics enabled and a keyword list truncation target of 95 % of the norm).

4.2 Our Approach

In this section we present our keyword extraction pipeline aiming to identify the software or hardware affected by a given vulnerability. Its input is the free-form description of a new vulnerability. It is analyzed using all vulnerability descriptions and metadata available at the time of disclosure. It outputs an ordered list of keywords, where each keyword is given a weight representing its estimated relevance. As an intuitive example, Table 4.1 presents a sample of vulnerabilities, including their free-form description and the corresponding keywords extracted by our analysis technique. For the reasons highlighted in Section 4.1, we deliberately choose to avoid elaborated machine learning methods (such as deep learning) when their accuracy comes at the expense of explicability.

A high-level overview of the proposed vulnerability analysis pipeline is shown in Figure 4.1. We now describe each stage of the pipeline in more details, starting with our choice of data sources in Section 4.2.1, then the word filtering stage in Section 4.2.2, the TF-IDF weighting scheme in Section 4.2.3, the domain-specific heuristics we use to improve this weighting in Section 4.2.4, and finally the keyword list truncation stage in Section 4.2.5.

4.2.1 Data Sources Considerations

As seen in Section 2.2.2, the CVE and the CPE corpus are linked together using a metadata called the *CPE URI*. A CPE URI is a unique reference to a specific entry in the CPE database, i.e. a specific version of a piece of software.

It is tempting to consider these corpus as two relational tables linked together using a foreign key. However we discovered two drawbacks of this approach. The first one is that the life cycles of the two databases are very different, resulting in an ever-increasing number of dead CPE URIs entries referenced in vulnerabilities metadata that do not actually exist in the CPE database. Figure 4.2 illustrates the problem. While both databases were mostly kept consistent from 2011 to 2016, the inconsistencies grew substantially since 2017. We want to emphasize that this is an important problem that, if left unchecked, will greatly decrease the relevance and real-world usefulness of the CPE dictionary.

A second drawback is the lack of a date-of-inclusion field for CPE entries in the CPE database. While this would have no impact in a production system, it prevents us from properly evaluating our results using a journaled view of the corpus. In order to properly simulate an analysis at disclosure time, we want to only consider CVE and CPE

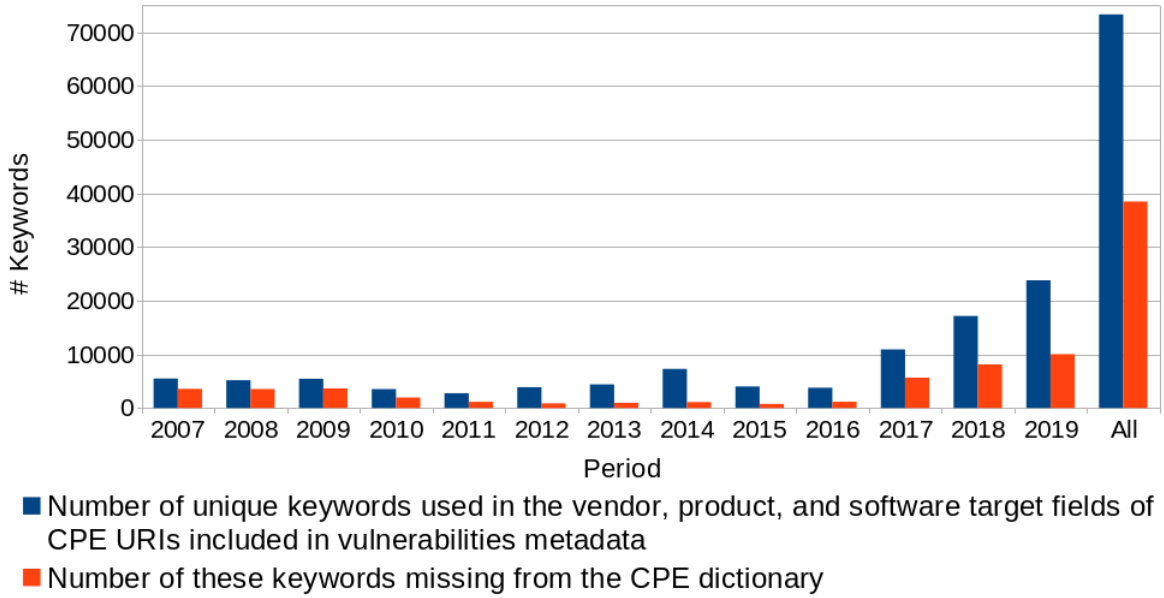

Figure 4.2 – Historical rate of software names used in vulnerabilities CPE URIs that are missing from the CPE dictionary.

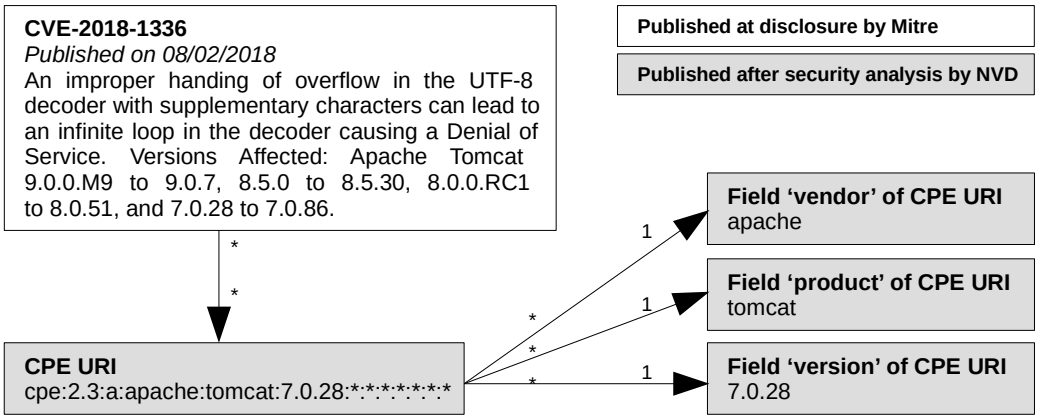

Figure 4.3 – When discarding the CPE dictionary we get a more robust data life cycle while retaining most of the inherent data.

Description	SQL injection vulnerability in register.php in GeniXCMS before 1.0.0 allows remote attackers to execute arbitrary SQL commands via the activation parameter.
Keyword set (in alphabetical order)	activation, before, commands, genixcms, in, parameter, php, register, remote, sql, the, to, via, vulnerability

Table 4.2 – Description and extracted keyword set for CVE-2016-10096, a vulnerability disclosed on 01/01/2017. The filtering list includes all CPE URIs published between 01/01/2007 and 12/24/2016.

published "in the past" compared to the disclosure date of the analyzed vulnerability. We solve both problems by discarding the CPE database completely and instead we use the data embedded in the fields of the CPE URIs, as described in Figure 4.3. As CPE URIs are part of a CVE vulnerability's metadata, we can reuse the date-of-publication field of the vulnerability for the included CPE URIs.

4.2.2 Bag-of-Words and Word Filtering

The first step of the pipeline applies the bag-of-words algorithm to all vulnerability descriptions, in order to view a description text as a set of word counts. Bag-of-words is described in Section 2.4.4 and is a very common text featurization algorithm that is simple and explicable. We then filter some of those words using CPE URIs. All available CPE URIs are parsed as a keyword filter list. Fields extracted from the CPE URI are the software vendor, software product, and target software. After extraction all these fields are tokenized into individual words. When analyzing vulnerabilities descriptions, only words belonging to this filter list are considered, the others are discarded. This filtering is a trade-off: it vastly reduces analysis noise but may filter out some relevant information. Specifically a new vulnerability affecting a never-seen-before software product will not have any relevant CPE URI in available historical data therefore the name of the product will be filtered out. However, we argue that this is the right trade-off in our context, as our goal is to feed keyword alerts to a security monitoring system: it is probably more relevant to output an alert because we did not find any highly relevant keyword than outputing an alert on a keyword that has never been seen before and is probably not monitored. This filtering gives us a set of keywords for every vulnerability. However this set is unordered and contains a lot of irrelevant keywords, as illustrated in Table 4.2. As we can see the filtering list includes very common words such as "before" that are not related to the present vulnerability ("before" was added to the filtering list through

Keyword list (after TF-IDF weighting)		**Keyword list** (after heuristics)	
Keyword	**Weight**	**Keyword**	**Weight**
genixcms	6.36	genixcms	12.71
activation	5.24	sql	5.46
register	3.40	activation	5.24
sql	2.73	register	3.40
commands	1.62	commands	1.62
parameter	1.22	parameter	1.22
php	1.19	php	1.19
vulnerability	0.57	vulnerability	0.57
before	0.56	before	0.56
the	0.21	the	0.21
in	0.18	in	0.18
remote	0.18	remote	0.18
via	0.12	via	0.12
to	0.01	to	0.01

Table 4.3 – Keywords order and weight for CVE-2016-10096, after TF-IDF weighting (left) and heuristics (right). The corpus includes all vulnerabilities disclosed between 01/01/2007 and 01/01/2017. The capitalization heuristic doubles the scores of "genixcms" and "sql" (spelled "GeniXCMS" and "SQL" in the description).

vulnerability CVE-2011-5107 affecting the Wordpress plugin *Alert Before You Post*). It is clear that the mere presence of a keyword in a vulnerability description is not enough to assess its relevance for the given vulnerability.

4.2.3 TF-IDF Weighting

Instead of treating the presence of a keyword in a vulnerability as a binary event, we weight each keyword by the term frequency-inverse document frequency (TF-IDF) [104] value of the word, in the context of the CVE corpus. The TF-IDF algorithm is described in Section 2.4.5. In our context, we consider the set of keywords extracted from a CVE description as an individual document, and the set of these sets as a corpus. We choose to use the logarithmic version [198] of TF, defined in Equation 4.1, as it better reflects the diminishing returns of repeating the same term several times.

$$\mathrm{TF}(t, d) = \log(1 + |t \in d|), \tag{4.1}$$

TF-IDF therefore allows more specific words to have a bigger impact on the mapping than common words. As an intuitive example, let us consider an actual software named

IBM Tivoli Service Request Manager. The word "Tivoli" is much more specific than "Request", therefore its weight should be higher. Every keyword in the set is now weighted, which allows us to order them by relevance. The left side of Table 4.3 gives an example of such weighting.

4.2.4 Domain-Specific Heuristics

A number of additional heuristics can be applied to the existing ordering to improve it even further. In this section we propose three of them which we detail below. We want to emphasize that an important part of our contribution is to give security experts the ability to formulate such kind of heuristics and reliably evaluate their accuracy. While these heuristics are very simple and domain-specific, we show in Section 4.3 that each of them increases the accuracy of the analysis.

Multiple-words keywords. When a CPE URI field contains entries that are multiple words long, such as "linux kernel", this heuristic treats "linux", "kernel", and "linux kernel" as three different individual terms with individual TF-IDF values. Therefore this heuristic allows the existence of keywords that are actually multiple words long. A match on "linux kernel" is considered more relevant than two separate matches on "linux" and "kernel", so this heuristic also multiplies the score of a keyword linearly by the number of words it is made of.

Capitalized words. We observed that software names in vulnerability descriptions are often capitalized. This heuristic doubles the score of every keyword that is capitalized in the vulnerability description. The software industry has a somewhat peculiar grasp of English capitalization rules, so this heuristic is triggered for any capitalized letter in a word and not just the first one: "iPhone" or "openSUSE" are considered capitalized.

Words starting by "lib". We empirically observed that words starting by "lib" that are not "library" are rare in English but are commonly used as software names (*libxml2*, *libssh*, *libpng*, etc.). This heuristic doubles the score of every keyword starting by "lib" that is not "library".

The right side of Table 4.3 gives an example of how applying all three heuristics alters the weights and order of keywords. We evaluate these heuristics in Section 4.3.

4.2.5 Keyword List Truncation

The list of keywords extracted from a vulnerability can be long. Assuming the analysis did a good job at sorting the relevant keywords first, the end of the list is empty of meaningful information. It is therefore desirable to truncate the list and only keep the beginning, as this operation retains most of the information while removing most of the noise. However it is not straightforward to decide where to cut. Keyword lists lengths vary greatly (from 0 to 196 keywords in our evaluation dataset) and the amount of relevant keywords inside them too. This makes static truncation threshold not appropriate, as it could remove too much information or retain too much noise. Instead we propose a dynamic truncation scheme based on the euclidean norm. At the truncation step of the pipeline, we view the untruncated weighted keyword list as a euclidean vector and we compute its norm. We then compute a truncation budget by defining a *target for the truncated norm*, such as staying above 95 % of the untruncated norm. Because most of the norm of the vector comes from the most relevant keywords, it is possible to cut out most irrelevant keywords while staying under budget. Table 4.4 gives a practical example of such a truncation. We evaluate experimentally the impact of keyword list truncation in Section 4.3.4.

4.3 Evaluation

In this section we present our evaluation protocol and results. In Section 4.3.1 we present the experimental setup. In Section 4.3.2 we present a first evaluation metric: the ranking of the first relevant keyword in the resulting keyword list. In Section 4.3.3 we present a second evaluation metric: the number of keywords necessary to fully reconstruct an affected software name. In Section 4.3.4 we evaluate the impact of the keyword list truncation scheme proposed in Section 4.2.5. We discuss the performance of our keyword extraction pipeline in Section 5.3.5. All the code and data used for our experiment are available at [69].

4.3.1 Experimental Setup

We analyzed all 31156 CVE vulnerabilities disclosed between January 1st, 2017 and January 1st, 2019, using all 57640 CVE vulnerabilities disclosed between January 1st, 2007 and December 31st, 2016 as past historical data. This experimental setup simulates

Untruncated keyword list		**Truncated keyword list** (target norm = 95 %)	
Keyword	**Weight**	**Keyword**	**Weight**
genixcms	12.71	genixcms	12.71
sql	5.46	sql	5.46
activation	5.24	activation	5.24
register	3.40		
commands	1.62		
parameter	1.22		
php	1.19		
vulnerability	0.57		
before	0.56		
the	0.21		
in	0.18		
remote	0.18		
via	0.12		
to	0.01		
Norm		Norm	
15.38		14.79	

Table 4.4 – Untruncated and truncated weighted keyword lists for CVE-2016-10096, with a target norm of 95 % for the truncated list.

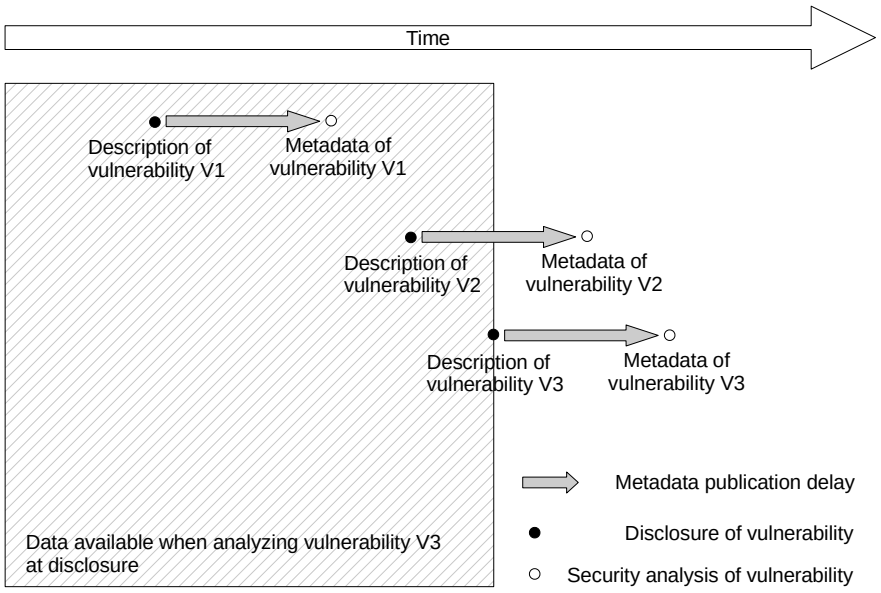

Figure 4.4 – In this example, when analyzing the text description of vulnerability V3 at disclosure time we have access to the text descriptions for V1 and V2 and to the metadata for V1, but not the metadata for V2 or V3.

the behavior of a production system put online on January 1st, 2017, initially fed with historical information from ten years before, which then monitored all newly disclosed vulnerabilities continuously for the next two years.

As seen in Section 3.2.2, the delay between a vulnerability disclosure and its analysis by NVD grew considerably in recent years. Therefore we decided to use a fixed *metadata publication delay* for all vulnerabilities which we set at 60 days. This means that when a vulnerability is disclosed on day N, we consider that its metadata is published on day $N + 60$. Conversely, when analyzing a vulnerability disclosed on day N, we have access to all vulnerability descriptions up to day N and to all vulnerability metadata up to day $N - 60$. The choice of 60 days ensures analysis conditions that are overall realistic (albeit simplified) but strictly worse than any recorded median case, and close to the worst recorded 9^{th} decile. Therefore if our analysis technique performs well during evaluation, we can be highly confident that it will perform as well or better in the real world. Figure 4.4 shows a simplified example of how time impacts the data available when analyzing vulnerabilities at disclosure.

Any non-zero metadata publication delay implies that the actual metadata of a vulnerability including its CPE URIs is not available during analysis. We can therefore use the CPE URIs from the vulnerability metadata as a ground truth for evaluation.

We propose two metrics to evaluate the quality of the ordered list of keywords: the ranking of the first relevant keyword, and the number of keywords necessary to reconstruct the software name. We evaluate our solution using the first metric in Section 4.3.2 and using the second one in Section 4.3.3.

4.3.2 Ranking of First Relevant Keyword

As our goal is to identify the software affected by a vulnerability, we formally define a *relevant* keyword as a substring of any software name or software vendor fields present in the CPE URIs included in this vulnerability metadata. As our analysis gives us a sorted list of keywords, we can expect relevant keywords to be placed at the beginning of the list before irrelevant ones. Therefore the ranking of the first relevant keyword is directly tied to the usability of the results. It should be emphasized that the number of CPE URIs included in a vulnerability metadata can vary greatly, as well as the reason why these CPE URIs were included in the first place. Usually at least one CPE URI is included as a machine-readable summary of the affected software described in the text description of the vulnerability. However security analysts sometimes include additional CPE URIs

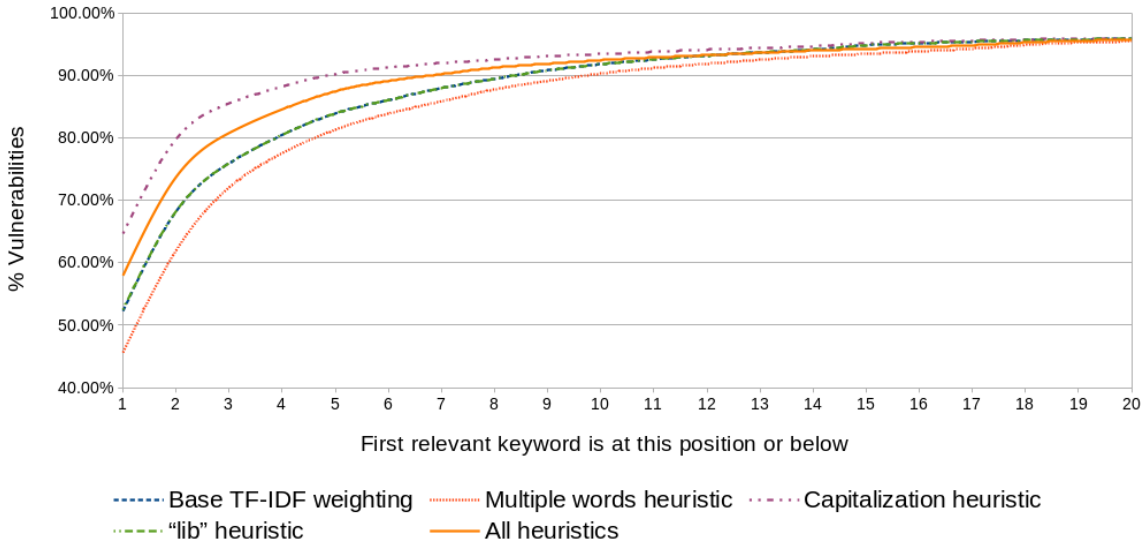

Figure 4.5 – First relevant keyword position.

for software absent from the text description (but still relevant for the vulnerability) to provide context after the fact. This metric sidesteps the problem of choosing the most appropriate CPE URI as an evaluation ground truth and instead focuses on extracting the relevant information actually present in the vulnerability text description.

The experimental results for this metric are described in Figure 4.5, for the base TF-IDF weighting, each heuristic described in Section 4.2.4, and all heuristics combined. We can see that in all cases at least 70 % of vulnerabilities have a relevant keyword in the top three keywords of their ordered list, at least 80 % of vulnerabilities have a relevant keyword in their top five keywords, and 90 % of vulnerabilities have a relevant keyword in the top ten. How to interpret these scores? In a control trial we randomly sampled 200 vulnerabilites and asked a security expert to guess the software product(s) affected by a vulnerability from the top three keywords without reading the vulnerability description. He gave 161 (80 %) correct answers, which is very close to our metric's (81 %) in the same configuration (all heuristics combined).

3 % of the vulnerabilities have no relevant keyword at all. For these vulnerabilities there is not a single common word between the description and the CPE URIs of the vulnerability, creating a plateau for the metric. We randomly sampled 20 of these vulnerabilities to find out the reason for the absence of keyword matching. In 18 cases the vulnerability disclosure is about a software that has never been seen before at the time. In the two other cases, the software was seen only one day and four days before, a time period within the

metadata delay set in Section 4.3.1. In all cases this leads to the CPE index not being populated with the proper software name, which is filtered out at the keyword extraction stage. Examples of such vulnerabilities are CVE-2016-1132 (first vulnerability disclosed for the *Shoplat* iOS application) and CVE-2016-1198 (*Photopt* Android application).

Regarding individual heuristics, we can see that our *capitalization* heuristic brings a substantial accuracy increase compared to the base TF-IDF weighting. The number of vulnerabilities with a relevant keyword at ranking 3 or below is increased from 76 % to 86 %.

The *multiple-words* heuristic decreases the accuracy under this metric. The reason is that multiple-words keywords are aggressively pushed to the beginning of the list most of the time in front of single word keywords. When the software and vendor names are only one word long, they might lose one rank because of an irrelevant multiple-words keyword. However the reconstruction metric sheds a different light on this heuristic's accuracy, as discussed in the next section.

The *lib* heuristic, while being strictly superior to the base TF-IDF weighting, provides such insignificant gains that it probably does not justify its maintenance cost.

All heuristics combined provide a measurable improvement over the base TF-IDF weighting without heuristics.

4.3.3 Number of Keywords Necessary for Software Name Reconstruction

Our second metric is about measuring our ability to fully reconstruct a software name using the smallest amount of keywords. Formally, this means finding a permutation of a subset of keywords equal to a full software name string from a CPE URI of the vulnerability, then measuring the highest keyword position in this group. As an intuitive example, if we want to reconstruct the software name "linux kernel" and our keyword list is, in order, "kernel", "overflow", "linux", and "buffer", we can reconstruct the software name using the first 3 keywords (disregarding "overflow"). This metric is strictly more difficult than the previous one, as we now want to reconstruct full strings instead of substrings, and we are focusing on the software name only and not the software vendor. However it is also more indicative of real-world usefulness, as reconstructing a complete software name provides more useful information than finding a substring of it.

The experimental results for this metric are described in Figure 4.6. As expected

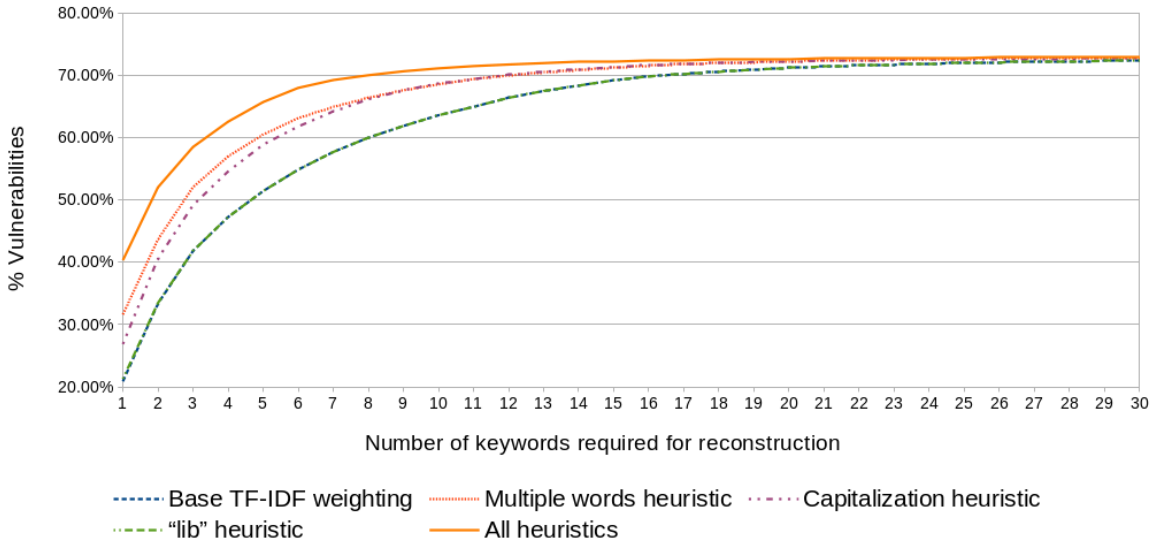

Figure 4.6 – Number of keywords necessary for name reconstruction.

reconstructing a full software name is more difficult than finding a relevant keyword. Using the base TF-IDF weighting, a software name can be reconstructed using the first three keywords only 42 % of the time.

27 % of the vulnerabilities do not have enough keywords in their description to reconstruct a software name at all. This leads to a lower plateau for the metric compared to *first relevant keyword position*. We sampled 20 of these vulnerabilities at random to investigate the cause of reconstruction failure. In 14 cases the affected software has never been seen before, leading to the same problem as described in Section 4.3.2. In the six other cases the software name is worded differently in the vulnerability description and the associated CPE URIs. As an example, CVE-2017-3814's description describes a vulnerability affecting the software *Cisco Firepower System Software* while the associated CPE URIs are referencing *Cisco Firepower Management Center*. One of these 6 cases, while technically a wording problem, can be attributed to excessive strictness in our parsing logic.

Regarding individual heuristics, the multiple-words heuristic now provides the biggest improvement. This makes sense, as having multiple-words keywords provides opportunities to drastically shorten reconstruction of multiple-words software name. For instance, in a single word setup, the software "linux kernel" takes at least two keywords to be reconstructed ("linux" and "kernel"), while it can be fully reconstructed in a single multiple-words keyword ("linux kernel"). The capitalization heuristic again brings a substantial improvement under this metric. This time again the *lib* heuristic brings a very small im-

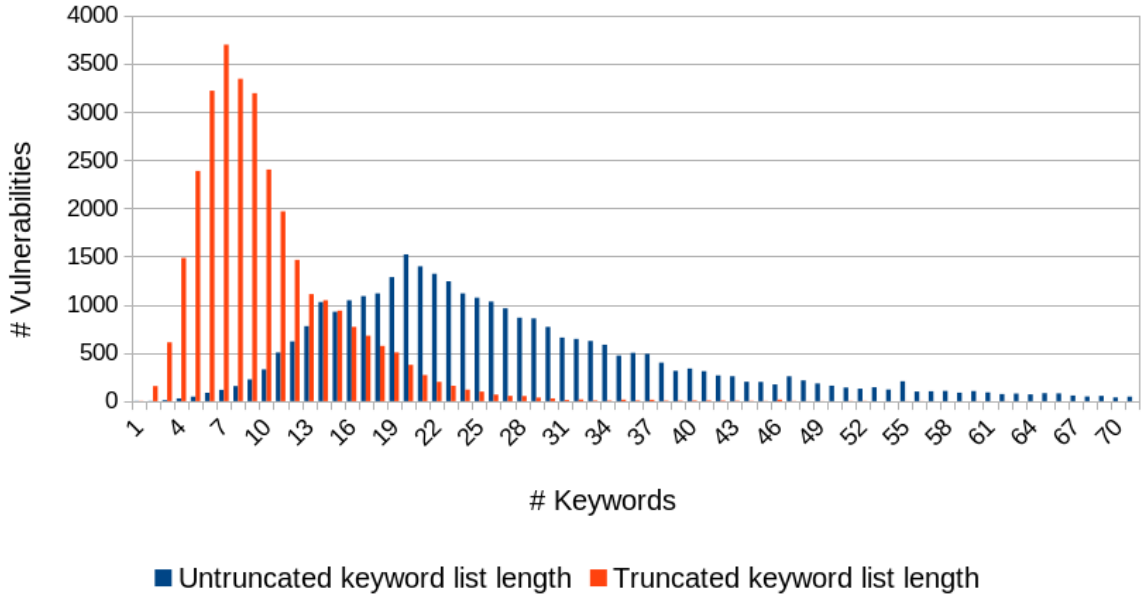

Figure 4.7 – Distribution of keyword list lengths before and after truncation with a target of preserving at least 95 % of the norm.

provement such that its maintenance cost is probably not justified. All heuristics combined together yield the best accuracy of all configurations. Using this setup we can reconstruct the full name of an affected software in 9 keywords or less for 71 % of the vulnerabilities in the evaluation dataset.

4.3.4 Keyword List Truncation Evaluation

In this section we evaluate two effects of the truncation step: the keyword list length reduction ratio and a possible accuracy loss due to excessive truncation. All our evaluations were done with a truncation target of preserving at least 95 % of the original norm. Figure 4.7 shows the distribution of keyword list lengths before and after truncation. The median untruncated keyword list length is between 23 and 24 words long, while the median truncated keyword list length is between 8 and 9 words long. The average reduction ratio is 3.22. We can conclude that keyword truncation has a substantial effect on keyword list length and is particularly effective at bringing keyword lists to sizes more easily readable by humans.

Does this reduction have an impact on the keyword list accuracy? Figures 4.8 and 4.9 show the impact of truncation on the two accuracy metrics studied before. We can see that while the effects of truncation are negligible on most vulnerabilities, the relevant

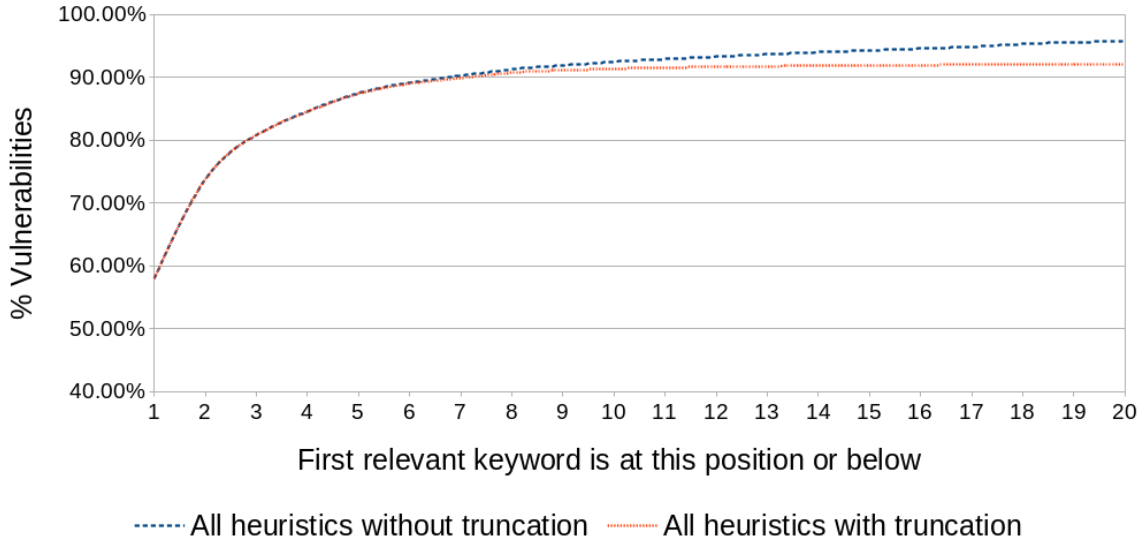

Figure 4.8 – Impact of keyword truncation on first relevant keyword position (truncated norm target = 95 %).

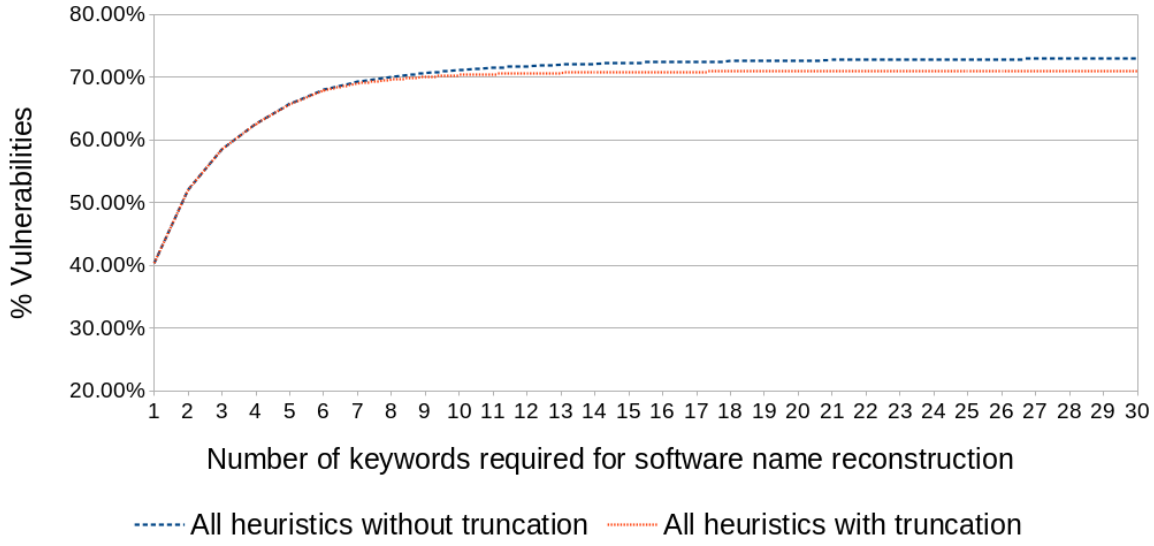

Figure 4.9 – Impact of keyword truncation on number of keywords necessary for name reconstruction (truncated norm target = 95 %).

keywords of a few hard-to-analyze vulnerabilities are lost during keyword truncation. 1446 vulnerabilities (4.64 %) went from having a low ranking first relevant keyword to having no relevant keyword at all. 649 vulnerabilities (2.08 %) had a software name that could be reconstructed before the truncation (albeit with difficulty) but not after.

The proper trade-off between truncation and accuracy probably depends on the nature of the keyword consumers downstream. Humans might prefer shortened keyword lists, as reading a 23 word long keyword list is probably less convenient than reading the actual vulnerability description. Meanwhile machine monitoring systems might or might not prefer untruncated lists, depending on their ability to properly handle keyword noise and detect weak signals in low-ranked keywords. In either case it is not straightforward to make good use of a relevant keyword at rank #20 or #25 when all preceding keywords are irrelevant, which makes a good case for truncation.

4.3.5 Performance of the Analysis Pipeline

While performance was not a major concern for us at this stage, analyzing a day worth of vulnerability historical data takes under a second on a commodity laptop with 16 GB of RAM and an Intel Core i7-7600U CPU @ 2.80GHz, making the pipeline suitable for near real-time analysis at disclosure. Indexing ten years of historical data, which would be a one-time operation on a production system, takes between 5 and 30 seconds on the same hardware, depending on the heuristics used: the multiple-words heuristic creates more keywords to index, increasing the time of indexing when this heuristic is activated. This fast turnaround time enables a security expert to easily formulate a new heuristic hypothesis, quickly reindex the full historical dataset using the new heuristic and get a prompt evaluation of how this heuristic increases or decreases the accuracy of the analysis.

4.4 Conclusion

In this chapter, we presented a pipeline for keyword extraction from vulnerability descriptions, a first step towards identifying the affected software at disclosure. To our knowledge this is the first work to focus on identifying the affected software immediately after a vulnerability disclosure. This pipeline is based on a bag-of-words featurization stage, a TF-IDF weighting improved by domain-specific heuristics, and a euclidean truncation scheme. Our results are promising, as a simple technique brings results that are

accurate enough to be useful in the real world. Since the initial publication of this research, Wåreus et al [221] built upon our work to show that software name reconstruction accuracy could be improved by using hard-to-explain machine learning techniques, a result that was expected.

A possible improvement for our technique would be to formulate an actual proposition for the name or for a complete CPE URI instead of returning a weighted list of keywords as we currently do. Still, a weighted list of keywords is very useful as it can be directly used as an input for other machine learning endeavors. In that scenario this contribution becomes a featurization stage for a larger machine learning pipeline, and we actually use it in this way in our contribution detailed in Chapter 6.

The affected software is an important property to have when analyzing the risk created by a vulnerability, but as seen in Section 3.2 there are other desirable properties that are not available at disclosure. In Chapter 5 we use similar techniques to reconstruct a complete severity analysis of a vulnerability seconds after its disclosure.

Chapter 5

Predicting CVSS Severity Using Linear Regression

In Chapter 3, we outline that automated vulnerability analysis at disclosure is an unsolved problem. In this chapter we present another contribution in that regard: predicting the CVSS base vector of a vulnerability at disclosure using its text description only, using explicable machine learning techniques such as bag-of-words and linear regression. We again focus on prediction explicability and a strong evaluation protocol to ensure the reliability of our prediction system in a production environment. This chapter content was published at the *15th International Conference on Availability, Reliability and Security* (ARES 2020) [65].

We present our objectives and how our work compares with the state of the art in Section 5.1. In Section 5.2 we present the CVSS prediction pipeline that we propose to fulfill these objectives. In Section 5.3 we present our evaluation protocol and results obtained. We discuss our results and their limitations in Section 5.4. We conclude in Section 5.5.

5.1 Objectives and Comparison with State of the Art

Our objective is to provide the first explicable CVSS vector prediction pipeline that can be used reliably at vulnerability disclosure. As described in Section 4.1 for our previous contribution, our strategy to achieve reliability comes from making the prediction explicable and having a strong evaluation protocol to evaluate its failure modes.

Before our contribution Khazaei et al had already explored CVSS severity *score* prediction [108]: they compared the use of SVM, Random Forests, and fuzzy systems to predict CVSS base scores, in both offline and online environments. However their work differs from ours in a number of ways:

— Their approach is focused on the severity score of the vulnerability, and they do not

predict it completely: they approach severity prediction as a classification problem, treating the integer part of the score as a class to be predicted. To the best of our knowledge, our approach is the first to reconstruct a full CVSS vector, thus providing more details about the vulnerability and allowing an actual CVSS score to be recomputed at disclosure.

— They do not take results explicability into account. Both their approach and ours start by treating a vulnerability description as a bag-of-words, as described in Section 2.4.4 (in their case, they also apply a TF-IDF [104] weighting scheme to the word count as described in Section 2.4.5). They use dimension reduction methods (Linear Discriminant Analysis and Principal Component Analysis) that create latent variables that cannot be easily interpreted. Moreover, some of the classifier algorithms they use, such as random forest or fuzzy systems, do not exhibit strong explicability properties either. For the reasons described in Chapter 3 and 4, we consider the explicability of automated analysis as a paramount quality of security systems. Therefore our prediction pipeline is designed to preserve explicability at all stages, as described in Section 5.2.

— Furthermore, their experimental protocol evaluates the prediction as a binary event: either the correct class has been predicted, or not. In our opinion there is a strong difference between incorrectly classifying a vulnerability with an actual severity score of 8.x as 7.y and 2.z. In our evaluation protocol we evaluate the prediction of each vector component individually and study the severity score prediction as a numerical error.

— Both their evaluation protocol and ours include an "online" evaluation, evaluating how a vulnerability can be analyzed using the data present at its disclosure. However they do not take into account that CVSS information is not immediately available after disclosure, whereas we take this delay into account. Moreover, their online evaluation uses monthly steps. This is not granular enough for evaluating a technique robustness for n-day vulnerabilities. Our evaluation protocol uses weekly steps for all variants of our technique and daily steps for the most promising variants.

— As our work is more recent, we evaluate our technique on both CVSS V2 and CVSS V3 using vulnerability data up to 2019 included. Their work only considers CVSS V2 and data up until 2013 included.

To summarize, our objectives are:

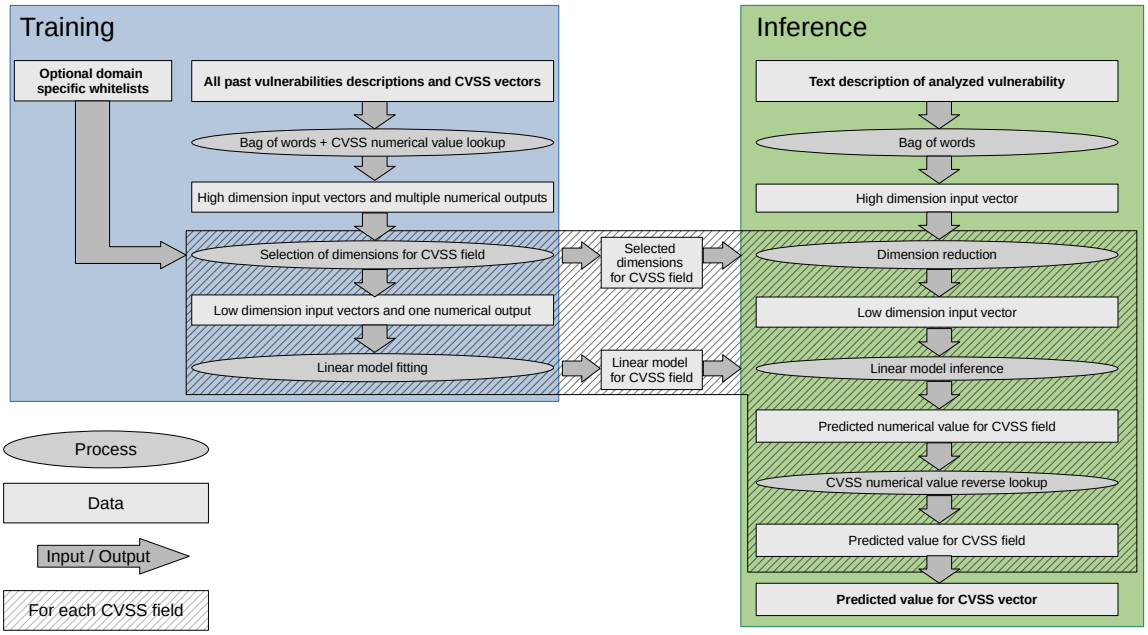

Figure 5.1 – Our CVSS vector prediction pipeline.

1. Predicting an entire CVSS V2 and V3 vector for a vulnerability using its text description only.
2. Ensuring that this prediction is explicable.
3. Evaluating the prediction system thoroughly to understand its possible failure modes.
4. Taking into account the delay between a vulnerability disclosure and the publication of the associated metadata.

In the next section we describe how we fulfill these objectives.

5.2 Proposed Approach

In this section we describe our CVSS vector prediction pipeline. Its input is the free-form description of a new vulnerability. The input is analyzed using all vulnerability descriptions and metadata available at the time of disclosure. The output is a predicted CVSS base vector (in V2 or V3 format). A high-level overview of the proposed vulner-

ability analysis pipeline is shown in Figure 5.1. We make use again of the NVD CVE database, as it includes descriptions (which we use as training *input*) and vulnerability CVSS vectors (which we use as training *output*) for all past CVE vulnerabilities. Some variants of the pipeline use a *whitelist-based dimension reduction* scheme (described in Section 5.2.1) and require additional data sources: CPE URIs and the CWE framework.

Just as in Chapter 4, beyond accuracy we have one major goal when designing our machine learning pipeline: keeping an explicable and reliable relationship between the input and the decision. This also implies limiting the number of hyperparameters in the pipeline. As we explained in Section 2.4.4, the space of all possible combinations of hyperparameters settings for a given training pipeline is called a *hyperstate space* and grows exponentially with the number of hyperparameters. As hyperparameters must be set before the beginning of training (which can be lengthy), a highly dimensional hyperstate space cannot be fully explored repeatedly, creating risks of accuracy drifts over time or subtle differences in failure modes when inferring. We would like all variations of our training pipeline to have either zero or one hyperparameter, allowing for a full understanding of the hyperstate space.

To handle these constraints, we propose the architecture described in Figure 5.1. For training we first adopt a bag-of-words approach on each vulnerability description. As the number of words in the bag-of-words index grows loosely linearly with the number of vulnerabilities, the size of the resulting dimensional space is unbounded. Therefore we apply a filtering scheme (described in Section 5.2.1) removing irrelevant words in order to keep the number of dimensions of the vulnerability vectors manageable. We then train one regression model (described in Section 5.2.2) for every component of the CVSS vector using past vulnerabilities as input.

Every step of the training phase is mirrored in the inference phase. For a vulnerability to be analyzed, its description is converted into a real vector through bag-of-words. For each CVSS field, dimension reduction is applied by retaining the words deemed relevant during training. The predicted value for the CVSS field is then inferred using the previously trained regression model. Once a value has been predicted for each CVSS field, a predicted CVSS vector is assembled by concatenating each field.

5.2.1 Dimension Reduction Through Filtering

As we wish to preserve the auditability of the bag-of-words embedding (each dimension counts the number of occurrences of a word) we choose not to use any dimension reduc-

CWE Weaknesses
Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
Stack-based Buffer Overflow
Use After Free
UNIX Symbolic Link (Symlink) Following
Extracted keywords
a after based basic buffer following free html improper in link neutralization of overflow page related script stack symbolic symlink tags unix use web xss

Table 5.1 – Examples of CWE weaknesses and keywords extracted from them.

tion technique creating latent variables. As seen in Section 2.4.4 this excludes techniques such as Linear Discriminant Analysis, Principal Component Analysis, Locality Sensitive Hashing [98], and many others. Instead, we retain existing dimensions but filter out the ones we deem irrelevant. We propose two approaches for that: domain-specific whitelists, and dimension sorting using conditional entropy.

Domain-Specific Whitelists

We can generate a keyword whitelist that let us control the number of dimensions of the vulnerability vectors. We create three whitelist variants based on two pieces of data as sources of keyword terms: CPE URIs [139], which we use in the same way as in Chapter 4, and Mitre's Common Weakness Enumeration (CWE) database [52]. As we described in Section 2.2.2, CWE is a list of common software and hardware security weaknesses, aiming to serve as a common vocabulary to describe similar vulnerabilities. We extract all terms used in CWE titles in order to use them as a whitelist. Table 5.1 shows several examples of CWE weaknesses and the keywords extracted from them. The third variant is both whitelists merged together.

We evaluate the impact of all three whitelists in Section 5.3.

Conditional Entropy Sorting

Our second approach to dimension reduction is not based on domain-specific knowledge but on information theory. We use *conditional entropy* to sort words by how much prediction power they provide. In this approach we consider a CVSS field as a random

Word count	***Attack Vector* value**			
"local"				
	Physical	Local	Adjacent	Network
0	5	62	11	556
1	0	53	1	6
"document"				
	Physical	Local	Adjacent	Network
0	5	98	12	545
1	0	17	0	17
"compiler"				
	Physical	Local	Adjacent	Network
0	5	115	12	505
1	0	0	0	56
TOTAL				
	5	115	12	562

Table 5.2 – Conditional distributions for various keywords and the value of CVSS field *Attack Vector* for vulnerabilities disclosed between January 1st, 2016 and April 1st, 2016.

variable to be predicted, and the count of a word in the vulnerability description as a random variable whose value is already known.

Table 5.2 depicts the distribution of results for the CVSS field *Attack Vector* for 694 vulnerabilities disclosed between January 1st, 2016 and April 1st, 2016, and the related conditional distributions for occurrences of words "local", "document", and "compiler" in the associated vulnerabilities descriptions. We can see that the word "compiler" has a low predictive power on *Attack Vector*, as the conditional distribution remains close to the original one whether the word is present or absent. Conversely, the word "local" has more predictive power as its presence completely changes the distribution, with the most probable value switching from *Network* to *Local*.

Conditional entropy, described in Equation 5.1, provides a synthesis of this difference by computing the entropy of one random variable when another one is known. All entropy computations in this work are done in base 2 with results expressed in bits.

$$H(Y|X) \ = - \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)} \tag{5.1}$$

Once we have computed the conditional entropy of a CVSS field for every word in our index, we can sort the associated dimensions: the words with the lowest conditional

Attack Vector (AV)	
Metric Value	**Numerical Value**
Network (AV:N)	0.85
Adjacent (AV:A)	0.62
Local (AV:L)	0.55
Physical (AV:P)	0.2

Table 5.3 – Enumerated members for CVSS V3 field *Attack Vector* and their associated numerical values as described by the CVSS 3.0 and 3.1 specifications.

entropy are the more predictive dimensions. We can choose a number of dimensions we wish to retain and discard the rest.

There are two theoretical drawbacks to this approach. First the mutual information between dimensions is not taken into account. The first and second most predictive dimensions might be very predictive on their own but the second might not provide a lot more information on top of the first one. The algorithm is linear in the number of dimensions and properly handling mutual information would make it quadratic as the conditional entropy for each non-selected dimension would need to be recomputed after each dimension selection. However not handling mutual information is not a problem in practice as the dimensions are used as an input to a regression model. Dimension sorting is only used to retain a tractable number of dimensions, and as long as enough predictive dimensions are retained, the regression model is still able to provide reasonable results.

Second, the number of dimensions we choose to retain is a hyperparameter. In practice, as this hyperparameter is the only one in the whole analysis pipeline, our hyperstate space is small enough to be explored thoroughly, as we see in Section 5.3.

5.2.2 Regression Modeling on CVSS Vectors

Once we have a tractable number of dimensions to work with, we can train a model with it. We consider each CVSS vector component, or *field*, as an independent problem and we train a different model for each of them. As described in Section 2.2.2, each CVSS vector component is valued using a multiple choice enumeration, as shown in Table 5.3. An intuitive approach would be to use multinomial logistic regression or multinomial ordinal regression to predict these enumerated fields. However, as seen in Section 2.4, multinomial regression models are not straightforward to train, with iterative analytical methods such as Iteratively Reweighted Least Squares (IRLS) [91] or iterative gradient-based solvers

such as L-BFGS-B [236]. All iterative methods inherently require one or two new hyperparameters (either the acceptable error threshold before stopping or the maximum number of steps, or both), and a lot of solvers require additional hyperparameters specific to their approach.

The CVSS specification gives us an interesting way of simplifying our modeling. All CVSS V2 fields values and all but one CVSS V3 fields have a numerical value associated with each enumerated value. This numerical value is used in the calculation of the CVSS severity score. Table 5.3 shows the numerical value for each enumerated values of the CVSS field *Attack Vector*.

Instead of using multinomial regression to predict the value of a CVSS field enumeration, we can use linear regression to predict the numerical value for the CVSS field. From the predicted numerical value, we then select the enumerated value whose associated numerical value is the closest to the prediction. A linear model can be trained analytically using Ordinary Least Squares (OLS) without requiring any new hyperparameter.

One CVSS V3 field, *Scope*, is a binary field (with possible values *Changed* or *Unchanged*) with no associated numerical value. For this field we associate *Unchanged* and *Changed* to -1 and 1 respectively then treat it in the same way as every other field. This is equivalent to a binary logistic regression problem followed by selecting the most probable outcome of the two, without considering the odds.

5.3 Evaluation

In this section we present our evaluation protocol and results obtained. In Section 5.3.1 we present the experimental setup. In Section 5.3.2 we evaluate the accuracy of our pipeline when predicting the individual CVSS fields of a vulnerability. In Section 5.3.3 we evaluate the accuracy of our pipeline when predicting the CVSS severity score of a vulnerability. In Section 5.3.4 we discuss the explicability of our experimental results. In Section 5.3.5 we evaluate the performance of our prediction pipeline. Finally, in Section 5.3.6 we discuss how retraining the model daily or weekly impacts accuracy. All the code and data used for our experiments are available at [68]. For this experimental protocol we also developed the open-source library *libcvss* [114], which aims to become the reference CVSS parsing and manipulation library in the Rust language.

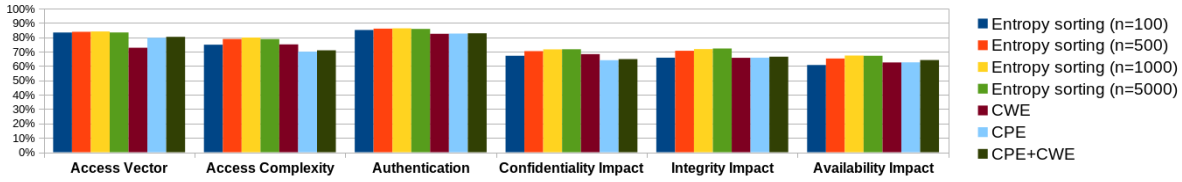

Figure 5.2 – Success rate for individual CVSS V2 field predictions.

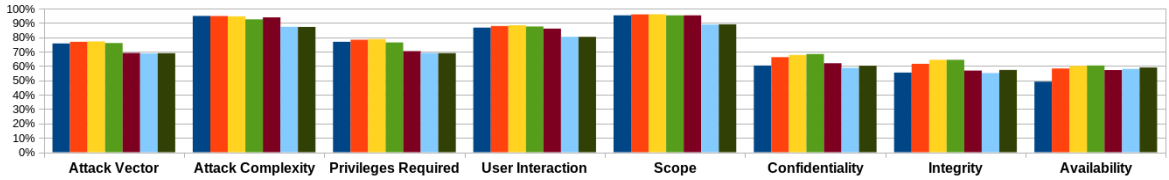

Figure 5.3 – Success rate for individual CVSS V3 field predictions.

5.3.1 Experimental Setup

We analyzed all 33807 CVE vulnerabilities disclosed between January 1st, 2018 and January 1st, 2020, using all 70172 CVE vulnerabilities disclosed between January 1st, 2007 and December 31st, 2017 as past historical data. This experimental setup simulates the behavior of a production system put online on January 1st, 2018, initially fed with historical information from eleven years before, which then monitors all newly disclosed vulnerabilities continuously for the next two years. Each vulnerability is analyzed using the data available at its disclosure day only, as in Chapter 4. We decided again to use a fixed metadata publication delay of 60 days for all vulnerabilities, for the reasons highlighted in Section 4.3.1.

Seven configurations of our analysis pipeline were evaluated. Three are whitelist-based: CPE, CWE, and both combined together. Four are based on conditional entropy: the number of retained dimensions was set to 100, 500, 1000, and 5000. All these configurations are evaluated for CVSS V2 and V3 prediction. In order to optimize our computing resources usage, our experiments are simulated assuming a weekly retraining of the regression model (simulating a daily retraining would require 7 times more resources). Nevertheless in Section 5.3.6 we evaluate the impact of going from weekly to daily retraining for the most promising configuration.

5.3.2 Prediction for Individual Fields

Figures 5.2 and 5.3 show the raw success rate for each field and predictor: that is, the number of correct predictions over the number of total predictions. From this data we can make a few conclusions:

— Dimension reduction using a whitelist based on CPE URIs exhibits overall worse accuracy than all other approaches, even when combining it with the CWE whitelist. Our hypothesis is that this is due to the CPE whitelist retaining much more dimensions than all other approaches (see Section 5.3.5 and Figure 5.8), to the point of being detrimental to accuracy, a common problem with linear regression in highly dimensional spaces.

— Dimension reduction using conditional entropy with a number of retained dimensions between 500 and 1000 exhibits better accuracy than all other approaches.

— Some fields are more difficult to predict than others. In particular, the Confidentiality, Integrity and Availability fields have lower accuracy than other fields, especially in CVSS V3.

Success rates show how the predictors are working "out of the box", but not the best possible predictor that can be constructed through the prediction technique, as it does not take into account "permuted" predictions. For example, it is possible to use a predictor guessing wrong 100 % of the time to construct another predictor that guesses right 100 % of the time.

Thus to further evaluate our predictors we can use conditional entropy again (this time as an evaluation tool) in order to measure the predictive power of each configuration on individual CVSS fields. By computing the original entropy of a given CVSS field, and then computing the conditional entropy of this field when knowing a predicted value of it, we can measure how much information is gained through the prediction. Figures 5.4 and 5.5 show the conditional entropy for each CVSS field and each prediction pipeline configuration, for CVSS V2 and CVSS V3 respectively. Each column group describes one CVSS field, with each column measuring the entropy of this field conditioned to the predicted value for this field according to the given configuration (the lower entropy the better). The last column of each group shows the entropy of the actual CVSS field, to serve as a baseline for comparison. We can see that on some fields the conditional entropy is even higher than the initial entropy: this can be interpreted as performing worse than a random guess (weighted using the past frequencies of the different members of the CVSS field). However our best configurations all have significantly lower conditional entropy

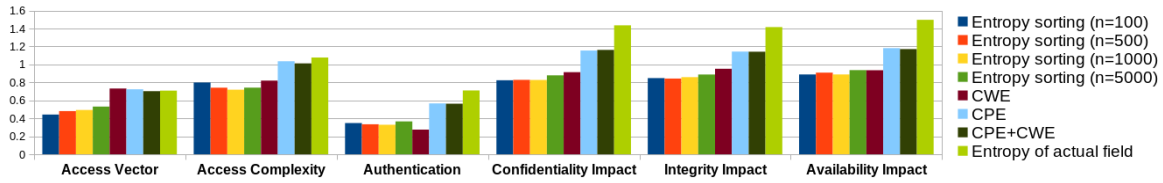

Figure 5.4 – Conditional entropy for individual CVSS V2 field predictions (lower is better).

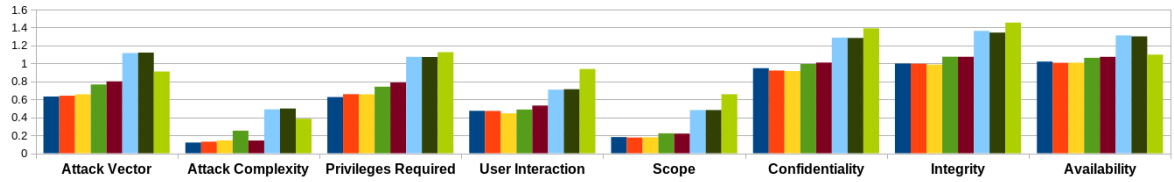

Figure 5.5 – Conditional entropy for individual CVSS V3 field predictions (lower is better).

than the real entropy of the CVSS field, which indicates that meaningful information was gained through the predictor.

5.3.3 Prediction for the CVSS Severity Score

After making a prediction for each CVSS field for a given vulnerability, we can assemble a complete, predicted CVSS vector and compute its severity score using the standard CVSS computation rules (V2 or V3). We can then compare this predicted severity to the actual severity of the vulnerability, computed from the actual CVSS vector for the vulnerability. We define the *severity prediction error* as $PredictedSeverity - ActualSeverity$, giving us a value between -10.0 and $+10.0$. A severity prediction error of 0 is a perfect prediction. Any value above 0 is a *false positive*, with the vulnerability being *less* severe than predicted. Any value below 0 is a *false negative*, with the vulnerability being *more*

Configuration	**50 %**		**80 %**		**99 %**	
Entropy sorting (n=100)	0	2.4	-1.5	3	-5	5.4
Entropy sorting (n=500)	0	1.8	-1.5	2.6	-5.4	5.4
Entropy sorting (n=1000)	0	1.5	-1.5	2.5	-5.8	5.4
Entropy sorting (n=5000)	0	1.5	-1.7	2.5	-6.4	5.4
CWE	-0.3	1.7	-1.9	2.8	-5.4	5.4
CPE	-0.6	1.7	-2.5	2.6	-7.8	5.4
CPE + CWE	-0.7	1.5	-2.6	2.5	-8.5	5.4

Table 5.4 – Error intervals for CVSS V2 score prediction for 50 %, 80 % and 99 % of the vulnerabilities in the evaluation dataset.

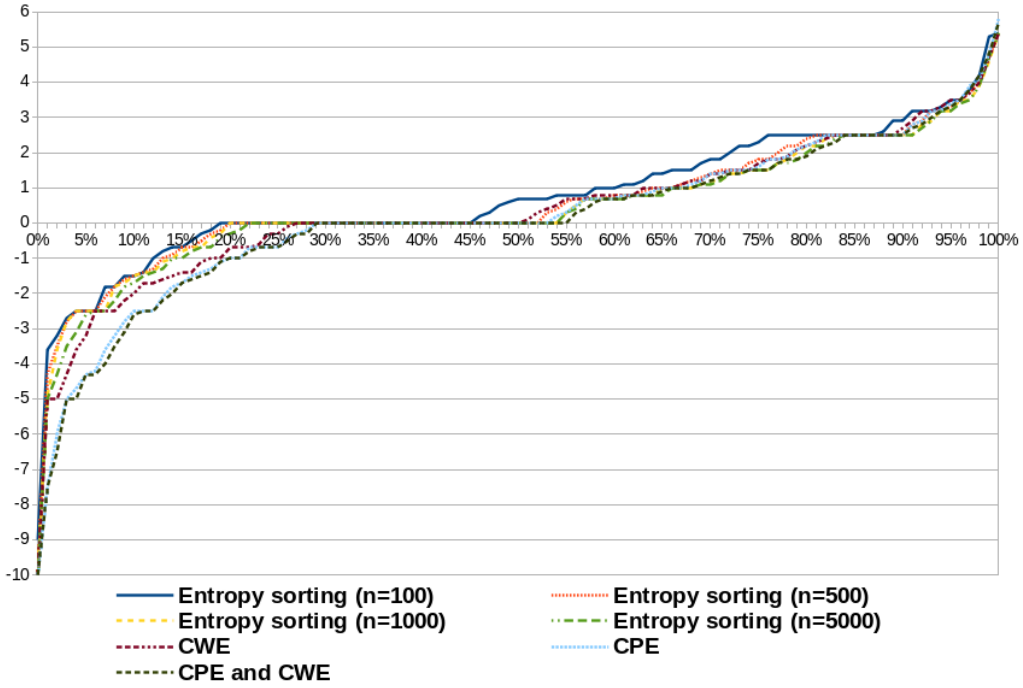

Figure 5.6 – Severity prediction error distribution for CVSS V2.

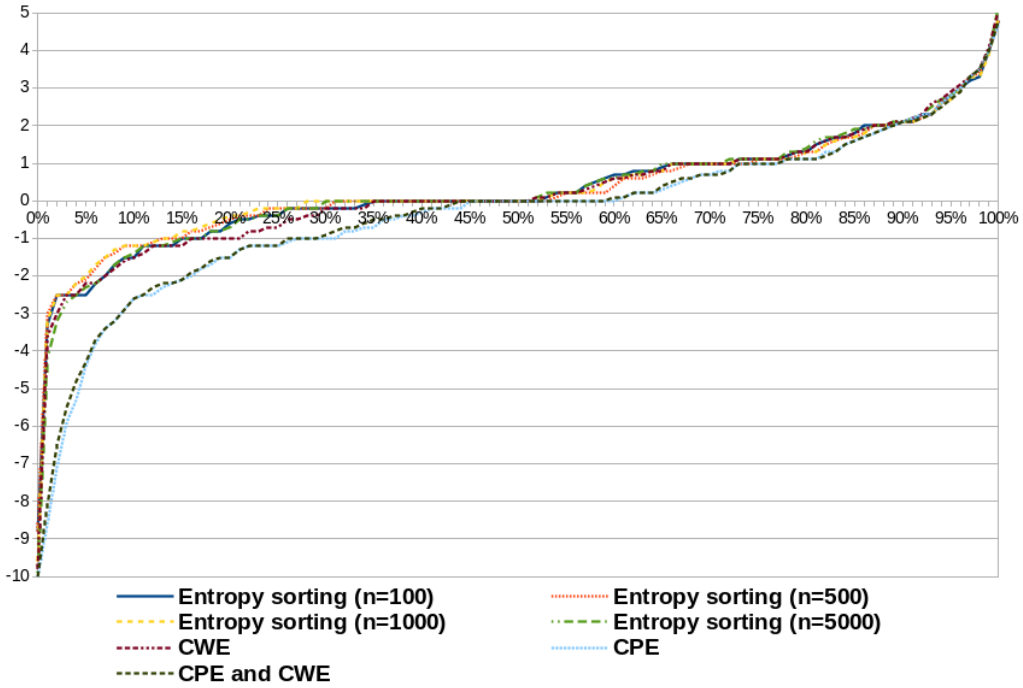

Figure 5.7 – Severity prediction error distribution for CVSS V3.

Configuration	50 %		80 %		99 %	
Entropy sorting (n=100)	-0.4	1.1	-1.4	2.1	-3.5	4.5
Entropy sorting (n=500)	-0.2	1.1	-1.2	2.1	-3.5	4.5
Entropy sorting (n=1000)	-0.2	1.1	-1.2	2.1	-3.7	4.5
Entropy sorting (n=5000)	-0.4	1.1	-1.4	2.1	-5.4	4.6
CWE	-0.7	1.1	-1.5	2.1	-5.4	4.7
CPE	-1.2	1	-2.6	2.1	-9.8	4.5
CPE + CWE	-1.1	1	-2.6	2.1	-9.8	4.5

Table 5.5 – Error intervals for CVSS V3 score prediction for 50 %, 80 % and 99 % of the vulnerabilities in the evaluation dataset.

severe than predicted.

Figures 5.6 and 5.7 show the distribution of the severity prediction error in our evaluation dataset, while Tables 5.4 and 5.5 show the error intervals for 50 %, 80 %, and 99 % of our evaluation dataset respectively. From these results we can make the following conclusions:

— Dimension reduction based on conditional entropy sorting provides systematically better accuracy than dimension reduction based on whitelists. In particular all whitelists based on CPE URIs provide strictly less accuracy than all other approaches. Our whitelist based on the CWE framework provides better results, closer in accuracy to entropy sorting but still outperformed by the best entropy sorting configurations. Interestingly, combining CWE and CPE whitelists into one decreases accuracy compared to either whitelists, making it the worse-performing configuration. This reinforces our hypothesis that the large number of dimensions of the CPE whitelist is detrimental to accuracy.

— Regarding conditional entropy sorting, the number of dimensions kept after sorting (our sole hyperparameter) does have an impact on accuracy. In both CVSS V2 and V3, keeping the number of dimensions between 500 and 1000 is important in order to get the best prediction results. When predicting the CVSS V3 severity score, accuracy differences between 100, 500 and 1000 retained dimensions are nearly indistinguishable. When predicting CVSS V2 severity scores, retaining 100 dimensions provides more false positives than 500 or 1000. However it is possible this is an artifact of our evaluation dataset (this could be checked in the future by reproducing our experiments using vulnerabilities disclosed from 2020 and later years), especially given that there are no such differences when predicting CVSS V3.

Description
This vulnerability allows remote attackers to execute arbitrary code on vulnerable installations of Foxit Reader 9.1.0.5096. User interaction is required to exploit this vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the handling of the setInterval() method. The issue results from the lack of validating the existence of an object prior to performing operations on the object. An attacker can leverage this vulnerability to execute code in the context of the current process. Was ZDI-CAN-6438.

Attack Vector	**Actual:** Network, **Predicted:** Network			
Keyword	remote	required	file	interaction
Weight	0.059	0.037	-0.029	0.025

User Interaction	**Actual:** Required, **Predicted:** Required			
Keyword	file	page	malicious	interaction
Weight	-0.287	-0.021	-0.020	-0.019

Table 5.6 – Top four keywords used to predict CVSS V3 fields *Attack Vector* and *User Interaction* for vulnerability CVE-2018-17625 disclosed on 01/23/2019, using a CWE whitelist for dimension reduction.

— In nearly all cases the prediction error for CVSS V3 is lower than for CVSS V2. This was surprising to us as CVSS V3 includes eight fields while CVSS V2 only includes six: this alone should make it more error prone when predicting a full CVSS V3 vector. Our hypothesis is that CVSS V3 computation rules are more likely to give closer severity scores to similar but not identical vectors.

— CVSS V3 score prediction using our best configuration (conditional entropy with 500 retained dimensions) is particularly resilient against false negatives, with 50 % of the evaluated vulnerabilities having a false negative prediction error below 0.2, 80 % below 1.2, and 99 % below 3.5 %. This makes the predictor suitable for real-world applications, such as using the predicted CVSS vector in a vulnerability management pipeline as a placeholder while waiting for the publication of a definitive CVSS vector by NVD.

5.3.4 Results Explicability

To maintain explicability, our prediction pipeline is able to show which weighted keywords were used to make a prediction. Table 5.6 shows an example of how explicability

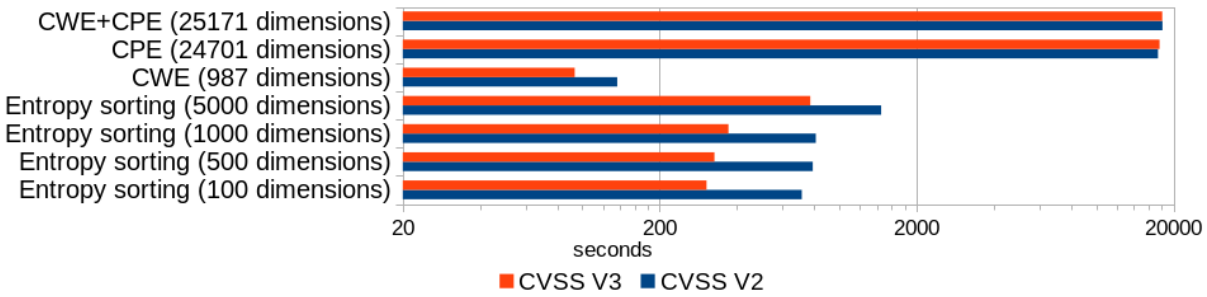

Figure 5.8 – Duration in seconds of the complete analysis pipeline at date 2019-01-01. Time scale is logarithmic.

can be maintained throughout the prediction pipeline. CVE-2018-17625 is a vulnerability affecting Foxit Reader disclosed on 01/23/2019. For two CVSS V3 fields (*Attack Vector* and *User Interaction*) we show the top four keywords used to predict their value. We can see that the presence of the word "remote" was the biggest factor when making the decision to predict the value *Network* for *Attack Vector* (meaning vulnerability exploitation can be accomplished remotely). Conversely, the word "file" was an important factor to predict the value *Required* for *User Interaction* (meaning vulnerability exploitation requires a user to do a specific action).

5.3.5 Performance Impact of Dimension Reduction

The performance of the pipeline is not a goal in itself for us, however we want to ensure that our architecture is suited for daily retraining. This means a complete retraining of all our models from scratch, followed by an inference for all new vulnerabilities disclosed on a new day should not take more than a fraction of a day. Figure 5.8 shows the duration of the pipeline (in seconds, using a logarithmic scale) for each configuration, at date 2019-01-01, assuming a complete retraining followed by an analysis of all vulnerabilities disclosed on that day. All computations were done on a Dell PowerEdge C6420 server with Intel Xeon Gold 6130 CPUs (Skylake, 2.10 GHz, 2 CPUs/node, 16 cores/CPU) and 192 GiB of RAM. All performance experiments were run three times: duration was very stable with half of variations below 1 % and all of them below 3 %.

Durations range from 1 min 33 s for the CWE whitelist in CVSS V3 (and 2 min 16 s in CVSS V2) to close to 5 hours for the CPE whitelist. Meanwhile, entropy sorting configurations run during 5 to 25 minutes depending on the number of retained dimensions. All configurations are therefore suited for daily retraining if deemed necessary. A production

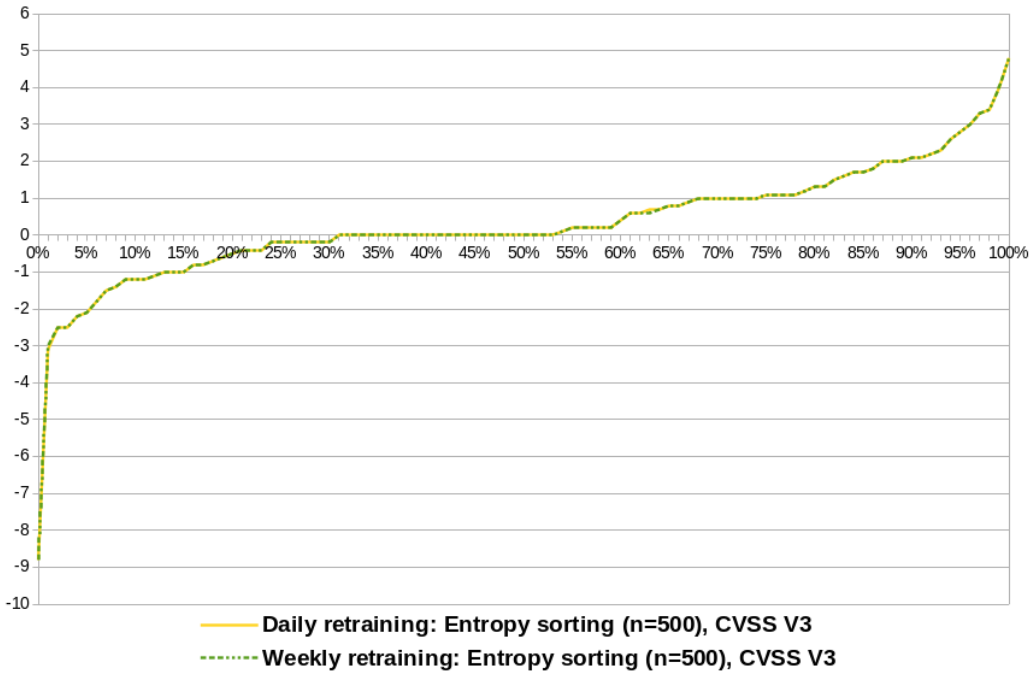

Figure 5.9 – Impact of daily and weekly retraining on the error distribution for CVSS V3 score prediction.

implementation of the pipeline could likely be faster than our current prototype as we did not spend significant effort optimizing this aspect of the experimental protocol beyond confirming the feasibility of daily retraining.

5.3.6 Daily Retraining vs Weekly Retraining

As described in Section 5.3.1, all experiments were run assuming a weekly retraining of the prediction model. This was an experimental constraint as we did not have the computing resources to simulate all experiments assuming a daily retraining (which consumes seven times more resources). However performance measurements from Section 5.3.5 suggest that daily retraining of a production system is feasible. Therefore we ran the most promising configuration twice (dimension reduction through entropy sorting with 500 retained dimensions) assuming both weekly and daily retraining of the model for CVSS V3 prediction. This impacted the score of 1371 vulnerabilities, around 4 % of our evaluation dataset. As shown in Figure 5.9 this has negligible impact on the severity prediction error distribution using our experimental setup. A similar experiment done with CVSS V2 on a subset of the same dataset gave similar results. One should note that our choice to

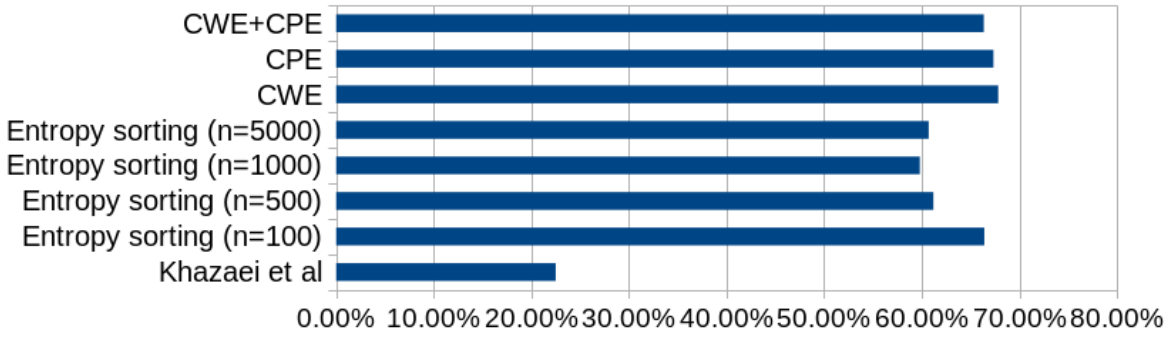

Figure 5.10 – Error rate (as defined by Khazaei et al) for CVSS V2 score prediction.

simulate a metadata publication delay of 60 days for all vulnerabilities could hide some of the impact of the training frequency compared to a production system relearning a new model as soon as new information becomes available. However in all cases the accuracy of such production systems should at least be identical or even better than our results, for the reasons described in Section 4.3.1.

5.4 Discussion and Limitations

Our prediction pipeline is designed with predictability and explicability in mind and this has an impact on accuracy. Figure 5.10 compares our approach with the more elaborated machine learning techniques used by Khazaei et al. The comparison has some limits as they used the now unavailable OSVDB dataset on years 2012 and 2013 using training data starting from 2004 while we used the NVD dataset on years 2018 and 2019 using training data starting from 2007. Moreover the error rate metric they proposed does not allow differentiation between small and large errors, as they consider score prediction as a classification problem: they consider the integer part of the score as a class to be predicted, and only distinguish between correct and incorrect predictions. Nevertheless, their techniques are indisputably more effective at predicting CVSS scores than ours. This raises a number of questions for an organization willing to deploy a production CVSS prediction pipeline: do CVSS vectors have inherent value or are they just a mean to compute a severity score? How to balance worst-case accuracy and average accuracy? Is decision explicability important? Different organizations may not have the same answers to these questions, leading them to different architectures. In the future we hope to improve the accuracy of our prediction pipeline to get closer to Khazaei et al, while still preserving

the predictability and explicability properties making our current architecture novel. This could involve refining the dimension reduction scheme, for instance by using alternative whitelists, or using additional input data for hard-to-predict fields such as Confidentiality, Integrity and Availability impact.

5.5 Conclusion

In this chapter we introduced a method to automatically predict CVSS vectors and scores for newly disclosed vulnerabilities, relying only on their human-readable description. Our architecture is based on explicable machine learning and information theory techniques such as bag-of-words, linear regression, and conditional entropy. Our results are promising, as a simple technique brings results that are accurate enough to be useful while providing decision explicability, a missing property in the state of the art.

Combined with the previous contribution described in Chapter 4, we now have a prediction model for most of the inherent properties of a vulnerability at its disclosure, without the need for a human expert analysis. However, this analysis still considers the vulnerability without considering the peculiarities of a given information system. In Chapter 6 we propose a risk analysis scheme taking into account a specific information system to be protected.

Chapter 6

Estimating Risk-Level Evolution Due to Vulnerability Disclosure Using Active Learning

In Chapters 4 and 5, we tried to reconstruct information inherent to a vulnerability, without taking into account the information system in which the vulnerability may be present. In this chapter we attempt to automatically evaluate the evolution of threat and risk for a given information system following the disclosure of a vulnerability.

As explained in Section 3.2.2, exhaustively listing the software and hardware components of an information system is non-trivial. This makes analyzing the risk created by a vulnerability disclosure to an information system even harder. Instead of basing the risk analysis on a list of components, we focus on the security team members tasked with protecting the information system, by studying how a Chief Information Security Officer (CISO) and her subordinates actually react to vulnerability disclosures. The core proposition of this chapter is to use active learning to extract the conscious and unconscious knowledge of an information system's security team in order to automate the risk analysis of a vulnerability for a specific system to be defended.

At this early stage of our research on this topic we choose not to execute any automated defense reaction following this risk analysis. Instead, the automated system evaluates, for all new vulnerability disclosures, the relevance of alerting an on-call human security operator who will then take action herself. In order to achieve this goal we present in this chapter a preliminary contribution that we plan to improve in the future.

We present our objectives for this contribution in Section 6.1. In Section 6.2 we present the active learning architecture that we propose to fulfill these objectives. In Section 6.3 we present our evaluation protocol and preliminary results. We discuss our results in Section 6.4. We conclude in Section 6.5.

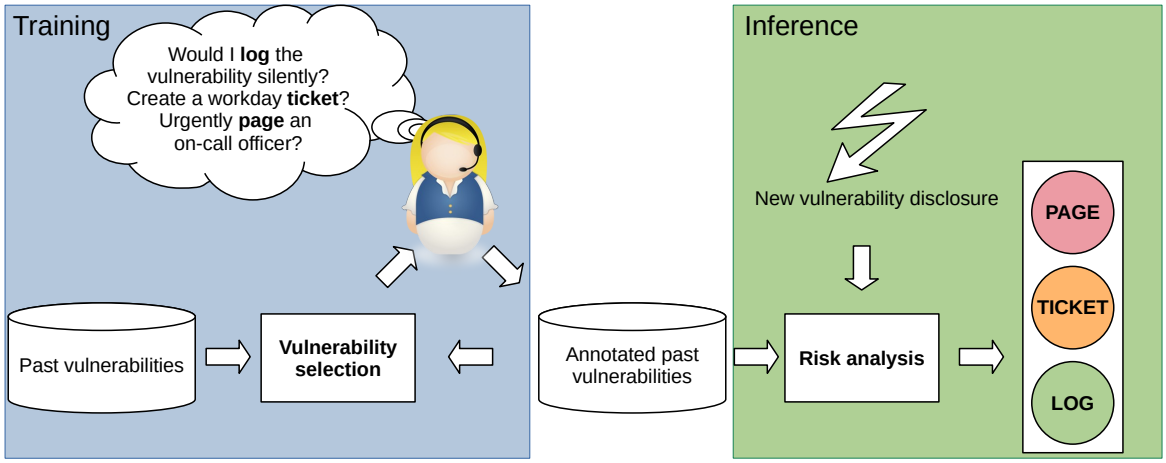

Figure 6.1 – Proposed active learning architecture to estimate the risk-level evolution.

6.1 Objectives

We want to propose an automated system evaluating, for a newly disclosed vulnerability, the necessity to alert an on-call security operator regarding a new risk in the context of a specific information system to be defended. To this end we intend to let an expert (a CISO or her subordinates) participate in the training of the prediction system by annotating past vulnerabilities to indicate whether she would have wished being alerted about a similar recent disclosure. We should also fulfill the following constraints:

1. We want to make the best use of past historical vulnerability data in order to predict how to react to future vulnerabilities.
2. We want to make the best use of the limited time of security experts by allocating a fixed *annotation budget* for the training and *selecting the most relevant vulnerabilities to be annotated*.
3. We want both the vulnerability selection process during the training stage, and the vulnerability analysis process during the inference stage, to be explicable.

6.2 Proposed Approach

We propose an active learning architecture based on the work by Settles et al [182], which we select because of its simplicity and explicability properties. This architecture

is depicted in Figure 6.1. It is composed of a *training phase* during which an expert and the system train together and an *inference phase* during which the system monitors new vulnerabilities and evaluates the risk that they create. Since the active learning architecture originally proposed by Settles et al is deterministic and only handles static datasets, we made slight modifications to the scheme (detailed in Sections 6.2.6 and 6.2.7) to handle the dynamic nature of an ongoing flow of vulnerability disclosures and support a controllable amount of randomness during the training process.

During the training phase the system iteratively selects a relevant past vulnerability, submits it to the expert through the user interface, who then annotates it by choosing an appropriate alert level for the vulnerability. The training phase is actually split in two parts. First, the *offline training phase* happens during the deployment of a production prediction system at a fixed point in time, and lets the prediction system and the expert create together an initial knowledge base by letting the prediction system select any past vulnerability at that point in time, with the system having full control on the vulnerability selection process. Second, the *online training phase* is a continuous interaction between the prediction system and the expert over time, discretized into *periods* such as weeks. Every time a new period starts, the prediction system iteratively selects new vulnerabilities among the ones disclosed during the last period and makes the expert annotate them. Each of these phases has a separate annotation budget, with the offline phase requiring a one-off effort from the expert and the online phase requiring a continuous effort over time. These two phases of the training are important, as they allow the prediction system to acquire both long-term context on past vulnerability data as well as awareness of the latest vulnerability disclosures.

Once the expert considers that the system has been made reliable enough through offline and online training (we come back to this in Section 6.3), the inference phase can begin. The system now monitors new vulnerability disclosures in near real time and chooses an appropriate alert level for them using the knowledge base it has constructed together with the expert.

The processing of a vulnerability is made of several building blocks, some of them common to both the training and inference phases, and some specific to one or the other, as detailed below. During the inference phase, evaluating the risk created by a new vulnerability is done by comparing it to similar, annotated vulnerabilities. To this end a *similarity metric* between vulnerabilities is needed. This similarity metric is made possible by viewing vulnerabilities as euclidean vectors through a *featurization stage*.

During the training phase, selecting relevant vulnerabilities to be annotated by the expert is based on the same building blocks. Our similarity metric is used to compute a *similarity matrix* for the entire past vulnerabilities dataset, which then lets the learning system compute an *information density metric* for every past vulnerability. Selecting a vulnerability to be annotated by the expert is done by combining this information density metric with an *uncertainty metric* computed by simulating an alert decision process for every vulnerability while measuring the confidence of the prediction.

In the rest of this section we first present the concepts necessary for both training and inference: in Section 6.2.1 we describe the possible alert levels for a vulnerability and the user interface used by the expert. In Section 6.2.2 we discuss the featurization stage of our architecture. In Section 6.2.3 we discuss our choice of similarity metric.

The actual alert decision process used during the inference phase is then presented in Section 6.2.4.

We finally cover the remaining concepts necessary for the training phase. In Section 6.2.5 we present our choice of uncertainty metric. We then discuss concepts proposed by Settles et al [182] that we use: first the similarity matrix and information density metric in Section 6.2.6, then in Section 6.2.7 the process used by the learning system to select the next vulnerability to be annotated by the expert during the training phase.

6.2.1 Alert Levels

An alert level scale is needed to let both the system and the expert translate their risk analysis of a vulnerability into an actionable decision. The recipient of this alert is a human, so it makes sense for the alert scale to be human-centered. We use the scale proposed by Google in their *Site Reliability Engineering* doctrine [18]:

— **LOG:** The disclosure is *logged* in an activity log that can be consulted afterward, but no alert is actively raised to a human security operator.

— **TICKET:** The disclosure causes the creation of a *ticket* in an issue tracking system. A ticket is expected to be solved in a non-urgent manner during working hours.

— **PAGE:** The disclosure causes an on-call security operator to be *paged* immediately, even outside regular working hours. A page is expected to be treated immediately.

As shown in Figure 6.2, in the training phase the system selects a vulnerability to be labeled by the human expert then shows its description on screen. The expert then selects the alert level she deems the most appropriate for the vulnerability, in the context of the

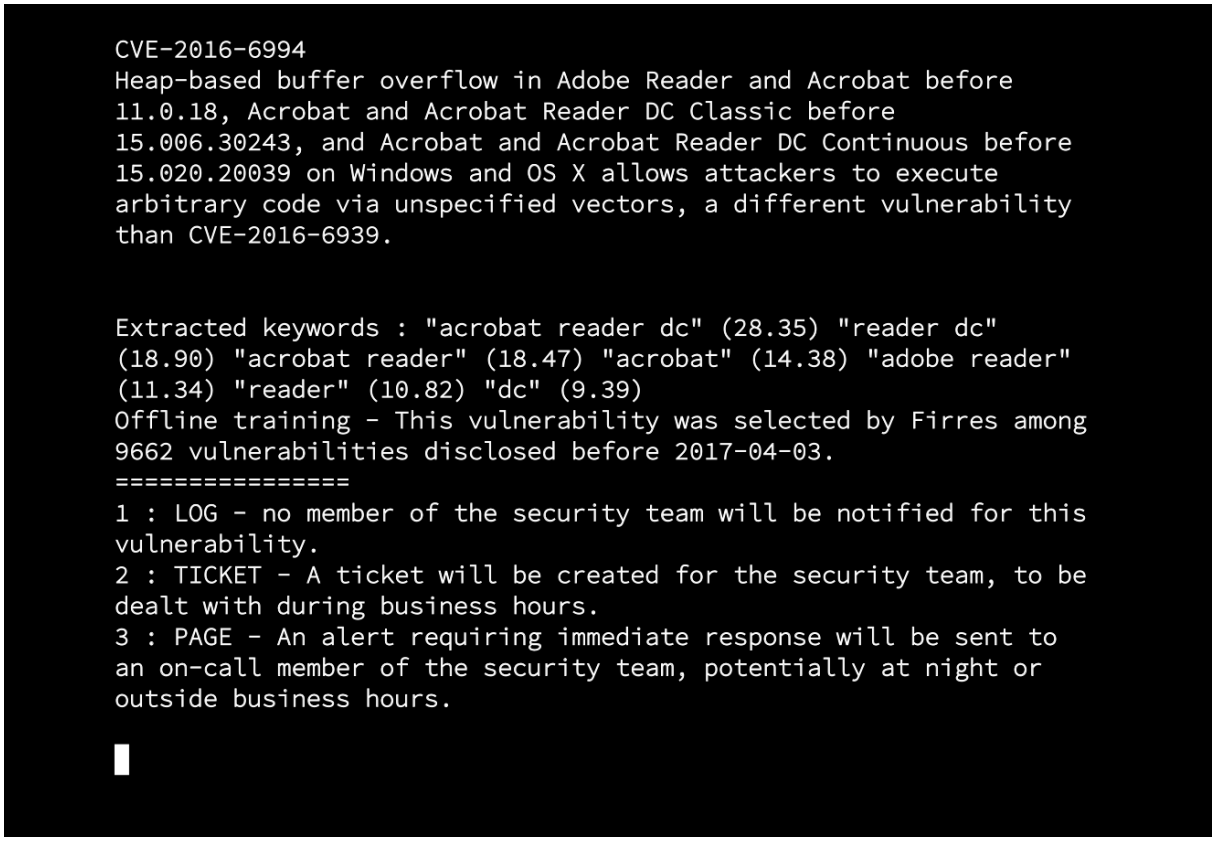

Figure 6.2 – A screen capture of the user interface (UI) of our prototype Firres (FIRst RESponder) during the active learning phase. For a vulnerability to be labeled, the UI displays its CVE ID, its description, the weighted keyword list extracted from the description. The expert can then select an appropriate alert level by entering 1 (for LOG), 2 (for TICKET) or 3 (for PAGE).

system she is tasked to protect. In the inference phase, the system automatically selects the appropriate alert level for new vulnerabilities using the same alert scale.

6.2.2 Vulnerability as a Euclidean Vector

In order to transform the raw description of a vulnerability into a euclidean vector, we reuse the same keyword extraction pipeline as described in Chapter 4: we apply a bag-of-words on the text description to get a dimension for each word, and the words are then filtered to retain only words present in past CPE URIs. The retained words are then weighted using TF-IDF and we apply all domain heuristics presented in Chapter 4 to improve the weighting. We finally use our euclidean truncation scheme to reduce the number of non-zero components for every vector. As shown in Chapter 4, this pipeline lets us identify the affected software for a vulnerability in the form of a *sparse* euclidean vector (with a low number of non-zero vector components). Vulnerabilities affecting the same software share many non-zero components, making them similar in the vector space.

The CVSS prediction contribution presented in Chapter 5 was not integrated here because of lack of time. We come back to this in Section 6.4.2.

6.2.3 Measuring Vulnerability Vectors Similarity

As a similarity metric between vulnerability euclidean vectors we choose the *cosine similarity* (presented in Section 2.4.4) that can be computed using the formula first shown in Equation 2.2 and shown again here:

$$\text{cosine similarity}(\mathbf{x}, \mathbf{x}') = \frac{\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}'_i}{\|\mathbf{x}\| \|\mathbf{x}'\|}, \tag{6.1}$$

Cosine similarity returns a similarity between 0 and 1 for two vectors. It has one main advantage compared to distance-based similarity metrics such as the RBF kernels (presented in Section 2.4.4): it returns a similarity of zero for vectors having no common non-zero components. This is helpful to avoid creating artificial similarity between unrelated vulnerabilities. In particular, distance-based similarity metrics tend to overestimate the similarity between vectors close to origin. In our case, a vector close to origin indicates a vulnerability for which we have not extracted any meaningful keyword and therefore have not a strong understanding of. Therefore having two vectors being close to origin does not necessarily means that the two related vulnerabilities are actually similar to each

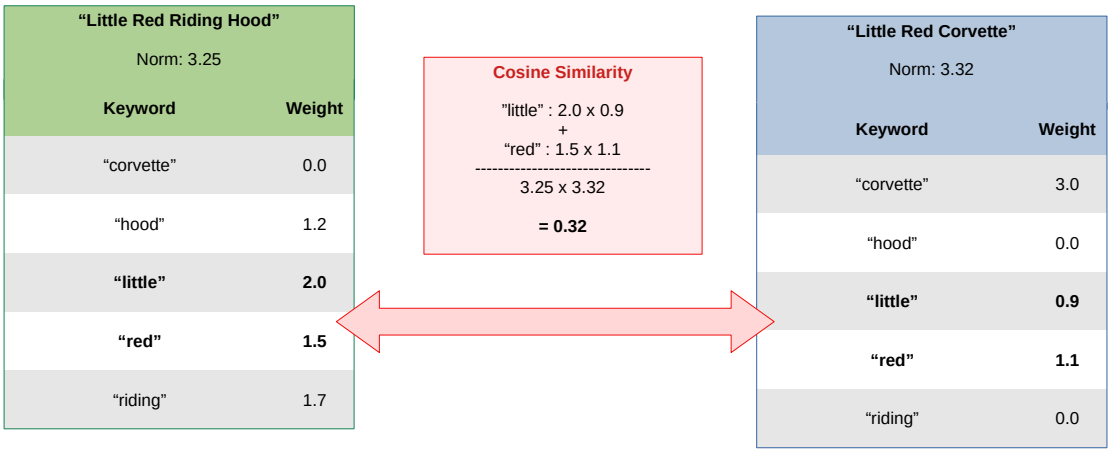

Figure 6.3 – Example of cosine similarity calculation between two weighted keyword vectors similar to the the ones generated after vulnerability featurization.

other. Another advantage of cosine similarity is to not require any hyperparameter, making it more explicable and stable than similarity metrics that do. An example of cosine similarity calculation is shown in Figure 6.3.

6.2.4 Alerting Decision Score

For every new vulnerability disclosure, the prediction system has to make a risk analysis by choosing an appropriate alert level for the vulnerability, among LOG, TICKET and PAGE. In order to do that, we had to make several hypotheses. These hypotheses can be challenged, as we discuss in Section 6.4.

Our first hypothesis is that the TICKET alert level is a reasonable default reaction for a vulnerability about which nothing is known. This is based on the assumption that security operators are *risk averse* and want to eventually review any vulnerability for which the prediction system could not make a meaningful decision.

In the light of this hypothesis, we propose an *alert decision score* in range $[-1; +1]$. It

can be converted into an *alert decision* for a vulnerability v using Equation 6.2.

$$\text{decision}(v) = \begin{cases} LOG, & \text{if score}(v) \in [-1; -\frac{1}{3}[\\ TICKET, & \text{if score}(v) \in [-\frac{1}{3}; +\frac{1}{3}[\\ PAGE, & \text{if score}(v) \in [+\frac{1}{3}; +1] \end{cases} \tag{6.2}$$

Conversely, a default alert decision score is set for all annotated vulnerabilities. The alert decision score of an annotated vulnerability av is set following Equation 6.3.

$$\text{score}(av) = \begin{cases} -1, & \text{if the vulnerability is annotated as LOG} \\ 0, & \text{if the vulnerability is annotated as TICKET} \\ +1, & \text{if the vulnerability is annotated as PAGE} \end{cases} \tag{6.3}$$

The formula to compute the decision score of a non-annotated vulnerability v is shown in Equation 6.4, assuming a knowledge base of k annotated vulnerabilities av_0 to av_{k-1}:

$$\text{score}(v) = \begin{cases} \dfrac{\sum_{i=0}^{k-1} \text{score}(av_i) \times \text{similarity}(v, av_i)}{\sum_{i=0}^{k-1} \text{similarity}(v, av_i)}, & \text{if } \exists i \in \{0, ..., k-1\} \quad \text{similarity}(v, av_i) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.4}$$

Put another way, the alert decision score is the average decision score for all annotated vulnerabilities weighted by their similarity to the analyzed vulnerability, with a special case of returning 0 when there are no similar annotated vulnerabilities yet.

This formula implies that any vulnerability for which there are few or no similar annotated vulnerabilities gets an alert decision score close or equal to 0, resulting in a TICKET alert, in line with our hypothesis that this is the correct default reaction. Conversely, LOG and PAGE alerts are only emitted if the vulnerability is strongly similar to known vulnerabilities that were annotated as such.

6.2.5 Uncertainty Score

The active learning architecture we use [182] requires an *uncertainty metric* for each vulnerability, as part of the vulnerability selection process used to submit vulnerabilities to be annotated by the expert (which we discuss in Section 6.2.7). However they do not

	A	B	C	D	ID
A	1.0	0.0	0.6	0.4	0.5
B	0.0	1.0	0.0	0.0	0.25
C	0.6	0.0	1.0	0.1	0.42
D	0.4	0.0	0.1	1.0	0.37

Table 6.1 – Example of similarity matrix and information density metric (ID) for a set of four vulnerabilities.

specify how to implement it.

We make the hypothesis that since TICKET is our default reaction, the closer to TICKET a vulnerability is, the more uncertain our decision is. Conversely, the closer to LOG or PAGE a decision is, the more certain our decision is. Therefore we compute uncertainty using Equation 6.5.

$$\text{uncertainty}(v) = 1 - |\text{score}(v)| \tag{6.5}$$

Therefore the uncertainty score of a vulnerability reaches 1 the closer its decision score is to 0 (TICKET), and reaches 0 the closer its decision score is to -1 or $+1$ (LOG or PAGE, respectively).

6.2.6 Similarity Matrix to Measure Information Density

We first describe as background the concepts of *similarity matrix* and *information density* proposed by Settles et al [182], as they are part of our architecture. We then present a slight modification to the scheme we add in the context of a dynamic dataset such as ongoing vulnerability disclosures.

Background: Similarity Matrix and Information Density

Assuming a set of n past vulnerabilities v_0 to v_{n-1}, a similarity matrix S of size $n \times n$ can be constructed using Equation 6.6.

$$\forall i, j \in \{0, ..., n-1\} \quad S_{i,j} = \text{similarity}(v_i, v_j) \tag{6.6}$$

From the similarity matrix S we compute an *information density* metric for every vulnerability by computing the average similarity of a vulnerability to every other vulnerabilities in the dataset, including itself (therefore no vulnerability can have an information

density of zero). This is described by Equation 6.7.

$$\forall i \in \{0, ..., n-1\} \quad \text{density}(v_i) = \sum_{j=0}^{n-1} \frac{S_{i,j}}{n} \tag{6.7}$$

Information density is a measure of how *typical* a vulnerability is. When a lot of vulnerabilities resemble each other, they all have a high similarity metric when compared to each other, increasing the information density of each of them. Conversely, an outlier vulnerability with few or no similar vulnerabilities has a low information density. An example of information density calculation can be found in Table 6.1.

Vulnerability Dataset Truncation

Settles et al proposed the concept of similarity matrix with a static dataset in mind. However, as many new vulnerabilities are disclosed every day, in our case the dataset is dynamic and the creation and update of a similarity matrix are quadratic through time and space. To avoid that, we modify the scheme proposed by Settles et al and decide to truncate our vulnerability dataset once it goes over a certain size. We define a hyperparameter N for the maximum number of vulnerabilities we find acceptable, then remove excess vulnerabilities once $n > N$. For choosing the vulnerabilities to be truncated we simply remove the oldest vulnerabilities first.

6.2.7 Selecting a Vulnerability to be Evaluated

The main decision task of the training phase is to select the next vulnerability to be annotated by the expert. This raises several challenges. Selecting only the most typical vulnerabilities can lead to submit many vulnerabilities similar to each other, with diminishing returns after the first few ones. Conversely, selecting vulnerabilities only based on decision uncertainty leads to submit many outliers to the expert, which does not help with getting a broader understanding of the entire vulnerability set. Another question is how much the selection process should be based on randomness. A solely deterministic process could get the learning stuck in local maxima of the vulnerability euclidean space, while leaning too much on randomness might lead to submit too many uninteresting vulnerabilities to the expert.

To address all these challenges, we propose an approach based on the work of Settles et al [182] which we present as background. We then propose a modification to incorporate

randomness in the selection strategy.

Background: Selection Score and Selection Strategy

Settles et al [182] proposed that every vulnerability eligible for annotation should be given a *selection score* according to a *selection strategy*. They proposed several strategies for the selection score and we choose the simplest one as a starting point, shown in Equation 6.8. This strategy does not include randomness yet, which we describe in the next section.

$$\text{select}(v) = \text{density}(v) \times \text{uncertainty}(v) \tag{6.8}$$

As seen in Section 6.2.5, when a vulnerability has no similar annotated vulnerabilities, it gets an uncertainty score of 1. Therefore, at the beginning of the training (when there are no annotated vulnerabilities yet) all vulnerabilities get the same uncertainty score, and the selection process is based on information density only (we show below how we add randomness). A large group of vulnerabilities similar to each other (and therefore with high information density) is called a vulnerability *cluster*. Once more vulnerabilities get annotated inside the biggest clusters, the uncertainty score of the unannotated vulnerabilities in the clusters progressively decreases as there are similar annotated vulnerabilities in the knowledge base. At some point, other smaller clusters with lower information density but higher uncertainty (as they are unexplored yet) reach higher selection scores and get picked up first.

Adding Randomness to the Selection Process

With no randomness at all, the vulnerability selection is entirely deterministic, assuming a fixed set of vulnerabilities and an expert always choosing the same alert level for the same vulnerability. This can create problems as some vulnerability clusters are much denser than others, leading the selection process to waste too much of the expert's time on too few clusters. However, selecting a vulnerability from a smaller pool of randomly chosen vulnerabilities allows for more exploration of the vulnerability dataset as the denser clusters are not always present in the random pool. Therefore in our work we slightly modify the selection strategy proposed by Settles et al [182] to incorporate a certain amount of randomness by randomly selecting r vulnerabilities from the set of all candidates vulnerabilities to be annotated, before applying the rules described in 6.8

to choose a vulnerability from this random subset. r is a hyperparameter. We claim that taken together, randomness, uncertainty, and information density provide the foundation for a robust annotation selection process.

Our complete training algorithm, including the vulnerability selection process is described in Algorithm 1.

```
Input: V: set of all past vulnerabilities
Input: B: annotation budget for the current training phase
Input: ONLINE: boolean set to true if the current training phase is online, false
       if offline
Input: r: size of the random subset of vulnerabilities
while B > 0 do
    C = get_all_unannotated_vulnerabilities(V);
    if ONLINE then
        C = retain_only_vulnerabilities_disclosed_in_last_period(C);
    end
    R = select_random_subset_of_vulnerabilities(C, r);
    v = select_vulnerability_with_highest_selection_score(R);
    submit_vulnerability_to_expert(v);
    B = B - 1;
end
```

Algorithm 1: Training algorithm, including the vulnerability selection process.

6.3 Preliminary Evaluation

Our evaluation goal is to check the real-world applicability of our prediction system, designed and configured with preliminary hypotheses, by confronting it to actual security experts in charge of real information systems. In our experimental protocol, these experts annotate vulnerabilities to assert their risk levels, in the context of the systems they are responsible for. The experiment is divided in two steps: in the first step, security experts train the prediction system, through an offline training followed by an online training, by simulating the passing of time. Second, they evaluate the prediction system: this is done by randomly selecting vulnerabilities, then submitting them to the prediction system and the expert simultaneously while comparing their answers. The main difference between the two steps is that during training the system controls which vulnerabilities are submitted to the expert, and the resulting annotations are added to the system's knowledge base. On the contrary during evaluation the vulnerability selection is completely random and

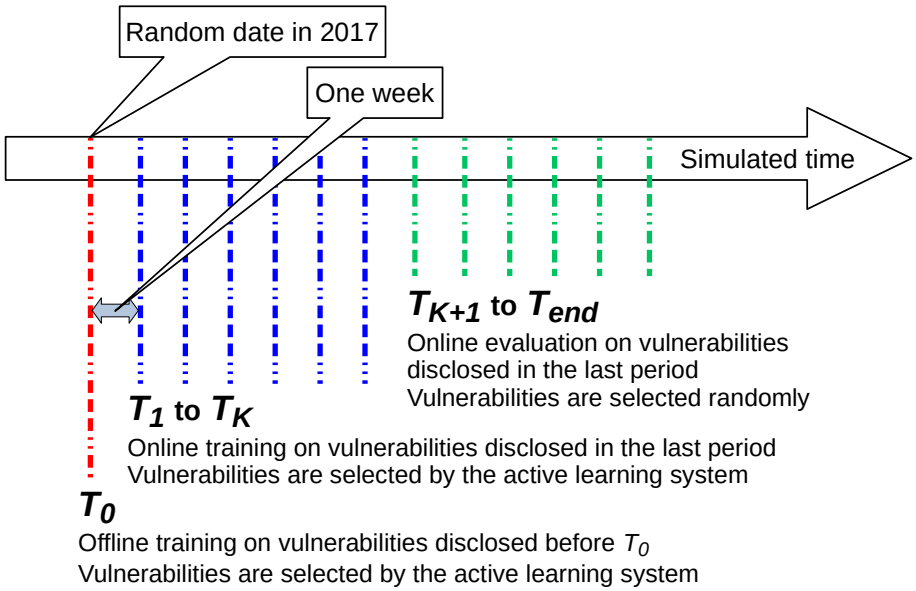

Figure 6.4 – Our experiments simulate the passing of time starting from a random date in 2017, using weekly time steps.

the expert's annotations are only used as an evaluation tool without being added to the system's knowledge base.

A major challenge of this experiment is the requirement for busy security experts to commit a certain amount of time to participate, such as half a day. Moreover, as the training's vulnerability selection process simultaneously depends on the active learning architecture, its hyperparameters, and the expert previous choices, any iteration in the design or configuration of the prediction system requires to undertake a brand new experiment (including the expert full participation) to properly evaluate the changes.

In the end we have been able to complete two experiments, which we denote Experiment A and B in the rest of this chapter. The information system studied in experiment A is a server-side software development environment, including a git server, an issue tracker, NTP and DNS servers, as well as Windows and Linux machines communicating through the Kerberos protocol. The information system studied during experiment B was also an information system for software development, with Linux machines only. It includes a git server, a Jenkins server, an issue tracker, a wiki system, and is administered through SSH.

A shortcoming of this preliminary evaluation is that both experiments were done as

part of a first experimental campaign, and we have not been able to secure the launch of a second experimental campaign based on what we learned during the first one. This campaign was based on initial hypotheses we took, some of which turned out to be questionable, resulting in mixed evaluation results. This is further discussed in Section 6.4.

We present our evaluation protocol in Section 6.3.1 and the results of the experiments in Section 6.3.2.

6.3.1 Experimental Protocol

Our experimental protocol uses past CVE vulnerabilities to simulate the production deployment of an alert system at a random past date. The protocol starts with an initial offline training session, then simulates the passing of time for an online training session, followed by an online evaluation session. Figure 6.4 shows the outline of an experiment.

A major experimental constraint was that vulnerability annotations authored by our participants were actually sensitive themselves, as they provided a lot of insights into the related information systems. This required the whole campaign to be carried out without us ever seeing those annotations, or storing them on our own hardware. Therefore experiments were done exclusively on security experts' own machines (usually laptop workstations), preventing the use of dedicated server-grade computing resources.

For this reason the vulnerability selection process, which is computationally intensive, had to be bounded in time and resources, as the experience had to stay interactive enough for the security expert to stay engaged while running entirely from a lightweight workstation. This is not straightforward as a similarity matrix for the entire CVE dataset requires hundreds of megabytes of storage, takes minutes to compute, and is only valid for a specific time step in the simulation as new vulnerabilities and keywords are added and discarded. We solved this engineering problem by storing all similarity matrices needed for every time step of the simulated timeline on the experts' workstations, ensuring the interactivity of the experience at the cost of each experiment requiring several hundreds of gigabytes of storage.

The experimental protocol was prepared in 2019, giving us access to all CVE vulnerabilities disclosed between 2007 and 2018. However, as explained in Section 6.2.6, the computation and storage of the vulnerability similarity matrix is quadratic in the number of vulnerabilities in our database, and differs at every time step (as more vulnerabilities are disclosed and more CPE URIs are added to the featurization word list). The hyperparameter N, first described in Section 6.2.6, is the maximum number of vulnerabilities the

Hyperparameter	Value	Justification
Active Learning parameters		
Maximum size of vulnerability dataset (N)	10000	Maximum value allowing the experiment to stay interactive.
Size of randomness pool for vulnerability selection (r)	10	Informal tests suggest randomness could add robustness to the selection process.
Time step duration	7 days	Experiment is not tractable on a workstation with daily time steps and would not be granular enough with monthly time steps.
Total annotation budget	300	Maximum budget acceptable for participants.
Offline training annotation budget	100	Initial hypothesis: offline training should be allocated a third of the total annotation budget.
Online training budget per time step	8	Initial hypothesis: online training should be allocated a third of the total annotation budget. However informal tests suggest having more time steps with fewer vulnerabilities in each of them could add robustness to the training process.
Online training number of time steps	12	Initial hypothesis: online training should be allocated a third of the total annotation budget. However informal tests suggest having more time steps with fewer vulnerabilities in each of them could add robustness to the training process.
Online evaluation budget per time step	10	Initial hypothesis: online evaluation should be allocated a third of the total annotation budget.
Online evaluation number of time steps	10	Initial hypothesis: online evaluation should be allocated a third of the total annotation budget.
Featurization parameters		
Multiple-words heuristic	enabled	see Chapter 4 results
Capitalized words heuristic	enabled	see Chapter 4 results
"lib" heuristic	enabled	see Chapter 4 results
Euclidean truncation target	95 %	see Chapter 4 results
Metadata publication delay	7 days	Initial hypothesis for a typical delay. In Chapter 4 experimental protocol the delay was eventually changed from 7 to 60 days as the latter is closer to a worst-case scenario while 7 days is closer to a median scenario according to real-world NVD delays. However Chapter 4 results should hold with 7 days as shorter delays are less strict than longer ones.

Table 6.2 – All hyperparameters necessary for our evaluation protocol, the values chosen for this first experimental campaign, and a justification for each value.

prediction system retains in its database. We empirically found that setting $N = 10000$ was close to the highest value we could set while keeping the user experience interactive using our current prototype. As we discard older vulnerabilities first and more than 5000 vulnerabilities have been disclosed every year for the last decade, in practice only vulnerabilities disclosed between 2014 and 2018 are actually used in the experiment. The featurization stage hyperparameters were set to the same values as in the experimental protocol of Chapter 4, except for the metadata publication delay (see Table 6.2).

Regarding the annotation budget, we had initial discussions with potential participants about the amount of time they would be able to commit to the experiments. From early participant feedback and preliminary empirical tests, we converged on an experimental duration of half a day consisting in annotating 300 vulnerabilities, an amount to be divided into offline training, online training, and online evaluation. This budget was the best compromise between participant's availability and experimental validity.

Our first intuition was to divide the annotation budget evenly between offline training, online training, and online evaluation, providing a budget of 100 annotations for each step. For online training and evaluation, we also have to divide the budget into a number of weeks of simulated time and a number of vulnerabilities to be annotated per week. For online evaluation we settled on an even divide of 10 annotations per week during 10 weeks of simulated time. However early informal experiments highlighted a potential problem for online training: it is common for many vulnerabilities affecting the same software or hardware to be disclosed simultaneously, creating a risk of wasting part of the training budget as most vulnerabilities disclosed in a single period are affecting the same component. For this reason, we decided to spread the online training budget on more weeks, with only 8 annotations per week during 12 weeks of simulated time, for a total of 304 vulnerability annotations in the entire experiment.

The last hyperparameter to set was r, the number of randomly picked vulnerabilities to be ranked during the vulnerability selection process. Early informal experiments highlighted the risk of getting stuck into local maxima when carrying a mostly deterministic vulnerability selection process. Therefore we decided to let randomness have an important role in vulnerability selection by setting $r = 10$. We hypothesized that this setting would let the training process propose varied vulnerabilities to the expert, while still having a high probability of having at least one interesting vulnerability to submit to the expert among the ten random ones.

For reference Table 6.2 shows a complete list of all the hyperparameters in our eval-

	System decision		
Expert decision	**LOG**	**TICKET**	**PAGE**
LOG	46	39	0
TICKET	4	2	0
PAGE	5	4	0

Table 6.3 – Evaluation results for experiment A. Correct decisions: 48. False positives: 39. False negatives: 13.

	System decision		
Expert decision	**LOG**	**TICKET**	**PAGE**
LOG	35	36	6
TICKET	5	12	1
PAGE	2	3	0

Table 6.4 – Evaluation results for experiment B. Correct decisions: 47. False positives: 45. False negatives: 10.

uation protocol, as well as the values they were set to and the justifications for these values.

6.3.2 Results

Evaluation results for experiments A and B are shown in Table 6.3 and 6.4. Correct decisions are shown in the diagonal row. False positives (for which the system chose a higher alert level than the expert) are shown above the diagonal, while false negatives (for which the system chose a lower alert level than the expert) are shown below.

Results show that our approach can be improved: less than half of the decisions are identical between the system and the expert, with a very high number of false positives yet a significant amount of false negatives. The most common type of false positives is a vulnerability deemed benign by the expert (LOG, no alert raised) for which the system created a non-urgent TICKET for inspection during working hours. This is a consequence of the low base rate of non-LOG vulnerabilities (experts for experiment A and B respectively classified 85 % and 77 % of evaluation vulnerabilities as LOG) coupled with our design choice of favoring TICKET as a default response.

As we said in Section 6.3.1, we did not get access to the expert's annotations for neither experiment A or B. However, for experiment B we had the opportunity to follow up with the expert's team to get some insight into the evaluation process. An important discovery

made during this follow-up is that at least five false negatives from experiment B are actually human errors instead of prediction errors (the five vulnerabilities annotated as PAGE by the expert were incorrectly annotated during the evaluation and the participant actually agrees with the system's decisions). According to the participant, this is partly due to a specific keyword in the vulnerability description that incorrectly startled him, and partly due to annotation fatigue, which we discuss in Section 6.4. We chose not to correct our results following this information for two reasons: first, there have been no comprehensive review of all evaluation vulnerabilities for neither experiments, leaving the possibility of even more undetected human errors. Moreover, we consider expert error a real-world condition that should be acknowledged by a robust evaluation protocol.

6.4 Discussion

As said previously, for both experiments A and B we did not get access to the 304 vulnerabilities submitted to the expert, nor the annotations provided by the expert, for confidentiality reasons. Under these conditions we can only speculate about what went right or wrong during the automated decision process, and can outline some design changes we would propose in the advent of another experimental campaign.

6.4.1 Experimental Limitations

It is likely that a budget of 304 vulnerabilities is not enough for both training and evaluating our prediction system. It is noteworthy that in the context of a production prediction system, having a security operator annotate vulnerabilities one hour a week for a year would result in an order of magnitude increase in annotation budget compared to our experimental campaign. We do not know how such an increase in budget would affect decision accuracy.

Moreover for the sake of simplicity we made the choice to completely separate online training from online evaluation, while still simulating the passing of time. This could have the effect of progressively making the training obsolete the further the evaluation progress in the simulated time. In a real-world scenario, online training would likely continue indefinitely in parallel of decision evaluation and exploitation. We do not know the impact of this experimental choice on the validity of our results.

6.4.2 Decision Quality

For this work we only used dimensions generated by the keyword extraction pipeline described in Chapter 4, with past CPE URIs used as a filtering whitelist to prevent too many noisy keywords. This raises two risks: the quality of the extracted names may not be good enough for accurate decisions and comparing vulnerabilities by affected component alone may not be enough for proper risk analysis.

As we could not study ourselves the training and evaluation data, we delivered a debugging tool to experimenters allowing them to look back at any evaluation vulnerability. Using this tool they could list the training vulnerabilities that were considered similar, and the keywords from which the similarity arose. While they did not have time to do a comprehensive analysis of all evaluation decisions, they did report some valuable insights. Many decision mistakes were due to incorrect links between vulnerabilities, due to noisy keywords that are nevertheless present in the CPE URI whitelist: this is the case for common technical words such as "internet" (present in the whitelist because of many software names starting with *Internet Explorer*), "api" (because of software names such as *Broker API* or *API connect*), "credentials" (because of software names such as *Credential Bindings*). As seen in Chapter 4, multiple words structures that are part of software names mentioned in CPE URIs can be included as multiple words keywords, creating some superfluous keywords made of very common English sentence structures: structures such as "to the" (included in the whitelist because of a vulnerability affecting the *Tic Tac to the Max* Android application), "to a" (because of a software named *Recommend to a Friend*), and "as a" (because of a software named *Metal as a Service*) are all superfluous keywords that led the prediction system to incorrectly pair many vulnerabilities together, leading to several wrong decisions.

It could be worthwhile to improve the filtering capabilities of our keyword extraction pipeline, as well as giving security experts the opportunity to blacklist keywords that led to incorrect decisions in the past. Last but not least, it would be interesting to evaluate how adding other vulnerability properties such as the predicted CVSS score (as we propose in Chapter 5) would help risk analysis.

6.4.3 Expert Expectations

When explaining our experimental protocol to participants we did our best to convey that the current prediction system is only based on affected component names, and does

not take into account other properties such as affected versions or vulnerability severity. However we never met directly the participant for experiment A (we presented the protocol to his colleagues) and more than a year had passed between initial project presentation and the starting of experiment B. This means participants may have forgotten (or were never aware to begin with) of this limitation while participating in the experiment, and participant's feedback following the experiments left this point unclear. If it was confirmed that security experts tend to include other information such as version or severity into their vulnerability analysis, this would be a strong indicator that more vulnerability properties are needed to increase prediction accuracy.

Another point where our system may have differed from participants' expectations is our choice to consider TICKET as a default choice. Our initial hypothesis was that false negatives are worse than false positives, as they imply the presence of an unmitigated risk for the information system. Many security experts we talked to disagreed with us: most of them told us that they had too many alerts from too many monitoring systems to handle them all, and they did not have time to check all newly disclosed CVE vulnerabilities manually. Therefore they strongly prefer some false negatives (which is not worse than their current situation) to false positives (which create even more alerts for them to handle). While there is probably no universal answer to this question, we believe a future version of this prediction system should allow the possibility for the security expert to choose a default alert behavior she feels appropriate.

6.4.4 Hyperparameters Tuning

Our learning system and evaluation protocol both require many hyperparameters, creating a hyperstate space too vast to be explored comprehensively. This is a problem shared by many machine learning endeavors, as changing a hyperparameter value requires the training to be started over from scratch. However as seen in Section 2.4.3 active learning is especially vulnerable to this phenomenon as training requires the active participation of a human, and the choice of a sample to be annotated during training is impacted by the hyperparameters settings. As security experts' time is expensive, any hyperparameter mistake is very costly. The list of hyperparameters shown in Table 6.2 illustrates how many big and small decisions must be taken to prepare such an evaluation protocol (sometimes without the ability to justify these decisions appropriately).

Even if we had the ability to launch a second experimental campaign, or even five other experimental campaigns, exploring the hyperstate space thoroughly would be still

out of our reach as it grows exponentially with the number of hyperparameters in our system. When the participating human is expected to be a domain-knowledge expert (as in our case), retraining the model is very expensive and doing it repeatedly is not possible in practice. Therefore there is no way to tune such a system following a rigorous experimental method. This realization is the reason why we chose to limit the number of hyperparameters as much as possible in our contributions described in Chapters 4 and 5.

6.4.5 Decision Explicability

In our opinion, our lack of access to this campaign's experimental data validates our design choice to consider decision explicability as a critical property of a security decision system. This enables us to design self-serving debugging tools usable by participants, giving them the ability to understand by themselves the reasons why predictions succeeded or failed. The participants were able to give us all the valuable insights we discussed in this section while still keeping the sensitive details of their information system confidential.

Figure 6.5 shows a concrete example of an alert decision explicability report, from an informal experiment done in the same conditions as experiments A and B. The evaluation vulnerability, CVE-2014-4914, is a vulnerability affecting Zend, a PHP framework. On a superficial level, the decision seems correct: the expert and the prediction system chose the same alert level (LOG), and overall the keywords extracted from the vulnerability are of good quality. However, when studying the reason why the prediction system chose the LOG alert level for this vulnerability, it appears that it was considered similar to CVE-2017-3614, a vulnerability affecting Berkeley DB annotated as LOG during training. The keyword creating a link between the two vulnerabilities is "db", as the Zend vulnerability is due to a database connection problem. One may consider this link to be correct, for instance if an expert wanted to classify all database-related vulnerabilities as LOG because her information system does not include any database. However it is likelier to be considered incorrect: the expert probably annotated the training vulnerability as LOG because she was not interested in Berkeley DB, and this should not allow the system to draw any conclusion on the expert's interest in the Zend framework. In that case the system reached the correct decision accidentally because of the low base rate of non-LOG vulnerabilities, but it should actually have chosen TICKET as there is no genuinely similar vulnerability in the training data. We can also notice that the keywords extracted from the Berkeley DB training vulnerability are generally of lower quality compared to the Zend evaluation vulnerability. This is mainly due to the inclusion of the CVSS vector of the

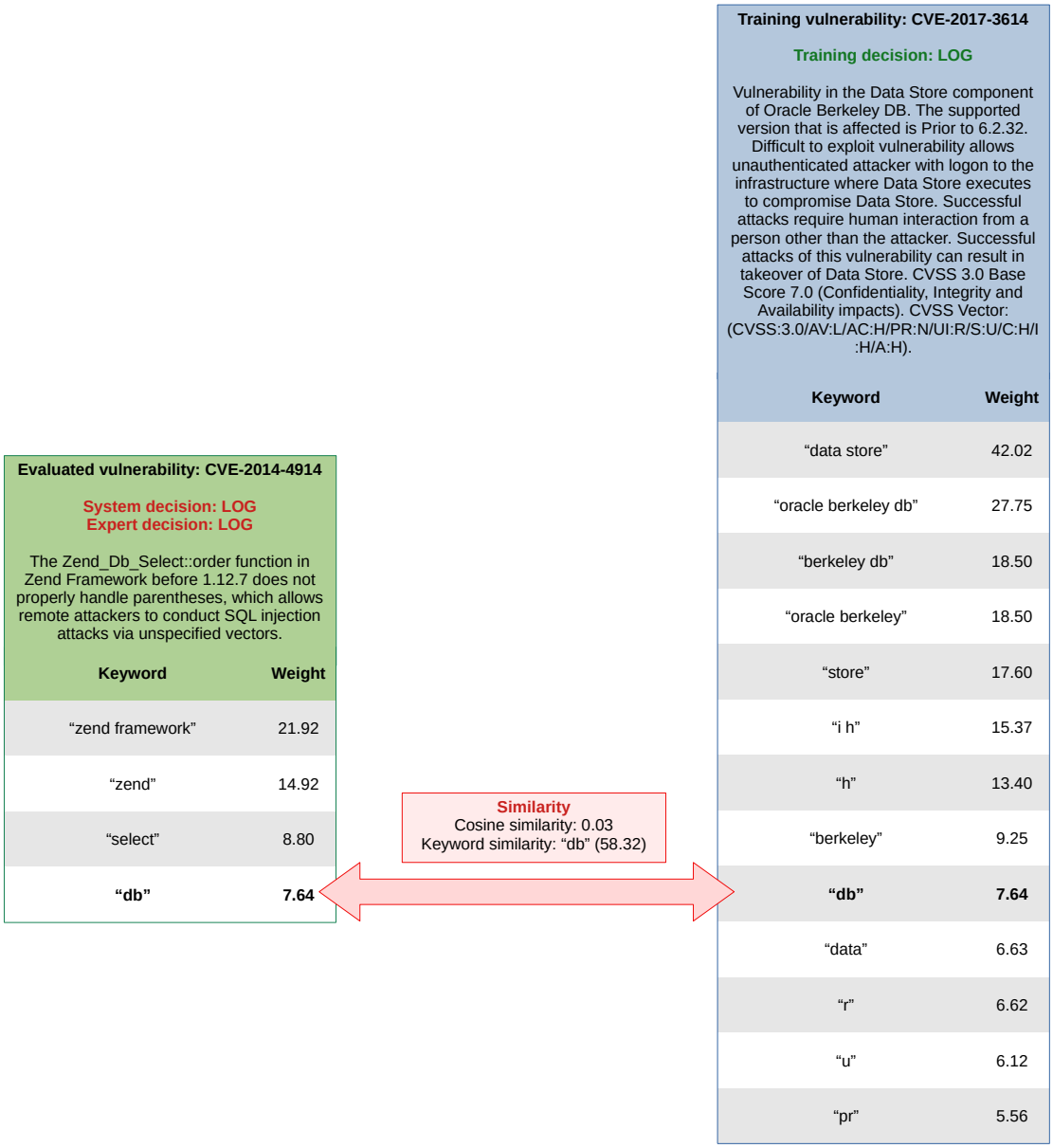

Figure 6.5 – An actual example of an alert decision explicability, from an informal experiment done in similar conditions to experiments A and B. See Section 6.2.3 and Figure 6.3 for more details about cosine similarity calculation.

vulnerability directly inside its raw text description, which is a non-standard practice yet occurring from time to time. An expert studying this report could conclude that adding a data-cleaning step in order to detect and remove CVSS vectors from vulnerability text descriptions could help increase decision quality.

We can see from this example that decision explicability is a powerful tool that lets us quickly understand how and why the prediction system made a certain decision, and suggests steps to improve decision quality. As discussed in Section 6.4.2, a natural next step would be to give the expert the tools to improve by themselves the quality of a decision after having diagnosed its root cause. A proposal that is simple for experts to understand and easy for us to implement would be to allow the expert to manually blacklist a low-quality keyword, for instance after having debugged an incorrect alert decision.

6.5 Conclusion

In this chapter we presented the initial design of an automated vulnerability risk analysis prediction system that can immediately choose an appropriate alert level for a just-disclosed vulnerability. It is interactively trained by a security operator using active learning: by mimicking the alert decisions of the human operator, the prediction system can gradually learn which vulnerabilities are considered important in the context of a specific information system. The system is based on an active learning architecture proposed by Settles et al [182] that includes the computation of a cosine similarity matrix and an information density vector. We modified this architecture to handle dynamic datasets such as the flow of vulnerability disclosures and to add a controllable amount of randomness in the vulnerability selection process. The architecture also uses our own work on vulnerability keyword extraction (described in Chapter 4) as a featurization scheme.

This contribution is still at an early stage. Our experimental results show some of our initial design choices could be revisited. However, security experts we collaborated with expressed great interest in the concept and their collaboration was a crucial aspect of this contribution. Notably, this partnership highlighted how decision explicability is an important factor for real-world applicability of machine learning for security alerting. Going forward we would like to carry on this relationship onward as well as extend it to include even more participants, opening the door to a more iterative research process and more sound experimental protocols. Validating such a system would be an important step forward in the defense against n-day vulnerabilities, allowing organizations to set up

an around-the-clock vulnerability monitoring system while only requiring their security operators to work during regular hours to train the system.

Chapter 7

Conclusion and Perspectives

In this last chapter of this thesis, we first summarize our contributions then present the perspectives opened by our work.

7.1 Contributions

The goal of this thesis is to propose novel tools to automatically analyze and react to the risk posed by n-day vulnerabilities at disclosure. Our ultimate goal is for an information system to exhibit *constant security over time*: daily vulnerability disclosures should not affect the overall risk faced by the system.

To fulfill these goals we presented four contributions in this thesis. Our first contribution (see Section 7.1.1 below) is a proposal for an end-to-end strategy for defending information systems against n-day vulnerabilities. Our three other contributions solved research problems highlighted by this strategy and are divided in two topics: automating vulnerability analysis at disclosure (see Section 7.1.2) and automating vulnerability risk analysis at disclosure in the context of a specific information system needing protection (see Section 7.1.3). In Section 7.1.4 we review how our contributions fulfill our goals and improve the state of the art.

7.1.1 End-to-end Strategy for Defending Information Systems Against N-Day Vulnerabilities

In Chapter 3 we outlined a complete strategy to defend information systems against n-day vulnerabilities. The core steps of the strategy are as follows:

1. Automatically gathering and analyzing information about newly disclosed vulnerabilities in an explicable fashion, without waiting for human analysis that comes later.

2. Automatically analyzing the risk created by newly disclosed vulnerabilities, in an explicable fashion, in the context of a specific information system to defend.
3. If warranted, triggering a reaction mitigating the risk posed by the vulnerability to the information system.

Outlining this strategy highlighted several open research problems, some of which we tackled in our other contributions.

7.1.2 Automated Vulnerability Analysis at Disclosure

In Chapters 4 and 5 we proposed two contributions to gain a better understanding of vulnerabilities at their disclosure. Disclosure can be a chaotic stage in the vulnerability life cycle, and some information existing in human readable form (such as the vulnerability free-form English description) may not exist in a machine readable format yet.

In Chapter 4 we showed that the affected software or hardware of a vulnerability could be determined at disclosure before the publication of CPE URI metadata from the text description of the vulnerability by using machine learning and information retrieval techniques such as TF-IDF. The results of the analysis process are presented as a short list of weighted and ordered keywords. Our evaluation protocol showed that keywords related to the name are present in the top three positions of the list 86 % of the time. There was no prior state of the art for this problem despite its practical relevance for the practitioners security community. Since the publication of this work other researchers built on top of this contribution [221], demonstrating the interest of the community for this problem.

In Chapter 5, we showed that we are able to automatically predict the CVSS vector of a new vulnerability at disclosure using linear regression, allowing for a complete and explicable severity analysis of the vulnerability. We achieve a prediction accuracy up to 96 % for individual CVSS fields and our CVSS score prediction error exhibits false negatives below 0.2 for 50 % of vulnerabilities and below 3.5 for 99 % of vulnerabilities. As for our previous contribution there was no prior state of the art for the problem of CVSS vector prediction despite the practical relevance of the problem.

A paramount requirement for both works is to keep the results *explicable*. This requires careful choice of featurization, dimension reduction, and training algorithms for both prediction pipelines, in order to keep an explicable relationship between the input and the output. We found that techniques such as bag-of-words, whitelist-based and conditional-

entropy-based filtering, TF-IDF and linear regression are helpful building blocks for designing explicable machine learning pipelines for security purposes.

Taken together these contributions create a state of the art for automated vulnerability analysis at disclosure.

7.1.3 Automated Vulnerability Risk Analysis at Disclosure

In our contribution detailed in Chapter 6, we go one step further to automate the risk analysis of newly disclosed vulnerabilities, in the context of a specific information system to protect. Instead of tackling the complex problem of charting the software used across an entire information system, we make use of the conscious and unconscious knowledge gathered by its security team by interactively training a risk analysis prediction system using active learning. The prediction system then monitors all newly disclosed vulnerabilities in real time, and uses the knowledge base it has acquired from the security expert to assert an appropriate alert level for each new vulnerability. This alert level is human-centered and made of three levels: a vulnerability can require an urgent response, a non-urgent response during business hours, or no response at all.

Our experimental protocol, made possible by the participation of actual security experts protecting real information systems, highlighted some practical difficulties in validating this design, such as how expensive it is to train active learning systems with time-constrained human experts. In this context, iterating over many hypotheses is challenging. However it also shows that keeping prediction systems explicable is a cornerstone of creating a trust relationship between automated prediction systems and security teams and that explicability is a decisive factor in the iterative improvement process of security systems. This contribution is an important step towards automating vulnerability risk analysis at disclosure.

7.1.4 Reacting to N-Day Vulnerabilities in Information Systems

Our contributions brought us closer to our ultimate goal of making security constant over time for information systems. In Chapter 3 we outlined an end-to-end strategy to automate defense against n-day vulnerabilities, and in this thesis work we solved multiple open problems implied by this strategy.

Before our contributions, to our knowledge there was no prior state of the art on how to react to new vulnerabilities in the seconds following their disclosure and this thesis

work bridges this gap. Our contributions detailed in Chapters 4 and 5 open the door to automatically gathering and understanding the properties of a vulnerability at disclosure, by proposing explicable automated methods while previous works rely on slow, manual analysis.

In Chapter 6 we proposed a preliminary contribution aiming to let a security expert train a defense system during working hours to automate vulnerability monitoring, removing the need for around-the-clock manual vulnerability management by security experts.

Our contributions already have had some impact:

— Our contribution described in Chapter 4 was published in the *2020 IEEE/IFIP Network Operations and Management Symposium* (NOMS 2020) [64] and we open-sourced all the related code and data [69]. Less than a year later other researchers (Wåreus et al [221]) already built upon our work, taking the problem we first identified and pushing it in a different direction.

— Our contribution described in Chapter 5 was published at the *15th International Conference on Availability, Reliability and Security* (ARES 2020) [65] and we open-sourced all the code and data for this contribution as well [68]. It was notably presented to the EPSS *Special Interest Group* (SIG), and was met with great interest. This contribution led us to participate regularly in the CVSS and EPSS SIGs and we hope to continue this participation in the future.

— We developed the open-source library libcvss [114], which aims to become the reference library when parsing and manipulating CVSS vectors using the Rust language.

In the next section we outline the perspectives opened up by this thesis.

7.2 Perspectives

In this section we detail the perspectives opened by this thesis, in the short, medium, and long term.

7.2.1 Short-Term

Our contribution related to vulnerability risk analysis, detailed in Chapter 6, is promising yet incomplete. More efforts are required in order to get stronger results, as well as maintaining and expanding our relationship with security experts. Increasing the accuracy

of our risk prediction system could have important practical implications for real-world application of our work.

During our discussions, the security experts we partnered with confirmed their interest in such a vulnerability monitoring system but highlighted how little time they had to participate in a purely scientific experiment with no immediate value for their day-to-day operations. Therefore a natural next step would be to build an actual vulnerability monitoring system that experts could use in a real-world context for an extended period of time, such as a year. On the one hand this would give them a useful tool (even if imperfect) while allowing us to collect more evaluation data, letting us iterate over more hypotheses.

Likewise, our contribution related to determining the affected component of a vulnerability, detailed in Chapter 4, could be taken further. In particular, the affected software is currently provided as a list of ordered keywords: it would be interesting to study the feasibility of providing it as an actual CPE URI containing the complete software name and vendor. Wåreus et al [221] built upon our work and showed that this is possible when using non-explicable machine learning techniques. It would be valuable to create a complete CPE URI while still retaining the explicability properties of our work.

A piece of vulnerability metadata we have not tried to predict is the CWE entry associated with the vulnerability, which describes the type of weakness exploited by the vulnerability. It would be worthwhile to apply the same techniques we used in our contributions from Chapters 4 and 5 to reach this goal.

This thesis was particularly multidisciplinary, highlighting that what is considered the state of the art in one community is sometimes considered obsolete in another. For instance, the TF-IDF algorithm is a standard algorithm in the Natural Language Processing (NLP) community, and we make use of it in Chapter 4. However it is considered obsolete in the Information Retrieval (IR) community, where the Okapi BM25 algorithm is considered superior. It would be interesting to revisit this contribution and see the effect of using Okapi BM25 instead of TF-IDF.

The tools and methods we designed during this thesis could be generalized to other pieces of data apart from vulnerability text descriptions. The activity of analyzing public data to extract valuable security information is called *Open-Source Intelligence* (OSINT). The OSINT community is very active, and it would be interesting to see how our contributions and in particular our keyword extraction pipeline (described in Chapter 4) could be applied to other data sources such as social networks, blogs, news sites, GitHub commits,

and many more.

7.2.2 Mid-Term

Deep learning allows very accurate machine learning decisions, but at the cost of making the decision process a non-explicable black box. Deep learning explicability is a very active research topic, with a state of the art improving every year. During this thesis we made the conscious choice to consider the state of the art in deep learning explicability as inadequate for security decisions. This may change in the future if the numerous works in progress on this topic eventually lead to a breakthrough. Such a result would create opportunities for our work to be superseded by other approaches.

Charting the software used across an entire information system is a challenging unsolved problem. There are widely used interpreted languages such as Python and Javascript, or semi-interpreted such as Java, that contradict the simplistic notion of software being only made of executable binaries. Static, dynamic, and more recently downloaded-on-the-fly dependencies all make it challenging to determine if a software component is present or absent in an information system. Complex and reprogrammable firmware blur the line between software and hardware.

Likewise, software versioning, both for charting software in an information system and cataloging the vulnerabilities present inside it, is another problem more complex than it seems at first sight. A software version is essentially a social convention which sometimes gets fuzzy. There are widely-used Javascript libraries for which the same version on Github and NPM differs. Debian maintainers are known to apply security patches to their packages if the original software author did not do it before, in which case they will create their own version number for the customized version. In a vulnerability free-form text description, it is difficult for machines and humans alike to discriminate between the last vulnerable and the first non-vulnerable version of the affected software.

A possible approach for solving both problems is to consider software not as data but as behavior. In that regard, techniques from intrusion detection (which separates legitimate from malicious behavior) and fuzzing (which attempts to trigger faulty behavior in a component by interacting with it) can be adapted to create a *chartist* system that recognizes the presence or absence of a software or hardware component, or even a specific version of a component, by sending an input to the information system and monitoring the output received in return. Just like with IDSs, some software or hardware might be detected only from a privileged observer on the same host while others could be detected

directly from the network. Unlike IDSs, there are no fundamental challenges in evaluating such an approach. There have been proposals in that direction [174], however this approach implies interacting with production software and hardware on a large scale, and an open research problem is how to guarantee the absence of side-effects such as crashes or data corruption during exploration.

7.2.3 Long-Term

Anomaly-based Intrusion Detection Systems (AIDSs) could one day become important tools to defend against n-day vulnerabilities. In Chapter 3 we showed that it was conceivable to create an Intrusion Prevention System (IPS) that would dynamically react to dangerous vulnerability disclosures. However, even if building such an IPS is practical, evaluating that it works properly is not. AIDS research is a twenty-five years-old domain, and yet a satisfactory evaluation protocol for them is still out of reach. The ideal dataset to this end would include realistic yet labeled attack and legitimate traffic, and would be correlated with realistic vulnerability disclosure events to measure their impact on recorded attacks. In practice, realistic attack traffic is very difficult to craft as the genuine attacker state of the art is unknown, while legitimate traffic is difficult to study and share because of privacy and confidentiality problems. The current state of the art in AIDS is divided between old, obsolete evaluation datasets and modern but unlabeled datasets (that are sometimes rife with privacy violations). A breakthrough in IDS evaluation would create many opportunities for better cybersecurity defense.

Assuming the component affected by a new vulnerability has correctly been identified, one could envision to automatically rediscover vulnerability exploitation vectors using a *fuzzer*: a class of test frameworks that use randomness to find bugs and vulnerabilities in software. This fuzzer could be dynamically guided by the preliminary information extracted from the vulnerability analysis, such as the affected software, vulnerable and non-vulnerable versions, the type of weakness exploited, etc. Once the vulnerability has been formally identified in the software, it might be possible to generate automatically a signature for it, either to detect the presence of a vulnerable binary in the information system, or for an IDS to detect an exploitation attempt.

Our work and its implications could be used in other domains, such as offensive security: could automated vulnerability analysis at disclosure facilitate or even automate exploit creation for n-day vulnerabilities? A state-sponsored offensive team could gain a strategic advantage from getting access to vulnerability exploits several hours, days or

weeks before the general public.

Another domain where defensive and offensive security meet is the concept of automated counter-attacks. While still within the realm of science-fiction for now, is it conceivable for an information system to react to an intrusion attempt by automatically launching a counter-attack aiming to stop and identify the authors of the intrusion? Such a possibility would raise fascinating ethical and scientific questions.

Most organizations face much stronger online threats than local ones. Why do our computers and smartphones have to be kept as secure as fortresses while many of us are content with very simple physical locks to protect our homes and offices? One answer is that the present Internet resembles an international war zone, in which anyone in the world can attack anyone else instantly, requiring everybody to adopt military-grade defense tactics and equipment. Is this sustainable? Should information systems become ever more secure or should the world become safer? It might be inevitable that online security has to be delegated to sovereign organizations providing protection to civilians. Currently this role is assumed by private sector organizations, including the major technology companies such as Google, Apple, Facebook, Amazon and Microsoft, and key network-level actors such as Cloudflare. Should this responsibility be handled by nation states instead? What would be the ethical and privacy implications of such a change? What are the technical implications of using intrusion detection techniques to protect a nation state comprised of many heterogeneous systems compared to protecting a single large cloud provider? A second direction would be a worldwide justice system enabling the prosecution of the myriads of criminal threat actors that currently operate unbothered in remote parts of the world. This would require making progress in attack attribution and more importantly, proof of this attribution, which are both very hard, largely unsolved problems. Could there be a worldwide initiative aiming to leave no online criminal behavior unpunished? Could we allow any organization to file a complaint and have the attacker unmasked and prosecuted? History has shown that prosperity and safety are not achieved by arming everyone for a perpetual war, but instead by creating the conditions for durable peace.

BIBLIOGRAPHY

[1] *A Complete Guide to the Common Vulnerability Scoring System: Version 2.0*, 2020-08-04, URL: https://www.first.org/cvss/v2/guide.

[2] *A list of portmortems*, 2020-18-11, URL: https://github.com/danluu/post-mortems.

[3] *Alex Stamos, CISO of Yahoo, on Hacker News*, 2020-03-08, URL: https://news.ycombinator.com/item?id=8418809.

[4] *Alexa - Keyword Research, Competitive Analysis, and Website Ranking*, 2020-31-07, URL: https://www.alexa.com/.

[5] *AlienVault is Now AT&T Cybersecurity*, 2020-12-11, URL: https://cybersecurity.att.com/.

[6] Mouhammd Alkasassbeh et al., « Detecting distributed denial of service attacks using data mining techniques », *in*: *International Journal of Advanced Computer Science and Applications* 7.1 (2016), pp. 436–445.

[7] Luca Allodi, « Economic Factors of Vulnerability Trade and Exploitation », *in*: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, 1483–1499, ISBN: 9781450349468, DOI: 10.1145/3133956.3133960, URL: https://doi.org/10.1145/3133956.3133960.

[8] Luca Allodi and Fabio Massacci, « 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*, BADGERS '12, Raleigh, North Carolina, USA: Association for Computing Machinery, 2012, 17–24, ISBN: 9781450316613, DOI: 10.1145/2382416.2382427, URL: https://doi.org/10.1145/2382416.2382427.

[9] Luca Allodi and Fabio Massacci, « Comparing Vulnerability Severity and Exploits Using Case-Control Studies », *in*: *ACM Trans. Inf. Syst. Secur.* 17.*1* (Aug. 2014), ISSN: 1094-9224, DOI: 10.1145/2630069, URL: https://doi.org/10.1145/2630069.

[10] Philip M Anderson and Cherie Ann Sherman, « Applying the Fermi estimation technique to business problems », *in*: *The Journal of Applied Business and Economics* 10.*5* (2010), p. 33.

[11] *Announcing Project Zero*, 2020-31-07, URL: https://security.googleblog.com/2014/07/announcing-project-zero.html.

[12] *Apache Log4j 2*, 2020-18-11, URL: https://logging.apache.org/log4j/2.x/index.html.

[13] *AWS WAF - Web Application Firewall*, 2020-08-04, URL: https://aws.amazon.com/waf/.

[14] Sean Barnum, « Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX) », *in*: (2014).

[15] Adam Barth et al., « A Learning-Based Approach to Reactive Security », *in*: *CoRR* abs/0912.1155 (2009), arXiv: 0912.1155, URL: http://arxiv.org/abs/0912.1155.

[16] Anaël Beaugnon, « Expert-in-the-Loop Supervised Learning for Computer Security Detection Systems », Theses, PSL Research University, June 2018, URL: https://hal.archives-ouvertes.fr/tel-01888971.

[17] Anaël Beaugnon, Pierre Chifflier, and Francis Bach, « Ilab: An interactive labelling strategy for intrusion detection », *in*: *International Symposium on Research in Attacks, Intrusions, and Defenses*, Springer, 2017, pp. 120–140.

[18] Betsy Beyer et al., *Site Reliability Engineering: How Google Runs Production Systems*, " O'Reilly Media, Inc.", 2016.

[19] S. Bhatt, P. K. Manadhata, and L. Zomlot, « The Operational Role of Security Information and Event Management Systems », *in*: *IEEE Security Privacy* 12.*5* (2014), pp. 35–41.

[20] Battista Biggio et al., « Security evaluation of support vector machines in adversarial environments », *in*: *Support Vector Machines Applications*, Springer, 2014, pp. 105–153.

[21] Leyla Bilge and Tudor Dumitraş, « Before We Knew It: An Empirical Study of Zero-day Attacks in the Real World », *in*: *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS' 12)*, Raleigh, North Carolina, USA: ACM, 2012, pp. 833–844, ISBN: 978-1-4503-1651-4, DOI: 10.1145/2382196.2382284.

[22] *Bitcoin Core version 0.9.1 released*, 2020-29-07, URL: https://bitcoin.org/en/release/v0.9.1.

[23] Alan Bivens et al., « Network-based intrusion detection using neural networks », *in*: *Intelligent Engineering Systems through Artificial Neural Networks* 12.*1* (2002), pp. 579–584.

[24] Mehran Bozorgi et al., « Beyond Heuristics: Learning to Classify Vulnerabilities and Predict Exploits », *in*: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, Washington, DC, USA: Association for Computing Machinery, 2010, 105–114, ISBN: 9781450300551, DOI: 10.1145/1835804.1835821, URL: https://doi.org/10.1145/1835804.1835821.

[25] Ferdinand Brasser et al., « Software Grand Exposure: SGX Cache Attacks Are Practical », *in*: *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC: USENIX Association, Aug. 2017, URL: https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser.

[26] Robert A. Bridges et al., « A Survey of Intrusion Detection Systems Leveraging Host Data », *in*: *ACM Comput. Surv.* 52.*6* (Nov. 2019), ISSN: 0360-0300, DOI: 10.1145/3344382, URL: https://doi.org/10.1145/3344382.

[27] A. L. Buczak and E. Guven, « A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection », *in*: *IEEE Communications Surveys Tutorials* 18.*2* (2016), pp. 1153–1176.

[28] Jo Van Bulck et al., « Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution », *in*: *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, Aug. 2018, 991–1008, ISBN: 978-1-939133-04-5, URL: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck.

[29] *Buying Into the Bias: Why Vulnerability Statistics Suck*, 2020-16-06, URL: https://txt.731my.com/tmp/blackHat/US-13-Martin-Buying-Into-The-Bias-Why-Vulnerability-Statistics-Suck-WP.pdf.

[30] Dario Caldara and Matteo Iacoviello, « Measuring geopolitical risk », *in*: *FRB International Finance Discussion Paper 1222* (2018).

[31] *Canvas*, 2020-12-11, URL: https://immunityinc.com/products/canvas/.

[32] M Emre Celebi and Kemal Aydin, *Unsupervised learning algorithms*, Springer, 2016.

[33] Yin-Wen Chang et al., « Training and Testing Low-degree Polynomial Data Mappings via Linear SVM », *in*: *Journal of Machine Learning Research* 11.*48* (2010), pp. 1471–1490, URL: http://jmlr.org/papers/v11/chang10a.html.

[34] Wayne Chappelle et al., *Sources of Occupational Stress and Prevalence of Burnout and Clinical Distress Among U.S. Air Force Cyber Warfare Operators*, tech. rep., SCHOOL OF AEROSPACE MEDICINE WRIGHT PATTERSON AFB OH, 2013.

[35] S. A. de Chaves, C. B. Westphall, and F. R. Lamin, « SLA Perspective in Security Management for Cloud Computing », *in*: *2010 Sixth International Conference on Networking and Services*, 2010, pp. 212–217, DOI: 10.1109/ICNS.2010.36.

[36] Ilaria Chillotti et al., « Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE », *in*: *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2017, pp. 377–408.

[37] Sandy Clark et al., « Familiarity Breeds Contempt: The Honeymoon Effect and the Role of Legacy Code in Zero-day Vulnerabilities », *in*: *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC '10)*, Austin, Texas, USA: ACM, 2010, pp. 251–260, ISBN: 978-1-4503-0133-6, DOI: 10.1145/1920261.1920299.

[38] *Cloudflare - Inside Shellshock: How hackers are using it to exploit systems*, 2020-08-04, URL: https://blog.cloudflare.com/inside-shellshock/.

[39] *Cloudflare - Stopping SharePoint's CVE-2019-0604*, 2020-08-04, URL: https://blog.cloudflare.com/stopping-cve-2019-0604/.

[40] *Cloudflare Web Application Firewall*, 2020-08-04, URL: https://www.cloudflare.com/waf/.

[41] *Common Vulnerabilities and Exposures (CVE)*, 2020-08-04, URL: https://cve.mitre.org/.

[42] *Common Vulnerability Scoring System*, 2020-08-04, URL: https://www.first.org/cvss/.

[43] *Common Vulnerability Scoring System v3.0: Specification Document*, 2020-08-04, URL: https://www.first.org/cvss/v3.0/specification-document.

[44] *Common Vulnerability Scoring System v3.1: Specification Document*, 2020-08-04, URL: https://www.first.org/cvss/v3.1/specification-document.

[45] *Core Infrastructure Initiative*, 2020-31-07, URL: https://www.coreinfrastructure.org/.

[46] Corinna Cortes and Vladimir Vapnik, « Support-vector networks », *in*: *Machine learning* 20.*3* (1995), pp. 273–297.

[47] Russ Cox, « Surviving Software Dependencies », *in*: *Queue* 17.*2* (Apr. 2019), 24–47, ISSN: 1542-7730, DOI: 10.1145/3329781.3344149, URL: https://doi.org/10.1145/3329781.3344149.

[48] *CVE - Request CVE IDs - Participating CNAs*, 2020-28-07, URL: https://cve.mitre.org/cve/request_id.html.

[49] *CVE-2014-6271: remote code execution through bash*, 2020-03-08, URL: https://seclists.org/oss-sec/2014/q3/649.

[50] *CVE-2017-13872 - Video Demonstration*, 2020-27-07, URL: https://twitter.com/patrickwardle/status/935608904377077761.

[51] *CVE and NVD Relationship*, 2020-08-04, URL: https://cve.mitre.org/about/cve_and_nvd_relationship.html.

[52] *CWE - Common Weakness Enumeration*, 2020-08-04, URL: https://cwe.mitre.org/.

[53] *Cybersecurity Solutions, Services and Training | Proofpoint*, 2020-12-11, URL: https://www.proofpoint.com/us.

[54] *D2 Elliot Web Exploitation Framework*, 2020-12-11, URL: https://www.d2sec.com/elliot.html.

[55] *DARPA Intrusion Detection Data Sets*, 2020-12-11, URL: https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset.

[56] *David A. Wheeler - Shellshock*, 2020-03-08, URL: https://dwheeler.com/essays/shellshock.html.

[57] *Debian – Security Information*, 2020-12-11, URL: https://www.debian.org/security/.

[58] B. Delamore and R. K. L. Ko, « A Global, Empirical Analysis of the Shellshock Vulnerability in Web Applications », *in*: *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015, pp. 1129–1135.

[59] *Discover eIDAS*, 2020-23-07, URL: https://ec.europa.eu/digital-single-market/en/discover-eidas.

[60] Thomas Duebendorfer and Stefan Frei, « Web Browser Security Update Effectiveness », *in*: *Critical Information Infrastructures Security*, ed. by Erich Rome and Robin Bloomfield, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 124–137, ISBN: 978-3-642-14379-3.

[61] Zakir Durumeric et al., « The Matter of Heartbleed », *in*: *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, Vancouver, BC, Canada: Association for Computing Machinery, 2014, 475–488, ISBN: 9781450332132, DOI: 10.1145/2663716.2663755, URL: https://doi.org/10.1145/2663716.2663755.

[62] Josiah Dykstra and Celeste Lyn Paul, « Cyber Operations Stress Survey (COSS): Studying fatigue, frustration, and cognitive workload in cybersecurity operations », *in*: *11th USENIX Workshop on Cyber Security Experimentation and Test (CSET 18)*, Baltimore, MD: USENIX Association, Aug. 2018, URL: https://www.usenix.org/conference/cset18/presentation/dykstra.

[63] *Edward Snowden about the Shadow Brokers - Twitter*, 2020-04-08, URL: https://twitter.com/Snowden/status/765513662597623808.

[64] C. Elbaz, L. Rilling, and C. Morin, « Automated Keyword Extraction from "One-day" Vulnerabilities at Disclosure », *in*: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9.

[65] Clément Elbaz, Louis Rilling, and Christine Morin, « Fighting N-Day Vulnerabilities with Automated CVSS Vector Prediction at Disclosure », *in*: *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ARES '20, Virtual Event, Ireland: Association for Computing Machinery, 2020, ISBN:

9781450388337, DOI: 10.1145/3407023.3407038, URL: https://doi.org/10.1145/3407023.3407038.

[66] Clément Elbaz, Louis Rilling, and Christine Morin, « Mesurer et prévenir l'évolution de la menace dans un cloud d'infrastructure », *in*: *ComPas'18 - Conférence d'informatique en Parallélisme, Architecture et Système*, Toulouse, France, July 2018, pp. 1–7, URL: https://hal.inria.fr/hal-01816674.

[67] *Exploit Database - Exploits for Penetration Testers, Researchers, and Ethical Hackers*, 2020-12-11, URL: https://www.exploit-db.com/.

[68] *Firres (ARES2020)*, 2020-25-11, URL: https://gitlab.inria.fr/celbaz/firres_ares.

[69] *Firres (NOMS2020)*, 2020-25-11, URL: https://gitlab.inria.fr/celbaz/firres_noms.

[70] *First Shellshock botnet attacks Akamai, US DoD networks*, 2020-03-08, URL: https://www.itnews.com.au/news/first-shellshock-botnet-attacks-akamai-us-dod-networks-396197.

[71] *Fortinet | Enterprise Security Without Compromise*, 2020-12-11, URL: https://www.fortinet.com/.

[72] David A Freedman, *Statistical models: theory and practice*, cambridge university press, 2009.

[73] Stefan Frei et al., « Large-scale Vulnerability Analysis », *in*: *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense (LSAD '06)*, Pisa, Italy: ACM, 2006, pp. 131–138, ISBN: 1-59593-571-1, DOI: 10.1145/1162666.1162671.

[74] Stefan Frei et al., « Modeling the Security Ecosystem - The Dynamics of (In)Security », *in*: *Economics of Information Security and Privacy*, ed. by Tyler Moore, David Pym, and Christos Ioannidis, Boston, MA: Springer US, 2010, pp. 79–106, ISBN: 978-1-4419-6967-5.

[75] Pedro Garcia-Teodoro et al., « Anomaly-based network intrusion detection: Techniques, systems and challenges », *in*: *computers & security* 28.*1-2* (2009), pp. 18–28.

[76] Craig Gentry, « Fully Homomorphic Encryption Using Ideal Lattices », *in*: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, Bethesda, MD, USA: Association for Computing Machinery, 2009, 169–178, ISBN: 9781605585062, DOI: 10.1145/1536414.1536440, URL: https://doi.org/10.1145/1536414.1536440.

[77] S Ghafur et al., « A retrospective impact analysis of the WannaCry cyberattack on the NHS », *in*: *NPJ digital medicine* 2.*1* (2019), pp. 1–7.

[78] A. Gharib et al., « An Evaluation Framework for Intrusion Detection Dataset », *in*: *2016 International Conference on Information Science and Security (ICISS)*, 2016, pp. 1–6.

[79] Leonid Glanz et al., « A Vulnerability's Lifetime: Enhancing Version Information in CVE Databases », *in*: *Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business (i-KNOW '15)*, Graz, Austria: ACM, 2015, ISBN: 978-1-4503-3721-2, DOI: 10.1145/2809563.2809612.

[80] J Gomez and C Konschak, « Cyber-security in healthcare-understanding the new world threat », *in*: *Divurgent* (2015), pp. 1–12.

[81] *Google Cloud Armor*, 2020-08-04, URL: https://cloud.google.com/armor/.

[82] Johannes Götzfried et al., « Cache Attacks on Intel SGX », *in*: *Proceedings of the 10th European Workshop on Systems Security*, EuroSec'17, Belgrade, Serbia: Association for Computing Machinery, 2017, ISBN: 9781450349352, DOI: 10.1145/3065913.3065915, URL: https://doi.org/10.1145/3065913.3065915.

[83] *GreyNoise Intelligence*, 2020-12-11, URL: https://greynoise.io/.

[84] *Hackers Are Already Using the Shellshock Bug to Launch Botnet Attacks*, 2020-03-08, URL: https://www.wired.com/2014/09/hackers-already-using-shellshock-bug-create-botnets-ddos-attacks/.

[85] Zellig S Harris, « Distributional structure », *in*: *Word* 10.*2-3* (1954), pp. 146–162.

[86] *Heartbleed Bug*, 2020-31-07, URL: https://heartbleed.com/.

[87] *Heartbleed vulnerability may have been exploited months before patch [Updated]*, 2020-31-07, URL: https://arstechnica.com/information-technology/2014/04/heartbleed-vulnerability-may-have-been-exploited-months-before-patch/.

[88] Hanan Hindy et al., « A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets », *in*: *CoRR* abs/1806.03517 (2018), arXiv: 1806.03517, URL: http://arxiv.org/abs/1806.03517.

[89] *Hints suggest an insider helped the NSA "Equation Group" hacking tools leak*, 2020-04-08, URL: https://arstechnica.com/information-technology/2016/08/hints-suggest-an-insider-helped-the-nsa-equation-group-hacking-tools-leak/.

[90] *Hogzilla IDS*, 2020-12-11, URL: http://ids-hogzilla.org/.

[91] Paul W. Holland and Roy E. Welsch, « Robust regression using iteratively reweighted least-squares », *in*: *Communications in Statistics - Theory and Methods* 6.*9* (1977), pp. 813–827, DOI: 10.1080/03610927708827533, eprint: https://doi.org/10.1080/03610927708827533, URL: https://doi.org/10.1080/03610927708827533.

[92] Hannes Holm and Khalid Khan Afridi, « An expert-based investigation of the Common Vulnerability Scoring System », *in*: *Computers & Security* 53 (2015), pp. 18 – 30, ISSN: 0167-4048, DOI: https://doi.org/10.1016/j.cose.2015.04.012, URL: http://www.sciencedirect.com/science/article/pii/S0167404815000620.

[93] Siv Hilde Houmb, Virginia N.L. Franqueira, and Erlend A. Engum, « Quantifying security risk level from CVSS estimates of frequency and impact », *in*: *Journal of Systems and Software* 83.*9* (2010), Software Dependability, pp. 1622 –1634, ISSN: 0164-1212, DOI: https://doi.org/10.1016/j.jss.2009.08.023, URL: http://www.sciencedirect.com/science/article/pii/S0164121209002155.

[94] *How The Internet's Worst Nightmare Could Have Been Avoided*, 2020-31-07, URL: https://www.huffpost.com/entry/heartbleed-bug_n_5120457.

[95] *How to Prevent the next Heartbleed*, 2020-31-07, URL: https://dwheeler.com/essays/heartbleed.html.

[96] *H.R.2481 - PATCH Act of 2017*, 2020-04-08, URL: https://www.congress.gov/bill/115th-congress/house-bill/2481.

[97] Ghaith Husari et al., « TTPDrill: Automatic and Accurate Extraction of Threat Actions from Unstructured Text of CTI Sources », *in*: *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, Orlando, FL, USA: Association for Computing Machinery, 2017, 103–115, ISBN: 9781450353458,

DOI: 10.1145/3134600.3134646, URL: https://doi.org/10.1145/3134600.3134646.

[98] Piotr Indyk and Rajeev Motwani, « Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality », *in*: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, Dallas, Texas, USA: Association for Computing Machinery, 1998, 604–613, ISBN: 0897919629, DOI: 10.1145/276698.276876, URL: https://doi.org/10.1145/276698.276876.

[99] *Intel® Software Guard Extensions*, 2020-23-07, URL: https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.

[100] *Interpretable Machine Learning*, 2021-25-05, URL: https://christophm.github.io/interpretable-ml-book/.

[101] *Israel Retaliates To A Cyber-Attack With Immediate Physical Action In A World First*, 2020-25-11, URL: https://www.forbes.com/sites/kateoflahertyuk/2019/05/06/israel-retaliates-to-a-cyber-attack-with-immediate-physical-action-in-a-world-first/.

[102] Jay Jacobs et al., « Exploit Prediction Scoring System (EPSS) », *in*: *Black Hat 2019*, 2019, URL: http://i.blackhat.com/USA-19/Thursday/us-19-Roytman-Predictive-Vulnerability-Scoring-System-wp.pdf.

[103] Gareth James et al., *An introduction to statistical learning*, vol. 112, Springer, 2013.

[104] Karen Spärck Jones, « A statistical interpretation of term specificity and its application in retrieval », *in*: *Journal of Documentation* 28 (1972), pp. 11–21.

[105] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore, « Reinforcement learning: A survey », *in*: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[106] Ho Tin Kam et al., « Random decision forest », *in*: *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, vol. 1416, Montreal, Canada, August, 1995, p. 278282.

[107] *KDD Cup 1999 Data*, 2020-12-11, URL: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[108] Atefeh Khazaei, Mohammad Ghasemzadeh, and Vali Derhami, « An automatic method for CVSS score prediction using vulnerabilities description », *in*: *Journal of Intelligent & Fuzzy Systems* 30.*1* (2016), pp. 89–96.

[109] P. Kocher et al., « Spectre Attacks: Exploiting Speculative Execution », *in*: *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19.

[110] Sailesh Kumar, « Survey of current network intrusion detection techniques », *in*: *Washington Univ. in St. Louis* (2007), pp. 1–18.

[111] R. Langner, « Stuxnet: Dissecting a Cyberwarfare Weapon », *in*: *IEEE Security Privacy* 9.*3* (2011), pp. 49–51, DOI: 10.1109/MSP.2011.67.

[112] Quoc Le and Tomas Mikolov, « Distributed representations of sentences and documents », *in*: *International conference on machine learning*, 2014, pp. 1188–1196.

[113] Kingsly Leung and Christopher Leckie, « Unsupervised anomaly detection in network intrusion detection using clusters », *in*: *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, 2005, pp. 333–342.

[114] *libcvss - Rust*, 2020-28-11, URL: https://docs.rs/libcvss.

[115] Moritz Lipp et al., *Meltdown*, 2018, arXiv: 1801.01207 [cs.CR].

[116] Yang Liu et al., « Cloudy with a Chance of Breach: Forecasting Cyber Security Incidents », *in*: *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C.: USENIX Association, Aug. 2015, pp. 1009–1024, ISBN: 978-1-939133-11-3, URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/liu.

[117] M. H. Bhuyan and D. K. Bhattacharyya and J. K. Kalita, « Network Anomaly Detection: Methods, Systems and Tools », *in*: *IEEE Communications Surveys Tutorials* 16.*1* (2014), pp. 303–336, ISSN: 1553-877X, DOI: 10.1109/SURV.2013.052213.00046.

[118] Aleksander Madry et al., « Towards deep learning models resistant to adversarial attacks », *in*: *arXiv preprint arXiv:1706.06083* (2017).

[119] MattMiller, « Trends, challenges, and strategic shifts in the software vulnerability mitigation landscape », *in*: BlueHat IL, 2019.

[120] V. Mavroeidis and S. Bromander, « Cyber Threat Intelligence Model: An Evaluation of Taxonomies, Sharing Standards, and Ontologies within Cyber Threat Intelligence », *in*: *2017 European Intelligence and Security Informatics Conference (EISIC)*, 2017, pp. 91–98.

[121] *Merck Cyberattack's 1.3 Billion dollars Question: Was It an Act of War?*, 2020-04-08, URL: https://www.bloomberg.com/news/features/2019-12-03/merck-cyberattack-s-1-3-billion-question-was-it-an-act-of-war.

[122] *Metasploit: Penetration Testing Software*, 2020-12-11, URL: https://www.rapid7.com/products/metasploit/.

[123] *Microsoft Security Bulletin MS17-010 - Critical*, 2020-03-08, URL: https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010.

[124] Yisroel Mirsky et al., « Kitsune: an ensemble of autoencoders for online network intrusion detection », *in*: *arXiv preprint arXiv:1802.09089* (2018).

[125] S. Mittal et al., « 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)*, 2016, pp. 860–867.

[126] *Myths About Samba*, 2020-04-08, URL: https://www.samba.org/samba/docs/myths_about_samba.html.

[127] A. Nappa et al., « The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching », *in*: *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 692–708.

[128] *National Vulnerability Database*, 2020-08-04, URL: https://nvd.nist.gov/.

[129] Kartik Nayak et al., « Some Vulnerabilities Are Different Than Others », *in*: *Research in Attacks, Intrusions and Defenses*, ed. by Angelos Stavrou, Herbert Bos, and Georgios Portokalidis, Cham: Springer International Publishing, 2014, pp. 426–446, ISBN: 978-3-319-11379-1.

[130] *Next Generation Firewall - Palo Alto Networks*, 2020-28-11, URL: https://www.paloaltonetworks.com/network-security/next-generation-firewall.

[131] *NHS cyber-attack: GPs and hospitals hit by ransomware*, 2020-03-08, URL: https://www.bbc.com/news/health-39899646.

[132] *NIST IR 7695 — Common Platform Enumeration: Naming Specification Version 2.3*, 2020-12-11, URL: http://csrc.nist.gov/publications/nistir/ir7695/NISTIR-7695-CPE-Naming.pdf.

[133] *NotPetya a coûté cher à Saint-Gobain*, 2020-04-08, URL: https://www.zdnet.fr/actualites/notpetya-a-coute-cher-a-saint-gobain-39855594.htm.

[134] *NotPetya Holds Up a Stop Sign for FedEx*, 2020-04-08, URL: https://www.uscybersecurity.net/csmag/notpetya-holds-up-a-stop-sign-for-fedex/.

[135] *NotPetya Technical Analysis – A Triple Threat: File Encryption, MFT Encryption, Credential Theft*, 2020-04-08, URL: https://www.crowdstrike.com/blog/petrwrap-ransomware-technical-analysis-triple-threat-file-encryption-mft-encryption-credential-theft/.

[136] *NSA-leaking Shadow Brokers just dumped its most damaging release yet*, 2020-04-08, URL: https://arstechnica.com/information-technology/2017/04/nsa-leaking-shadow-brokers-just-dumped-its-most-damaging-release-yet/.

[137] *NSA officials worried about the day its potent hacking tool would get loose. Then it did.* 2020-03-08, URL: https://www.washingtonpost.com/business/technology/nsa-officials-worried-about-the-day-its-potent-hacking-tool-would-get-loose-then-it-did/2017/05/16/50670b16-3978-11e7-a058-ddbb23c75d82_story.html.

[138] *NSL-KDD dataset*, 2020-12-11, URL: http://www.unb.ca/cic/datasets/nsl.html.

[139] *NVD - CPE*, 2020-08-04, URL: https://nvd.nist.gov/products/cpe.

[140] *NVD - CVE-2014-0160*, 2020-27-07, URL: https://nvd.nist.gov/vuln/detail/CVE-2014-0160.

[141] *NVD - CVE-2014-6271*, 2020-27-07, URL: https://nvd.nist.gov/vuln/detail/CVE-2014-6271.

[142] *NVD - CVE-2017-0144*, 2020-03-08, URL: https://nvd.nist.gov/vuln/detail/CVE-2017-0144.

[143] *NVD - CVE-2017-13872*, 2020-27-07, URL: https://nvd.nist.gov/vuln/detail/CVE-2017-13872.

[144] *NVD - CVMAP*, 2020-27-07, URL: https://nvd.nist.gov/vuln/cvmap.

[145] Obi Ogbanufe and Janine Spears, « Burnout in Cybersecurity Professionals », *in*: Munich, Germany: WISP 2019 - Workshop in Information Security and Privacy, 2019.

[146] *One Year After WannaCry, EternalBlue Exploit Is Bigger Than Ever*, 2020-04-08, URL: https://www.bleepingcomputer.com/news/security/one-year-after-wannacry-eternalblue-exploit-is-bigger-than-ever/.

[147] Kris Oosthoek and Christian Doerr, « Cyber Threat Intelligence: A Product Without a Process? », *in*: *International Journal of Intelligence and CounterIntelligence* 0.*0* (2020), pp. 1–16, DOI: 10.1080/08850607.2020.1780062, eprint: https://doi.org/10.1080/08850607.2020.1780062, URL: https://doi.org/10.1080/08850607.2020.1780062.

[148] *OpenCTI: Open Cyber Threat Intelligence Platform*, 2020-01-09, URL: https://github.com/OpenCTI-Platform/opencti.

[149] *OpenSSL - Cryptography and SSL/TLS Toolkit*, 2020-31-07, URL: https://www.openssl.org/.

[150] *OSVDB Shut Down Permanently - Securityweek.com*, 2020-28-11, URL: https://www.securityweek.com/osvdb-shut-down-permanently.

[151] Rain Ottis, « Analysis of the 2007 cyber attacks against estonia from the information warfare perspective », *in*: *Proceedings of the 7th European Conference on Information Warfare*, 2008, p. 163.

[152] *Patching is Hard*, 2020-16-06, URL: https://www.cs.columbia.edu/~smb/blog/2017-05/2017-05-12.html.

[153] *Payment Card Industry - Data Security Standard (PCI DSS) v3.2.1 May 2018*, 2020-23-07, URL: https://www.pcisecuritystandards.org/document_library.

[154] Dana Petcu and Ciprian Crăciun, « Towards a Security SLA-based Cloud Monitoring Service », *in*: *Proceedings of the 4th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,* INSTICC, SciTePress, 2014, pp. 598–603, ISBN: 978-989-758-019-2, DOI: 10.5220/0004957305980603.

[155] Gabriele Piccoli, *Information Systems for Managers: Texts and Cases*, 1st, Wiley Publishing, 2007, ISBN: 047008703X.

[156] *Project Zero: Policy and Disclosure: 2020 Edition*, 2020-27-07, URL: https://googleprojectzero.blogspot.com/2020/01/policy-and-disclosure-2020-edition.html.

[157] *qmail is a vector for CVE-2014-6271 (bash shellshock)*, 2020-03-08, URL: https://marc.info/?l=qmail&m=141183309314366&w=2.

[158] *Quarterly Threat Report: Q4 and 2018 Wrap-Up*, 2020-04-08, URL: https://www.rapid7.com/info/threat-report/2018-q4-threat-report/.

[159] *Ransomware attack at German hospital leads to death of patient*, 2020-29-10, URL: https://www.bleepingcomputer.com/news/security/ransomware-attack-at-german-hospital-leads-to-death-of-patient/.

[160] *Rapid7 2020 Threat Report: Exposing Common Attacker Trends*, 2020-04-08, URL: https://blog.rapid7.com/2020/03/03/rapid7-2020-threat-report-exposing-common-attacker-trends/.

[161] *Reinsurance to take minimal share of 8 billion dollars WannaCry economic loss: A.M. Best*, 2020-04-08, URL: https://www.reinsurancene.ws/reinsurance-take-minimal-share-8-billion-wannacry-economic-loss-m-best/.

[162] *Report: Devastating Heartbleed Flaw Was Used in Hospital Hack*, 2020-31-07, URL: https://time.com/3148773/report-devastating-heartbleed-flaw-was-used-in-hospital-hack/.

[163] *Report: Shellshock Attack Hits Yahoo*, 2020-03-08, URL: https://www.bankinfosecurity.com/report-shellshock-attack-hits-yahoo-a-7404.

[164] Markus Ring et al., « A survey of network-based intrusion detection data sets », *in*: *Computers & Security* 86 (2019), pp. 147–167.

[165] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al., « On data banks and privacy homomorphisms », *in*: *Foundations of secure computation* 4.*11* (1978), pp. 169–180.

[166] Stephen E Robertson et al., « Okapi at TREC-3 », *in*: *Nist Special Publication Sp* 109 (1995), p. 109.

[167] Armin Roeseler and Anneliese von Mayrhauser, « Managing the quality of computing services: A user-oriented approach using utility theory and service-level indices », *in*: *Journal of Systems and Software* 16.*3* (1991), pp. 185 –196, ISSN: 0164-1212, DOI: https://doi.org/10.1016/0164-1212(91)90013-V, URL: http://www.sciencedirect.com/science/article/pii/016412129190013V.

[168] Lior Rokach and Oded Z Maimon, *Data mining with decision trees: theory and applications*, vol. 69, World scientific, 2008.

[169] Ben Rothke, « Building a Security Operations Center (SOC) », *in*: RSA Conference 2012, 2012.

[170] Mikko Ruohonen, « Stakeholders of strategic information systems planning: theoretical concepts and empirical examples », *in*: *The Journal of Strategic Information Systems* 1.*1* (1991), pp. 15 –28, ISSN: 0963-8687, DOI: https://doi.org/10.1016/0963-8687(91)90004-3, URL: http://www.sciencedirect.com/science/article/pii/0963868791900043.

[171] Stuart Russel, Peter Norvig, et al., *Artificial intelligence: a modern approach*, Pearson Education Limited, 2013.

[172] Carl Sabottke, Octavian Suciu, and Tudor Dumitras, « Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits », *in*: *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C.: USENIX Association, Aug. 2015, pp. 1041–1056, ISBN: 978-1-939133-11-3, URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/sabottke.

[173] Luis Alberto Benthin Sanguino and Rafael Uetz, « Software Vulnerability Analysis Using CPE and CVE », *in*: *CoRR* abs/1705.05347 (2017), arXiv: 1705.05347.

[174] Nathan Schagen et al., « Towards Automated Vulnerability Scanning of Network Servers », *in*: *Proceedings of the 11th European Workshop on Systems Security*, EuroSec'18, Porto, Portugal: Association for Computing Machinery, 2018, ISBN: 9781450356527, DOI: 10.1145/3193111.3193116, URL: https://doi.org/10.1145/3193111.3193116.

[175] Bruce Schneier, *Beyond fear: Thinking sensibly about security in an uncertain world*, Springer Science & Business Media, 2006.

[176] Michael Schwarz, Samuel Weiser, and Daniel Gruss, « Practical enclave malware with Intel SGX », *in*: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2019, pp. 177–196.

[177] Michael Schwarz et al., « Malware guard extension: Using SGX to conceal cache attacks », *in*: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2017, pp. 3–24.

[178] *Security Content Automation Protocol*, 2020-08-04, URL: https://csrc.nist.gov/projects/security-content-automation-protocol.

[179] *Security-Enhanced Linux in Android*, 2020-12-11, URL: https://source.android.com/security/selinux.

[180] Ravi Sen, Joobin Choobineh, and Subodha Kumar, « Determinants of Software Vulnerability Disclosure Timing », *in*: *Production and Operations Management* 29.*11* (2020), pp. 2532–2552, DOI: https://doi.org/10.1111/poms.13120, eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/poms.13120, URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/poms.13120.

[181] Burr Settles, *Active learning literature survey*, tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2009.

[182] Burr Settles and Mark Craven, « An Analysis of Active Learning Strategies for Sequence Labeling Tasks », *in*: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, Honolulu, Hawaii: Association for Computational Linguistics, 2008, 1070–1079.

[183] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X. Liu, « A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles », *in*: *Proceedings of the 34th International Conference on Software Engineering*, Zurich, Switzerland: IEEE Press, 2012, pp. 771–781, ISBN: 978-1-4673-1067-3, URL: http://dl.acm.org/citation.cfm?id=2337223.2337314.

[184] *Shellshock OpenSSH restricted shell RCE/PE Proof of Concept*, 2020-03-08, URL: https://www.zdziarski.com/blog/?p=3905.

[185] Amit Singhal et al., « Modern information retrieval: A brief overview », *in*: *IEEE Data Eng. Bull.* 24.*4* (2001), pp. 35–43.

[186] *SMBv1 is not installed by default in Windows 10 version 1709, Windows Server version 1709 and later versions*, 2020-03-08, URL: https://docs.microsoft.com/en-us/windows-server/storage/file-server/troubleshoot/smbv1-not-installed-by-default-in-windows.

[187] Robin Sommer and Vern Paxson, « Outside the closed world: On using machine learning for network intrusion detection », *in*: *2010 IEEE symposium on security and privacy*, IEEE, 2010, pp. 305–316.

[188] Kyle Soska and Nicolas Christin, « Automatically Detecting Vulnerable Websites Before They Turn Malicious », *in*: *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA: USENIX Association, Aug. 2014, pp. 625–640, ISBN: 978-1-931971-15-7, URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/soska.

[189] *Specification gaming examples in AI*, 2020-10-09, URL: https://vkrakovna.wordpress.com/2018/04/02/specification-gaming-examples-in-ai/.

[190] Diomidis Spinellis and Vaggelis Giannikas, « Organizational adoption of open source software », *in*: *Journal of Systems and Software* 85.*3* (2012), Novel approaches in the design and implementation of systems/software architecture, pp. 666–682, ISSN: 0164-1212, DOI: https://doi.org/10.1016/j.jss.2011.09.037.

[191] S. C. Sundaramurthy et al., « An Anthropological Approach to Studying CSIRTs », *in*: *IEEE Security Privacy* 12.*5* (2014), pp. 52–60.

[192] Sathya Chandran Sundaramurthy et al., « A Human Capital Model for Mitigating Security Analyst Burnout », *in*: *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, Ottawa: USENIX Association, July 2015, pp. 347–359, ISBN: 978-1-931971-249, URL: https://www.usenix.org/conference/soups2015/proceedings/presentation/sundaramurthy.

[193] *Suricata - Open Source IDS / IPS / NSM engine*, 2020-12-11, URL: https://suricata-ids.org/.

[194] Christian Szegedy et al., « Intriguing properties of neural networks », *in*: *arXiv preprint arXiv:1312.6199* (2013).

[195] *Talos - Author of the Official Snort Rule Sets*, 2020-12-11, URL: https://snort.org/talos.

[196] Nazgol Tavabi et al., *DarkEmbed: Exploit Prediction With Neural Language Models*, 2018, URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17304.

[197] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, « Toward Credible Evaluation of Anomaly-Based Intrusion-Detection Methods », *in*: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.*5* (2010), pp. 516–524.

[198] *Term Frequency - Inverse Document Frequency statistics*, 2020-12-11, URL: https://jmotif.github.io/sax-vsm_site/morea/algorithm/TFIDF.html.

[199] Amir Teshome, Louis Rilling, and Christine Morin, « Including Security Monitoring in Cloud SLA », *in*: *SEC2 2016 - Second workshop on Security in Clouds*, Lorient, France, July 2016, URL: https://hal.inria.fr/hal-01354974.

[200] *The 2019 Official Annual Cybersecurity Jobs Report*, 2020-20-07, URL: https://www.herjavecgroup.com/2019-cybersecurity-jobs-report-cybersecurity-ventures/.

[201] *The Deprecation of SMB1 – You should be planning to get rid of this old SMB dialect*, 2020-04-08, URL: https://docs.microsoft.com/en-us/archive/blogs/josebda/the-deprecation-of-smb1-you-should-be-planning-to-get-rid-of-this-old-smb-dialect.

[202] *The Mad Dash to Find a Cybersecurity Force - The New York Times*, 2020-20-07, URL: https://www.nytimes.com/2018/11/07/business/the-mad-dash-to-find-a-cybersecurity-force.html.

[203] *The MITRE Corporation*, 2020-08-04, URL: https://www.mitre.org/.

[204] *The Myth of Software and Hardware Vulnerability Management*, 2016-05-04, URL: https://www.foo.be/2016/05/The_Myth_of_Vulnerability_Management/.

[205] *The Untold Story of NotPetya, the Most Devastating Cyberattack in History*, 2020-04-08, URL: https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/.

[206] *The White House Blames Russia for NotPetya, the 'Most Costly Cyberattack In History'*, 2020-04-08, URL: https://www.wired.com/story/white-house-russia-notpetya-attribution/.

[207] *Too many machine learning papers?*, 2020-10-09, URL: https://data-mining.philippe-fournier-viger.com/too-many-machine-learning-papers/.

[208] *Total WannaCry losses pegged at 4 billion dollars*, 2020-04-08, URL: https://www.reinsurancene.ws/total-wannacry-losses-pegged-4-billion/.

[209] Ruben Trapero et al., « A novel approach to manage cloud security SLA incidents », *in*: *Future Generation Computer Systems* 72 (2017), pp. 193 –205, ISSN: 0167-739X, DOI: https://doi.org/10.1016/j.future.2016.06.004, URL: http://www.sciencedirect.com/science/article/pii/S0167739X16301844.

[210] *United States Securities And Exchange Commission - Form 8-K - Community Health Systems, Inc.* 2020-31-07, URL: https://www.sec.gov/Archives/edgar/data/1108109/000119312514312504/d776541d8k.htm.

[211] *University of Brescia dataset*, 2020-12-11, URL: http://netweb.ing.unibs.it/~ntw/tools/traces/.

[212] *U.S. declares North Korea carried out massive WannaCry cyberattack*, 2020-04-08, URL: https://www.washingtonpost.com/world/national-security/us-set-to-declare-north-korea-carried-out-massive-wannacry-cyber-attack/2017/12/18/509deb1c-e446-11e7-a65d-1ac0fd7f097e_story.html.

[213] *US10616258B2 - Security information and event management*, 2020-01-09, URL: https://patents.google.com/patent/US10616258B2/en.

[214] *US10699012B2 - Endpoint detection and response utilizing machine learning*, 2020-01-09, URL: https://patents.google.com/patent/US10699012B2/en.

[215] *US20080091681A1 - Architecture for unified threat management*, 2020-01-09, URL: https://patents.google.com/patent/US20080091681A1/en.

[216] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik, « Dimensionality reduction: a comparative », *in*: *J Mach Learn Res* 10.*66-71* (2009), p. 13.

[217] Eduardo K Viegas, Altair O Santin, and Luiz S Oliveira, « Toward a reliable anomaly-based intrusion detection in real-world environments », *in*: *Computer Networks* 127 (2017), pp. 200–216.

[218] Ruowen Wang et al., « EASEAndroid: Automatic Policy Analysis and Refinement for Security Enhanced Android via Large-Scale Semi-Supervised Learning », *in*: *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C.: USENIX Association, Aug. 2015, pp. 351–366, ISBN: 978-1-939133-11-3, URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/wang-ruowen.

[219] Z. Wang, « Deep Learning-Based Intrusion Detection With Adversaries », *in*: *IEEE Access* 6 (2018), pp. 38367–38384.

[220] *WannaCry: the ransomware worm that didn't arrive on a phishing hook*, 2020-04-08, URL: https://nakedsecurity.sophos.com/2017/05/17/wannacry-the-ransomware-worm-that-didnt-arrive-on-a-phishing-hook/.

[221] Emil Wåreus and Martin Hell, « Automated CPE Labeling of CVE Summaries with Machine Learning », *in*: *Detection of Intrusions and Malware, and Vulnerability Assessment*, ed. by Clémentine Maurice et al., Cham: Springer International Publishing, 2020, pp. 3–22, ISBN: 978-3-030-52683-2.

[222] *What is cryptojacking? How to prevent, detect, and recover from it*, 2020-11-11, URL: https://www.csoonline.com/article/3253572/what-is-cryptojacking-how-to-prevent-detect-and-recover-from-it.html.

[223] *What Is EDR and Why Do You Need It?*, 2020-28-11, URL: https://blog.devolutions.net/2019/08/what-is-edr-and-why-do-you-need-it.

[224] *What is Mimikatz: The Beginner's Guide*, 2020-04-08, URL: https://www.varonis.com/blog/what-is-mimikatz/.

[225] *What is WannaCry ransomware?*, 2020-04-08, URL: https://www.kaspersky.com/resource-center/threats/ransomware-wannacry.

[226] John S Whissell and Charles LA Clarke, « Improving document clustering using Okapi BM25 feature weighting », *in*: *Information retrieval* 14.*5* (2011), pp. 466–487.

[227] *White House, spy agencies deny NSA exploited 'Heartbleed' bug*, 2020-31-07, URL: https://www.reuters.com/article/us-cybersecurity-internet-bug-nsa/white-house-spy-agencies-deny-nsa-exploited-heartbleed-bug-idUSBREA3A1XD20140411.

[228] Clifford C Wright, *Host intrusion prevention system using software and user behavior analysis*, US Patent 8,607,340, 2013.

[229] Chaowei Xiao et al., « From Patching Delays to Infection Symptoms: Using Risk Profiles for an Early Discovery of Vulnerabilities Exploited in the Wild », *in*: *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, Aug. 2018, pp. 903–918, ISBN: 978-1-939133-04-5, URL: https://www.usenix.org/conference/usenixsecurity18/presentation/xiao.

[230] L. Xiao et al., « Security in Mobile Edge Caching with Reinforcement Learning », *in*: *IEEE Wireless Communications* 25.*3* (2018), pp. 116–122, DOI: 10.1109/MWC.2018.1700291.

[231] Ning Xie et al., « Explainable deep learning: A field guide for the uninitiated », *in*: *arXiv preprint arXiv:2004.14545* (2020).

[232] Tianyi Xing et al., « Snortflow: A openflow-based intrusion prevention system in cloud environment », *in*: *2013 second GENI research and educational experiment workshop*, IEEE, 2013, pp. 89–92.

[233] C. Yin et al., « A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks », *in*: *IEEE Access* 5 (2017), pp. 21954–21961.

[234] A. Young and Moti Yung, « Cryptovirology: extortion-based security threats and countermeasures », *in*: *Proceedings 1996 IEEE Symposium on Security and Privacy*, 1996, pp. 129–140, DOI: 10.1109/SECPRI.1996.502676.

[235] Su Zhang, Xinwen Zhang, and Xinming Ou, « After We Knew It: Empirical Study and Modeling of Cost-effectiveness of Exploiting Prevalent Known Vulnerabilities Across IaaS Cloud », *in*: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, Kyoto, Japan: ACM, 2014, pp. 317–328, ISBN: 978-1-4503-2800-5, DOI: 10.1145/2590296.2590300, URL: http://doi.acm.org/10.1145/2590296.2590300.

[236] Ciyou Zhu et al., « Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization », *in*: *ACM Trans. Math. Softw.* 23.*4* (Dec. 1997), 550–560, ISSN: 0098-3500, DOI: 10.1145/279232.279236, URL: https://doi.org/10.1145/279232.279236.

[237] Xiaojin Jerry Zhu, *Semi-supervised learning literature survey*, tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2005.

Appendix A

N-Day Vulnerabilities Case Studies

In this appendix we present three case studies for major n-day vulnerabilities that had widespread real-world impact: Heartbleed, Shellshock, and EternalBlue.

A.1 Heartbleed

CVE-2014-0160 [140] or *Heartbleed* [61] was a software vulnerability affecting the widely-used cryptography library OpenSSL [149]. Its life cycle is depicted in Figure A.1. It was introduced by a PhD student, Robin Seggelmann, on December 31st 2011 with the vulnerable code reviewed and merged by OpenSSL core developer Stephen Henson without the flaw being corrected. The vulnerability was then released as part of OpenSSL 1.0.1 on March 14th 2012. For the next two and half years the vulnerability stayed dormant before being independently discovered by Neel Metha at Google and the Codenomicon team, on March 21st and April 2nd 2014 respectively. Both reported it to the OpenSSL team, which disclosed the vulnerability and released the corrective patch 1.0.1g on April 7th 2014.

Heartbleed allowed an attacker to read a random slice of memory from a remote web server process, including the content of user requests (in particular login and password information) as well as the web server private keys used for securing TLS / HTTPS communications. Between 24 % and 55 % of popular HTTPS websites were deemed vulnerable to the attack at the time of disclosure. The disclosure of Heartbleed triggered a world-wide race against the clock between attackers and defenders. 22 hours after disclosure, 95 out the 100 most popular websites (as estimated by the Alexa ranking [4]) were non-vulnerable, either because they applied the patch or because they were not affected in the first place. Meanwhile, the first Heartbleed attack ever recorded had occured a few minutes earlier. 48 hours after disclosure, more than 110 000 websites out of the top

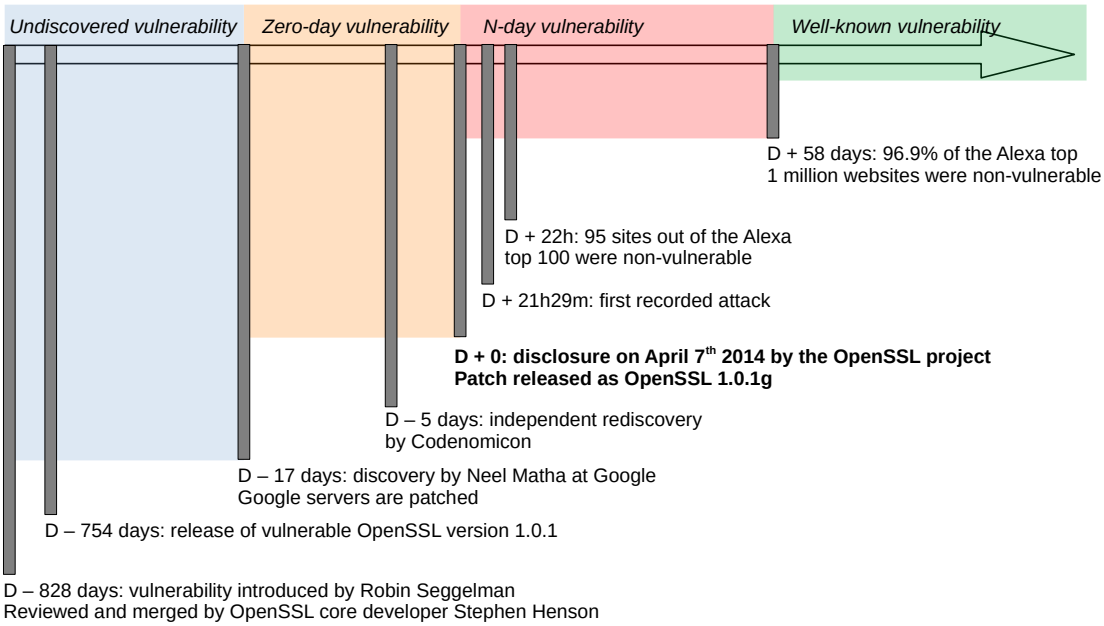

Figure A.1 – The life cycle of the Heartbleed vulnerability (CVE-2014-0160).

one million most popular websites were still vulnerable, while attacks intensified. Two months later, the number was down to 31 000 (with 96.9 % of the top 1 million websites being non-vulnerable at the time), as awareness of the vulnerability had spread among the system administrators community [61].

In many ways, Heartbleed was the ground zero of modern n-day vulnerabilities.

A first important characteristic of Heartbleed is that its usage both as a zero-day and as a fully public vulnerability has been limited or non-existent. While there have been speculation about government agencies such as the NSA being aware of the vulnerability before its known discovery, the NSA denied having knowledge of the vulnerability prior to its disclosure [227] and all evidences of the contrary have turned out to be inconclusive [87]. Meanwhile, its long-term impact beyond the immediate aftermath of the disclosure has been fairly limited as well.

Still, Heartbleed had important real-world impact during the days and weeks following the disclosure. The most notable report was the data theft of non-medical personal data for 4.5 millions patients from the hospital chain Community Health Systems (CHS) less than a week after disclosure [162] [210]. The attackers used Heartbleed as an initial vector to steal CHS security access secrets then used those to achieve the actual data theft.

The unprecedented nature of the Heartbleed disclosure led the public opinion to understandably focus on what went wrong at the time, such as the massive underfunding of critical open-source projects like OpenSSL [94]. However arguably some things went right with Heartbleed: the patch was issued on the day of disclosure. This patch completely fixed the problem and it did not cause additional issues. Thanks to an efficient marketing campaign [86], the awareness of Heartbleed among the technical community and the general public had been extremely high (remarkably, 60 % of American adults had heard of Heartbleed when polled in April 2014). This, coupled with the fact that the vulnerability targeted web servers usually administered by professional staff, led to an overall timely patch dissemination, limiting the global impact of the vulnerability. Still, Heartbleed was a wake-up call for the technical community and led to the start of multiple initiatives aimed at improving the security of critical Internet infrastructure software. These initiatives included the Core Infrastructure Initiative [45] and the creation of the Project Zero team at Google [11]. Consensus among security experts at the time was that Heartbleed would be the first of similar vulnerabilities [95], and the future proved them right.

A.2 Shellshock

Shellshock was a family of vulnerabilities (CVE-2014-6271 [141] quickly followed by CVE-2014-6277, CVE-2014-6278, CVE-2014-7169, CVE-2014-7186, and CVE-2014-7188) affecting the ubiquitous bash shell, used in the vast majority of Unix systems across the world. Its life cycle is depicted in Figure A.2.

Shellshock had an even more extreme timeline than Heartbleed. The original vulnerability was introduced on August 5$^{\text{th}}$ 1989 by original bash author Brian Fox and was then released as part of Bash 1.03 on September 1$^{\text{st}}$ 1989. It stayed undiscovered *during more than twenty five years* before being discovered on September 12$^{\text{th}}$ 2014 by Stéphane Chazelas who then contacted current bash maintainer Chet Ramey on the same day [56].

Shellshock allowed an attacker to achieve remote code execution on a vulnerable machine because of unforeseen consequences of the way bash historically handled function definitions in environment variables. Attack vectors included CGI scripts on Apache or Nginx [58] and mail delivery with qmail [157] which both allowed remote code execution, while another attack vector through OpenSSH allowed privilege escalation [184].

Taken together these attack vectors had the potential to leave millions of hosts vulnerable to remote attacks. This led Chazelas and Ramey to plan a *coordinated disclosure* along

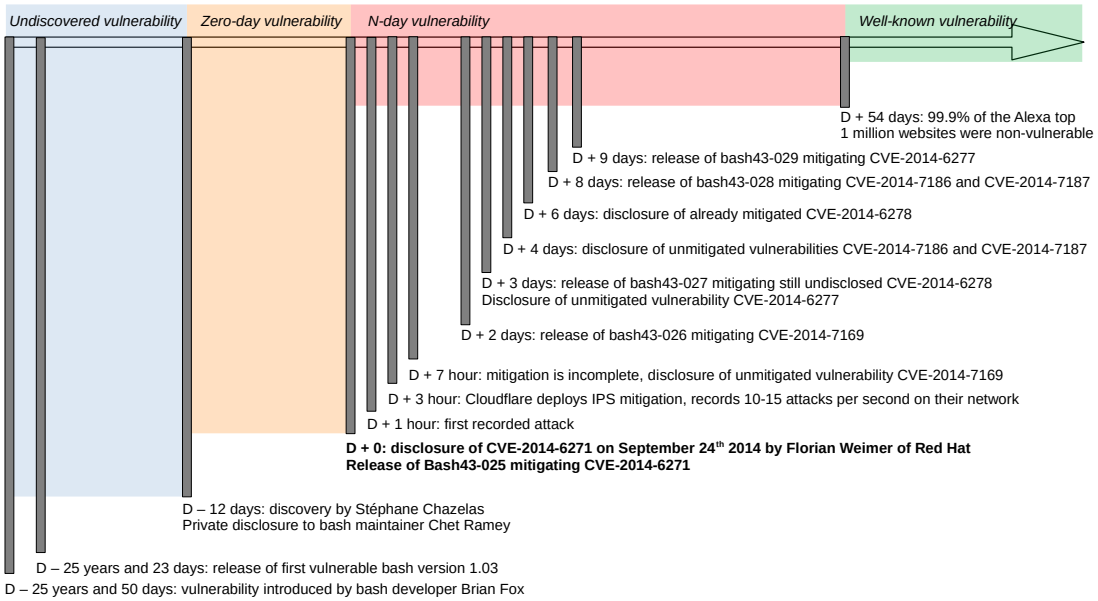

Figure A.2 – The life cycle of the Shellshock vulnerabilities (CVE-2014-6271, CVE-2014-6277, CVE-2014-6278, CVE-2014-7169, CVE-2014-7186, CVE-2014-7188).

with Florian Weimer from Red Hat to make sure all major Linux and Unix distributions would be ready to distribute a patch at the time of disclosure.

CVE-2014-6271 was disclosed by Weimer on September 24$^{\text{th}}$ 2014 at 14:00 UTC [49], along with the release of Bash 4.3-025 which provided a mitigation. All major Linux and Unix distributions included it in their security repositories and urged their users to upgrade as soon as possible. In the next hours, multiple things happened.

The first recorded attack occurred less than an hour after disclosure [84] as taking control of a vulnerable web server was as simple as executing a simple curl command such as this one:

```
curl -H "User-Agent: () { :; }; echo vulnerable" http://victim.com/
```

Three hours after disclosure, the Cloudflare CDN started logging and blocking Shellshock attacks at the HTTP request level using their *Web Application Firewall* (WAF) [40], first for their customers, and a few days later for all their users. From the moment they activated their mitigation they recorded as many as 10 to 15 attacks per second on their network continuously for several weeks [38].

More worryingly, in the evening of the disclosure day it became apparent that the 4.3-025 patch was incomplete [56] and that the problem was broader than initially thought. For more than a week the situation stayed chaotic with five more related vulnerabilities discovered and patched in a few days, with several of these vulnerabilities being disclosed without any mitigation in place.

At the beginning of October 2014 bash maintainer Chet Ramey and other parties involved in the vulnerability mitigation became more confident they had mitigated all variants of Shellshock and the daily new releases of Bash stopped.

Patch dissemination continued and two months after disclosure 99.9 % of the Alexa top 1 million websites were non-vulnerable [58].

Like Heartbleed, Shellshock had real-world impact. In the first hours and days after the disclosure, attackers used the vulnerability to quickly assemble a massive botnet of infected computers dubbed *wopbot*, which they then used to provoke distributed denial of service against the Akamai CDN and the American Department of Defense (DoD) networks [70]. To get a sense of how confusing the initial moments after the disclosure were, Yahoo networks were initially believed to have been breached using Shellshock [163] although it later turned out that Yahoo had patched its infrastructure but a Shellshock attack accidentally triggered a non-Shellshock code execution bug in Yahoo custom monitoring infrastructure [3]. Fortunately there have not been reports of a major organization being directly compromised through Shellshock, possibly because CGI scripts (introduced in 1993) were already an old-fashioned technology in 2014.

Still Shellshock was arguably worse than Heartbleed in several ways. The fact that it had stayed undiscovered for 25 years (compared to two years for Heartbleed) raises even more questions about potential secret discovery and exploitation (however like Heartbleed there are no credible reports of usage before Stéphane Chazelas's discovery). It was considerably easier to use than Heartbleed and allowed remote code execution on the vulnerable machine while Heartbleed only allowed unauthorized reads of information. The Heartbleed OpenSSL patch was flawless while the initial Shellshock mitigation was incomplete, requiring no less than five releases of bash over a week to fully mitigate the vulnerability. Moreover these additional patches happened in *full disclosure*, with the vulnerability having been disclosed publicly and used in the wild already.

While both Heartbleed and Shellshock were critical n-day vulnerabilities with actual real-world impact, thanks to the tireless work of countless system administrators around the world the actual damages caused by these vulnerabilities have been somewhat lim-

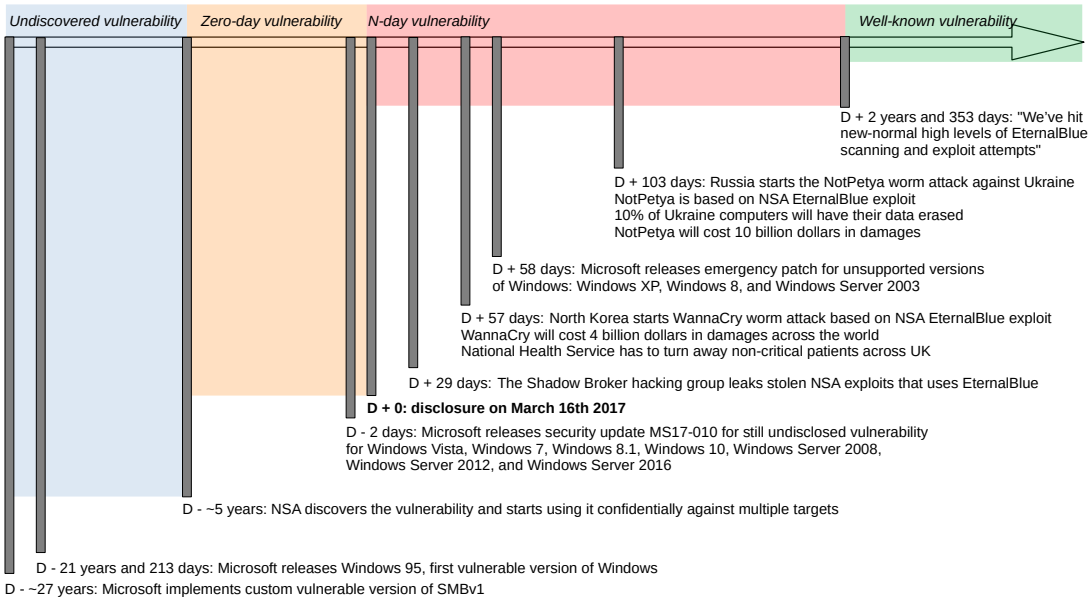

Figure A.3 – The life cycle of the CVE-2017-0144 vulnerability (EternalBlue).

ited. However they were warning shots of what the impact of a badly handled n-day vulnerability could look like, which leads us to our last case-study.

A.3 EternalBlue

CVE-2017-0144 [142] is a software vulnerability in Microsoft's implementation of the first version of the Server Message Block protocol (SMBv1), which provides network access to resources such as files and printers. SMBv1 has been available on every version of Microsoft Windows from Windows 3.1 (released in 1992) onward [126]. SMBv2 superseded SMBv1 in 2007 and SMBv1 was deprecated in 2015 [201]. Still, SMBv1 was heavily used and Windows 10 (released in 2015) was the first version of Windows which did not install SMBv1 by default [186]. The life cycle of CVE-2017-0144 is depicted in Figure A.3.

In 2012 the American *National Security Agency* (NSA) discovered CVE-2017-0144 [137], and realized it could compromise any version of Windows available at the time using the vulnerability as a vector for remote code execution. It chose not to report it to Microsoft, and instead started using it as a zero-day vulnerability in confidential cyber-warfare operations. To this end it created an exploit for vulnerability CVE-2017-0144. This exploit

was internally code-named *EternalBlue*. Then the situation stayed unchanged until March 2017.

On March 14$^{\text{th}}$ 2017's patch Tuesday [123], Microsoft released a mitigation for the still undisclosed CVE-2017-0144. And then disclosed the vulnerability two days later on March 16$^{\text{th}}$ [142].

For some time, nothing happened. On the attack side, CVE-2017-0144 is not easy to exploit and while the vulnerability had been disclosed, not many details about it were revealed. However on the defense side patch propagation was slow as well. While individuals and small organizations get Windows updates automatically, major organizations managing vast Windows computer fleets can choose how quickly they want to update their fleet to ensure backward-compatibility. Some of them end up being very late in applying critical security updates.

On April 14$^{\text{th}}$ 2017, 29 days after disclosure, something peculiar happened. An anonymous hacking group which named itself *The Shadow Brokers* publicly released several sets of leaked, highly sensitive NSA exploit software, which included EternalBlue [136]. The identity and motivation of the Shadow Brokers have not been confirmed to this day, and neither how did they get access to the exploit software. Some speculated the group was linked to an NSA insider threat [89]. Others such as Edward Snowden suggested the Shadow Brokers were linked to the Russian government and acted in the context of the diplomatic tension regarding the recent American Democratic National Committee (DNC) cyber attacks [63]. Some speculated Microsoft had gotten an advance warning about the leak, allowing them to locate and patch the vulnerability in time for the release of the exploit [137].

Whatever their reasons were, the Shadow Brokers effectively put highly efficient cyber weapons in the hand of the general public. While CVE-2017-0144 had been difficult to exploit so far, following the public release of the NSA tools taking control of a vulnerable Windows computer was now as easy as typing a command line into a shell. Even though the vulnerability was not zero-day anymore, patch dissemination was poor and a vast number of Windows computers across the world were vulnerable to EternalBlue. Attackers around the world took notice, including those from other cyberwarfare government agencies.

The *WannaCry* worm and ransomware attack was launched on May 12$^{\text{th}}$ 2017 [220]. It was based on EternalBlue but was modified to behave as a self-replicating computer worm, encrypting the files present on the infected computer host then requesting a Bitcoin

ransom in order to get the decryption key. The US government would formally attribute the Wannacry worm to North Korea later in 2017 [212]. The worm had widespread consequences: 200 000 computers across 150 countries had been reported to have been infected [225]. Worryingly, one the most impacted organization was the United Kingdom's National Health Service, whose computers and networks came to a complete halt for a few days [131]. Ghafur et al [77] did a comprehensive study of the impact of WannaCry on NHS patient care and concluded that impacted hospitals had to divert away 4 % of their emergency patients and 9 % of their non-emergency patients and had a 50 % increase in appointment cancellation. However, mortality rate was not impacted although the authors consider it to be a crude measure of patient harm, which they claimed not being able to evaluate. Globally WannaCry has been estimated to cost between 4 and 8 billion dollars in damages across the world [208] [161].

WannaCry was not the only cyberwarfare act based on EternalBlue. On June 27$^{\text{th}}$ 2017, Russia launched the *NotPetya* malware attack against Ukraine in the context of ongoing tensions between the two countries [205]. NotPetya was also a self-replicating worm, even more devastating than WannaCry as it combined EternalBlue with an older attack vector called Mimikatz [224], using EternalBlue to breach into networks then Mimikatz to replicate inside them [135]. NotPetya deceived its victims into believing it acted like a ransomware but was instead plainly destroying the infected computer files without encrypting them [135]. The attack shut down most major public and private organizations in Ukraine, including banks, power companies, airports, hospitals, credit card payment systems, with an estimated 10 % of all computers across the country having their data erased [205]. It spread widely beyond Ukraine too, paralyzing 17 cargo port terminal facilities across the world administered by the Danish company Maersk, impacting the delivery of hundred of thousands of shipping containers for a full week [205]. The American pharmaceutical company Merck & Co saw 30 000 of its workstations and 7 500 servers impacted, halting sales, manufacturing, and research during two weeks, while some of its researchers reported having lost 15 years of research data [121]. Other affected organizations included Fedex [134], French construction company Saint-Gobain [133] [205], and more. A US White House representative called the NotPetya attack the "*most destructive and costly cyberattack in history*" [206] and estimated its overall cost to 10 billion dollars [205].

While WannaCry and NotPetya were the most visible consequences of EternalBlue, the leaked NSA exploit also became a widely used tool for non-nation state attackers. It

was still one of the most common attacks seen in the wild in 2018 [146] [158], and only in 2019 would its usage start to decrease, prompting the security company Rapid7 to assert in March 2020 that "*we've hit new normal high levels of EternalBlue scanning and exploit attempts*" [160].

A complex question related to EternalBlue is the non-disclosure of critical zero-day vulnerabilities discovered by government agencies such as the NSA. In the wake of the WannaCry attack US House Representative Ted Lieu introduced the *Protecting Our Ability to Counter Hacking Act* (PATCH Act) [96] in the US Congress, which would establish an independent Vulnerability Equities Review Board that would decide for each vulnerability found by agencies such as the NSA or CIA, if it should be disclosed or not. However the proposal has not been ratified so far.

It seems reasonable to call EternalBlue a world-scale disaster. The overall damage caused by the CVE-2017-0144 vulnerability is estimated to be between 14 to 18 billion dollars when cumulating the cost of both the WannaCry and NotPetya attacks. It paralyzed the global infrastructure of an entire country as well as many organization's ones across the world, including many hospitals. No vulnerability or exploit disclosure has come close, before or since then. A striking aspect of EternalBlue's CVE-2017-0144 is how it had its most damaging impact as an n-day vulnerability and not as a zero-day: it was disclosed and patched 58 and 103 days prior to the launch of WannaCry and NotPetya respectively. EternalBlue makes arguably the case that a mishandled n-day vulnerability is many times more dangerous than either zero-day or public vulnerabilities.

The only positive news regarding the impact of EternalBlue is that there has been no fatal casualty reported in its aftermath. As WannaCry and NotPetya paralyzed multiple hospitals each and led some patients to be turned away, the absence of casualties could only be attributed to luck and to the bravery of many medical workers around the world.

A.4 First Fatal Casualty Related to an N-Day Vulnerability

For the first time in history, a fatal casualty was attributed to an n-day vulnerability during the last days of our work on this thesis. On September 10th, 2020, a ransomware allegedly exploiting the vulnerability CVE-2019-19781 (disclosed on December 27th, 2019 and affecting the Citrix ADC software) paralyzed the University Hospital Düsseldorf (UKD) in Germany. A patient in critical condition had to be transferred to another

hospital following the incident, and passed away during travel [159]. This is an ongoing story as we are writing. For this reason we do not provide a comprehensive analysis of this n-day vulnerability like for Heartbleed, Shellshock and EternalBlue.

Yet this tragic event is a milestone highlighting how dangerous n-day vulnerabilities can be not only for information systems but also for human lives.

Résumé

Cette thèse porte sur la protection des systèmes d'information contre les vulnérabilités n-day. Dans ce résumé nous commençons par établir le risque posé par les vulnérabilités n-day contre les systèmes d'information, puis présentons les mécanismes de défenses utilisés aujourd'hui, ainsi que leur limites. Nous présentons ensuite l'objectif de notre thèse et nos contributions. Nous terminerons par des perspectives à la suite de nos travaux.

1. Risques liés aux vulnérabilités n-day pour les systèmes d'information

Dans cette section nous proposons une définition moderne des systèmes d'information, définissons les vulnérabilités n-day, et mettons en avant le risque qu'elles posent sur les systèmes d'information. Nous présentons ensuite trois vulnérabilités illustrant le risque posé par les vulnérabilités n-day. Nous terminons la section en mettant en avant le besoin d'un effort de sécurité continu dans le temps.

1.1 Les systèmes d'information modernes

Nous définissons un système d'information comme incluant les personnes, matériels, logiciels, et processus organisationnels permettant à une organisation de manipuler l'information lui permettant de prendre des décisions. Bien que les systèmes d'information soient plus anciens que l'informatique elle même, les systèmes d'information modernes ont plusieurs caractéristiques récurrentes. La première est d'être *multi-partis*. L'organisation qui utilise un système d'information n'est pas nécessairement la même que l'organisation qui le maintient en condition opérationnelle, ou qui le maintient en condition de sécurité, ou qui développe les composants logiciels du système d'information. Une seconde caractéristique des systèmes d'information modernes est d'être *composites*, et d'inclure du matériel et logiciel réalisés par de très nombreux fabricants et éditeurs. En particulier, une partie de ces logiciels sont publics, voire open-source, tandis qu'une partie peut n'être ni l'un ni l'autre. Enfin, les systèmes d'information modernes sont des *systèmes complexes*,

pour lesquels cartographier intégralement leur nombreux composants et l'interaction entre ces composants est très difficile.

1.2 L'ère de la cyber-insécurité

La cyber-sécurité est un enjeu majeur du 21ème siècle, pour la communauté technique d'une part et pour la société en général d'autre part. Toujours plus de domaines sont automatisés via l'informatique, rendant la sécurité de ces systèmes informatiques toujours plus critique. Toutes les métriques observables pointent vers la même conclusion : les enjeux montent continuellement.

Durant la dernière décennie deux tendances ont contribué à rendre les cyber-attaques considérablement plus dangereuses que dans le passé. La première est le développement de la *cyber-guerre* : de nombreux États disposent aujourd'hui de capacité cyber-offensives évoluées, et ciblent des acteurs publics ou privés pour des motifs géopolitiques. Certaines de ces attaques étatiques ont coûté des milliards de dollars à leur victimes, et d'autres ont été suivies de représailles militaires. La seconde tendance est l'apparition de cyber-attaques criminelles à motif financier, via de nouveaux types de malware incluant les *ransomware* et les *cryptominers*. En particulier les ransomware, qui chiffrent les données d'un ordinateur infecté puis exigent une rançon en échange de la clé de déchiffrement, ont déjà coûté des milliards de dollars de dommage à leurs victimes et ont récemment mené à un décès suite à la paralysie du système d'information d'un hôpital allemand.

1.3 Vulnérabilités n-day notables

Une vulnérabilité *n-day* est une vulnérabilité *publique*, *récemment divulguée*, mais qui n'est pas encore *bien connue*. *Heartbleed*, *Shellshock*, et *EternalBlue* sont trois célèbres vulnérabilités illustrant les risques que posent les vulnérabilités n-day.

Heartbleed était une vulnérabilité sur le logiciel OpenSSL qui permettait à un attaquant d'obtenir les clés privées SSL de la majorité des sites web accessibles depuis Internet. Plusieurs milliers d'administrateurs système ont été contraints de déployer en urgence un correctif pour sécuriser leurs systèmes.

Shellshock était une vulnérabilité affectant toutes les versions de bash shell sorties entre 1989 et 2014. Elle pouvait être exploitée à distance via des scripts CGI et était si facile à utiliser que de premières attaques ont été détectées dans l'heure suivant la divulgation de la vulnérabilité. Elle mena à des attaques par déni-de-service distribué

contre le CDN Akamai et le département de la défense américaine.

EternalBlue était un logiciel confidentiel de la NSA permettant l'exploitation d'une vulnérabilité présente dans l'implémentation du protocole SMB par Microsoft. La vulnérabilité affectait toutes les versions de Windows de Windows 3.1 à Windows 10. La diffusion publique du logiciel EternalBlue mena au lancement de l'attaque ransomware *WannaCry* et de la cyber-attaque étatique *NotPetya* contre l'Ukraine. Ces deux attaques coûtèrent plusieurs milliards de dollars de dommages autour du monde, et paralysèrent plusieurs hôpitaux, les conduisant à rediriger des patients vers d'autres destinations.

Ces exemples montrent que le risque induit par les vulnérabilités n-day n'est pas théorique. De futures vulnérabilités n-day auront probablement un impact similaire ou supérieur. Les dizaines de vulnérabilités divulguées chaque jour font peser sur les systèmes d'information un facteur de risque *dynamique au cours du temps*.

1.4 La sécurité des systèmes d'information au cours du temps

Un système d'information peut avoir un profil de risque très différent à deux instants distincts, même si aucune modification ne lui a été apportée dans l'intervalle. Le système peut être adéquatement protégé contre toutes les vulnérabilités publiquement connues à un moment donné, puis entrer dans une période de risque suite à la divulgation d'une dangereuse vulnérabilité qui n'a pas encore reçu de mitigation.

Cela signifie que la sécurité des systèmes d'information ne peut pas être une activité ponctuelle : le risque doit être évalué de façon continue et des actions défensives doivent être prises de façon régulière pour s'assurer de la sécurité des systèmes sur le temps long.

2. Motivation

La section précédente a établi le risque posé par les vulnérabilités n-day envers les systèmes d'information. Dans cette section nous justifions notre conviction que des efforts de recherche supplémentaires doivent être menés pour atteindre un niveau de défense adéquat face à ce risque.

2.1 Les vulnérabilités n-day : un problème de long terme

L'impact concret des vulnérabilités n-day provient de trois tendances : la réutilisation logicielle, la divulgation publique de vulnérabilités dans les logiciels et matériels publics,

et l'usage de l'informatique dans un nombre croissant de cas d'usages, incluant des contextes critiques. Aucune de ces tendances ne semble perdre de sa pertinence à court et moyen terme. La communauté d'ingénierie logicielle considère la réutilisation logicielle comme une bonne pratique apportant de nombreux bénéfices, particulièrement via l'essor du logiciel libre. La divulgation publique et coordonnée de vulnérabilités logicielles est également considérée comme une bonne pratique par la communauté de la sécurité informatique. Enfin, inclure plus de connectivité et d'intelligence dans toujours plus d'objets et de lieux est le concept de multiples industries majeures de notre époque, à commencer par *l'Internet des Objets*. Tout cela implique que le risque posé par les vulnérabilités n-day est peu susceptible de diminuer dans le futur, et il est même probable qu'il augmente.

2.2 Les pratiques actuelles sont inadéquates pour se défendre contre les vulnérabilités n-day

Après avoir établi que les vulnérabilités n-day sont susceptibles de devenir de plus en plus problématiques dans le futur, nous passons maintenant en revue les pratiques actuelles utilisées pour s'en défendre et mettons en lumière leurs déficiences.

2.2.1 Mises à jour logicielles

Garder à jour les logiciels d'un système d'information est une pièce vitale de toute stratégie de mitigation du risque lié aux vulnérabilités n-day. Cependant, dans les minutes ou heures qui suivent une divulgation, il est possible qu'aucun correctif ne soit disponible pour cette vulnérabilité. Et même si un correctif est disponible, il est possible qu'il contienne un bug ou présente un problème de rétro-compatibilité empêchant de le déployer avec célérité. Dans un contexte où certaines vulnérabilités n-day comme Shellshock sont exploitées dans l'heure suivant leur divulgation, des mises à jour logicielles ponctuelles sont nécessaires mais pas suffisantes.

2.2.2 Systèmes de détection d'intrusion par signature

Les *Systèmes de Détection d'Intrusion* (IDSs) surveillent l'activité d'un système informatique et tentent de séparer les comportements *légitimes* des comportements *malveillants*. Plusieurs approches de détection d'intrusion existent et l'une d'entre elles a déjà eu un impact important dans la lutte contre les vulnérabilités n-day : la détection et prévention d'intrusion par signature, qu'on retrouve sous la forme de *Web Application*

Firewalls (WAF) et *Next-Generation Firewalls* (NGFW) dans l'industrie. En particulier, de nombreux fournisseurs Cloud proposent une offre WAF à leurs clients et surveillent le traffic réseau qui leur est destiné, leur permettant de bloquer les requêtes correspondant à une *signature* rédigée manuellement pour détecter de l'activité malveillante.

Bien que très utiles pour se défendre contre les vulnérabilités connues, ces techniques sont imparfaites dans le cadre de la défense contre les vulnérabilités n-day. Créer une signature pour détecter l'usage d'une vulnérabilité requiert du temps, des efforts, de l'expertise, et une connaissance approfondie de l'attaque à détecter. Il est donc très difficile de rédiger de façon systématique une signature pour chaque nouvelle vulnérabilité dans les premières heures suivant sa divulgation.

2.2.3 Sécurité par design et défense en profondeur

La *sécurité par design* est un ensemble de pratiques de conception et implémentation logicielles visant à incorporer la sécurité comme un élément primordial d'un projet logiciel dès les premières étapes de son cycle de vie. Bien qu'il soit possible d'obtenir d'importants gains en sécurité via ces approches, elles sont également inadéquates pour défendre des systèmes d'information entiers contre les vulnérabilités n-day, pour deux raisons. La première est que ces approches visent à limiter le nombre de vulnérabilités présentes dans un composant logiciel mais ne peuvent prétendre les retirer toutes. La seconde est que les systèmes d'information sont composites : même si certains composants d'un système d'information sont conçus en respectant les principes de la sécurité par design, le système entier inclut probablement des composants logiciels ou matériels historiques moins sécurisés.

Il est possible de concevoir des systèmes d'information entiers via les pratiques de sécurité par design, et de les rendre plus sécurisés que leur composants individuels. Une pratique très utilisée dans ce but est le concept de *défense en profondeur*. La défense en profondeur consiste à utiliser simultanément plusieurs mécanismes de défense afin que lorsqu'une couche échoue à contrer une attaque, les autres couches aient une chance d'y parvenir à leur tour. Pour la mitigation de vulnérabilités n-day, une stratégie de défense en profondeur pourrait inclure une mise à jour diligente des composants logiciels *et* l'utilisation de systèmes de détection d'intrusion *et* l'usage de composants conçus via la sécurité par design. La défense en profondeur est une part cruciale de nombreuses stratégies de défense, mais même cette pratique a des limites. En particulier, les divulgations de vulnérabilités sont des évènements publiques, qui informent simultanément l'attaquant

et le défenseur de la possibilité de contourner une couche de défense. Cependant, un attaquant persistant a potentiellement pu trouver secrètement comment contourner toutes les autres couches de défense, rendant le système vulnérable sans que le défenseur ne le sache.

2.2.4 Air Gap

L'*air gap* est la pratique d'isoler physiquement un système d'information de l'extérieur, et notamment de retirer tout lien réseau avec Internet. Il s'agit d'une pratique courante pour les réseaux sensibles des forces armées de nombreux pays, ainsi que de nombreuses agences de renseignement. Un air gap est indubitablement utile pour la défense contre les vulnérabilités n-day, mais abandonner la possibilité d'un accès à Internet dans un système d'information est un compromis que peu d'organisations sont prêtes à faire.

De plus, il y a eu des cas documentés d'air gap compromis par des attaquants hautement motivés. L'exemple le plus spectaculaire est Stuxnet, une opération de cyber-guerre conjointe entre Israël et les États-Unis où un ver auto-répliquant se diffusant sur clé USB a pu se propager sur un réseau confidentiel Iranien placé derrière un air gap. Cela a mené à la destruction de centrifugeuses nucléaires iraniennes et au ralentissement du programme nucléaire de l'Iran.

2.3 Défis liés à la défense contre les vulnérabilités n-day

La défense contre les vulnérabilités n-day est une *course contre la montre*, car les défenseurs doivent agir plus rapidement que les attaquants. Mais agir trop rapidement peut aussi être dangereux : une mitigation inappropriée peut se révéler incomplète, voire endommager le système d'information. Mitiger le risque posé par les vulnérabilités n-day d'une façon à la fois sûre et rapide serait l'idéal, mais coûte très cher. De plus, les techniques de mitigation proactives, comme la sécurité par design ou la défense en profondeur, sont nécessaires mais pas suffisantes pour éliminer complètement ce risque. Cela mène les organisations considérant la sécurité de leur systèmes d'information comme critique à rechercher des temps de réaction toujours plus rapides face à la possibilité d'un incident de sécurité. Cependant, dans le cas des vulnérabilités n-day, toutes les techniques réactives actuelles sont limitées par le temps de réaction humain, par exemple le temps de développement d'un correctif par l'éditeur d'un logiciel ou de rédaction d'une signature IDS par des experts en sécurité. En ce sens, une façon prometteuse d'améliorer

significativement la sécurité des systèmes d'information contre les vulnérabilités n-day est de proposer de nouvelles techniques de mitigation réactive ne requérant pas d'intervention humaine.

3. Objectif de la thèse

L'objectif de cette thèse est de proposer de nouveaux outils pour l'analyse et la réaction automatiques face au risque posé par la divulgation de nouvelles vulnérabilités. Notre but ultime est de permettre à un système d'information de présenter une *sécurité constante au cours du temps*, sans que le flux quotidien de divulgations de vulnérabilités n'affecte le risque global auquel le système fait face.

Un objectif plus concret est de rassembler, immédiatement et automatiquement, et pour toute vulnérabilité nouvellement divulguée, des informations qui sont habituellement rassemblées par des humains experts au bout de plusieurs jours. Dans les premiers instants suivant la divulgation d'une vulnérabilité, de nombreuses informations à propos de celle-ci sont fragmentées, manquantes, ou présentes dans un format inadapté à une analyse automatique. Cela inclut d'une part des informations intrinsèques à la vulnérabilité en elle-même, comme le logiciel ou matériel qu'elle affecte, ou la sévérité hors-contexte de la vulnérabilité. Le délai dans l'obtention de ces données retarde d'autant le processus de gestion des vulnérabilités. D'autre part, les informations spécifiques au contexte dans lequel le composant affecté est utilisé sont également nécessaires mais difficiles à obtenir promptement. Cela inclut notamment le risque qu'une vulnérabilité divulguée crée pour un système d'information donné.

Rassembler toutes ces informations automatiquement est une condition nécessaire à une gestion plus rapide des vulnérabilités. Cependant ajouter de l'automatisation induit toujours le risque de transformer un processus décisionnel en une boîte noire ne pouvant pas être expliquée par quiconque. Nous postulons que des systèmes de défense automatisés ne peuvent être considérés comme de confiance que s'ils peuvent être compris et audités. L'explicabilité des décisions est donc un critère primordial de tous les systèmes de sécurité automatisés que nous proposons.

4. Contributions

Dans cette thèse nous proposons quatre contributions ayant pour but d'atteindre les objectifs décrits précédemment. La première est une proposition de stratégie complète pour défendre les systèmes d'information contre les vulnérabilités n-day. Les trois autres contributions solutionnent des problèmes de recherche mis en lumière par cette stratégie. Ces problèmes sont divisés en deux axes : automatiser l'analyse d'une vulnérabilité à sa divulgation et automatiser l'analyse du risque posé par la divulgation d'une vulnérabilité pour un système d'information donné.

4.1 Stratégie de défense des systèmes d'information contre les vulnérabilités n-day

Notre première contribution est l'établissement d'une stratégie complète visant à défendre un système d'information contre les vulnérabilités n-day. Les principales étapes de cette stratégie sont les suivantes :

1. Automatiquement inférer, d'une façon explicable, les informations disponibles à propos d'une vulnérabilité nouvellement divulguée, sans attendre l'analyse réalisée par un expert humain ultérieurement.
2. Automatiquement analyser, d'une façon explicable, le risque créé par la divulgation d'une vulnérabilité sur le système d'information à protéger.
3. Si nécessaire, déclencher une réaction de mitigation du risque posé par la vulnérabilité sur le système d'information.

Énoncer cette stratégie met en lumière plusieurs problèmes de recherche ouverts, pour lesquels nous proposons des solutions dans nos contributions suivantes.

4.2 Analyse automatisée de vulnérabilité à la divulgation

Nos deuxième et troisième contributions visent à obtenir une meilleure compréhension d'une vulnérabilité dans les instants suivant sa divulgation. La divulgation peut être une étape chaotique du cycle de vie d'une vulnérabilité, et des informations lisibles par un être humain (comme la description textuelle brute de la vulnérabilité) peuvent ne pas avoir d'équivalent lisible par une machine à ce stade.

Dans notre deuxième contribution nous montrons que le nom du logiciel ou matériel affecté par une vulnérabilité peut être déterminé à la divulgation via l'analyse de la description textuelle brute de la vulnérabilité, avant la publication de la méta-donnée officielle fournissant cette information (l'URI CPE). Nous réalisons cela via l'usage de techniques d'apprentissage machine et de recherche d'information, comme TF-IDF. Les résultats de notre processus d'analyse sont présentés sous la forme d'une liste pondérée et ordonnée de mots-clés. Notre protocole d'évaluation montre que des mots-clés liés au nom recherché sont présents dans les trois premières positions de la liste 86 % du temps. Ce problème ne semble pas avoir été abordé dans la littérature, malgré l'applicabilité d'un tel résultat. Depuis la publication de ces travaux, d'autres chercheurs ont proposé des travaux fondés sur cette contribution, démontrant l'intérêt de la communauté pour ce problème.

Dans notre troisième contribution nous montrons comment analyser et prédire automatiquement la sévérité d'une vulnérabilité à sa divulgation, en réalisant une prédiction explicable du vecteur CVSS de cette vulnérabilité. Nous atteignons une précision allant jusqu'à 96 % pour les champs CVSS individuels du vecteur CVSS, et notre prédiction du score CVSS de la vulnérabilité présente un taux de faux négatif inférieur à 0.2 pour 50 % des vulnérabilités, et inférieur à 3.5 pour 99 % d'entre elles. Tout comme notre précédente contribution, le problème de la prédiction de vecteur CVSS ne semble pas avoir été abordé dans la littérature malgré l'applicabilité d'un tel résultat.

Une exigence primordiale qui a guidé les choix effectués pour nos deux contributions est de s'assurer que les prédictions réalisées sont *explicables*. Cela nécessite des choix spécifiques de techniques d'apprentissage machine, afin de conserver une relation explicable entre les entrées et les sorties à chaque étape du processus de décision. Nous avons déterminé que des techniques comme *bag-of-words*, les *listes blanches*, *l'entropie conditionnelle*, *TF-IDF* et la *régression linéaire* étaient pertinentes pour la mise en place de processus de décision explicables.

Combinées, ces deux contributions créent un état de l'art quant à l'analyse automatisée de vulnérabilité à leur divulgation.

4.3 Analyse automatisée du risque posé par les vulnérabilités à leur divulgation

Dans notre quatrième contribution nous allons plus loin en automatisant l'analyse de risque des vulnérabilités nouvellement divulguées, dans le contexte d'un système d'information spécifique. Au lieu d'aborder le problème complexe de cartographier un système d'information entier, nous faisons usage de la connaissance consciente et inconsciente dont dispose l'équipe de sécurité du système d'information, en entrainant de façon interactive un système d'analyse de risque via de *l'apprentissage actif*. Le système de prédiction surveille ensuite toutes les divulgations de vulnérabilités, et utilise la base de connaissance qu'il a construite via son entrainement avec l'équipe de sécurité pour décider d'un niveau d'alerte approprié pour chaque nouvelle vulnérabilité. Ces niveaux d'alerte sont au nombre de trois et sont centrés sur l'humain : une vulnérabilité peut nécessiter une réponse urgente par un personnel d'astreinte, une réponse non-urgente durant les heures ouvrées, ou ne pas nécessiter de réponse.

Notre protocole expérimental, rendu possible par la participation d'experts en sécurité chargés de défendre de véritables systèmes d'information, met en lumière des difficultés pratiques à évaluer la conception d'un tel système de prédiction, et notamment à entrainer correctement un tel système à l'aide d'une équipe de sécurité disposant de peu de temps. Dans ce contexte, itérer sur de nombreuses hypothèses est difficile. Toutefois, cette évaluation montre également que *l'explicabilité* d'un système de décision automatisé est une composante primordiale de la confiance que lui accorde une équipe de sécurité, et que cette explicabilité est un facteur décisif d'amélioration continue de ces systèmes. Cette contribution est une étape importante vers l'automatisation de l'analyse de risque des vulnérabilités à leur divulgation.

5. Conclusion

Nos contributions nous rapprochent de notre but ultime de permettre aux systèmes d'information d'avoir un niveau de sécurité constant dans le temps. Notre première contribution propose une stratégie complète de défense contre les vulnérabilités n-day, et dans cette thèse nous proposons des solutions pour plusieurs problèmes de recherche ouverts induits par cette stratégie.

Avant nos contributions, il n'y avait à notre connaissance pas de travaux sur la façon

de réagir à de nouvelles vulnérabilités dans les secondes suivant leur divulgation. Notre travail de thèse comble ce manque. Nos seconde et troisième contributions ouvrent la porte à la collecte et analyse automatique des propriétés d'une vulnérabilité à sa divulgation, via des méthodes automatisées et explicables, alors que les travaux précédents se fondaient sur de l'analyse manuelle plus lente.

Notre quatrième contribution vise à permettre à un expert en sécurité d'entraîner un système de défense et d'automatiser le suivi des divulgations de vulnérabilité, retirant le besoin d'avoir une équipe de veille analysant manuellement les nouvelles vulnérabilités 24h sur 24.

Nos travaux ouvrent plusieurs perspectives. La première est le perfectionnement et l'industrialisation de notre quatrième contribution en vue de la création d'un outil de veille automatisée pouvant être utilisé en production par les équipes de sécurité en charge de la protection de systèmes d'information. Une seconde est de cartographier exhaustivement et automatiquement les composants logiciels et matériels d'un système d'information et d'inférer les différentes interactions entre ces composants. Une troisième est la mise en place de nouveaux types de réaction automatisée suite à l'identification d'un risque lié à la divulgation d'une vulnérabilité n-day.

UNIVERSITÉ DE RENNES 1

Titre : Réagir aux vulnérabilités “n-day” dans les systèmes d’information

Mot clés : Informatique, Systèmes informatiques – Mesures de sûreté, Cyberdéfense, Protection de l’information (informatique)

Résumé : Les vulnérabilités n-day sont des vulnérabilités logicielles et matérielles publiques, récemment divulguées, mais pas encore bien connues. Des vulnérabilités n-day passées comme Heartbleed, Shellshock et EternalBlue ont déjà eu un impact important sur des milliers de systèmes d’information et organisations autour du globe. Durant les premiers jours après la divulgation d’une vulnérabilité, les informations préliminaires à propos de celle ci ne sont pas disponibles dans un format lisible automatiquement, ce qui retarde l’usage de processus automatisés de gestion de vulnérabilité. Cette situation créée une période de temps où seule une coûteuse analyse manuelle peut être utilisée pour réagir à une nouvelle vulnérabilité car aucune donnée n’est disponible pour des analyses automatisées moins coûteuses. Nous proposons plusieurs contributions visant d’une part à automatiquement analyser les vulnérabilités récemment divulguées pour prédire leurs propriétés inhérentes (comme le logiciel ou matériel qu’elles affectent, ou leur sévérité), et d’autre part à automatiquement évaluer le risque qu’elles posent à un système d’information donné. Nos contributions prennent place dans une stratégie complète de mitigation du risque posé par les vulnérabilités pour les systèmes d’information.

Title: Reacting to “N-Day” Vulnerabilities in Information Systems

Keywords: Computer Science, Information Systems – Security Measures, Cyberdefense, Information Security

Abstract: N-day vulnerabilities are public, recently disclosed, but not well-known software and hardware vulnerabilities. Past n-day vulnerabilities such as Heartbleed, Shellshock, and EternalBlue already had important real-world impact on thousands of information systems and organizations across the world. One difficulty for proper defense against n-day vulnerabilities is that during the first days after a vulnerability disclosure, preliminary information about the vulnerability is not available in a machine readable format, delaying the use of automated vulnerability management processes. This situation creates a period of time when only expensive manual analysis can be used to react to new vulnerabilities because no data is available for cheaper automated analysis yet. We propose several contributions to automatically analyze newly disclosed vulnerabilities to predict inherent properties such as the software or hardware they affect, and automatically evaluate the risk they pose in the context of a specific information system. We highlight how our contributions take part in a greater end-to-end strategy aiming to mitigate the risk faced by information system because of n-day vulnerabilities.