Model-Based Evaluation of Vulnerabilities in Software Systems

Andy Kenner
METOP GmbH and Otto-von-Guericke University
Magdeburg, Germany
Andy.Kenner@metop.de

ABSTRACT

Vulnerabilities in software systems result from faults, which occur at different stages in a software's life cycle, for example, in the design (i.e., undesired feature-interactions), the development (i.e., buffer overflows), or the operation (i.e., configuration errors). Various databases provide detailed information about vulnerabilities in software systems or the way to exploit it, but face severe limitations. The information is scattered across these databases, fluctuates in quality and granularity, and provides only an insight into a single vulnerability per entry. Even for a single software system it is challenging for any security-related stakeholder to determine the threat level, which consists of all vulnerabilities of the software system and its environment (i.e., operating system). Manual vulnerability management is feasible only to a limited extend if we want to identify all configurations that are affected by vulnerabilities, or determine a system's threat level and the resulting risk we have to deal with. For variant-rich systems, we also have to deal with variability, allowing different stakeholders to understand the threats to their particular setup. To deal with this variability, we propose *vulnerability feature models*, which offer a homogeneous view on all vulnerabilities of a software system. These models and the resulting analyses offer advantages in many disciplines of the vulnerability management process. In this paper, we report the research plan for our project, in which we focus on the model-based evaluation of vulnerabilities. This includes research objectives that take into account the design of vulnerability feature models, their application in the process of vulnerability management, and the impact of evolution, discovery, and verification of vulnerabilities.

CCS CONCEPTS

• **Security and privacy → Vulnerability management**; • **Software and its engineering → Software product lines**.

KEYWORDS

Vulnerability, Exploit, Vulnerability Analysis and Management, Variability Model, Feature Model

SPLC '20, October 19–23, 2020, MONTREAL, QC, Canada

ACM ISBN 978-1-4503-7570-2/20/10…$15.00
https://doi.org/10.1145/3382026.3431246

ACM Reference Format:
Andy Kenner. 2020. Model-Based Evaluation of Vulnerabilities in Software Systems. In *24th ACM International Systems and Software Product Line Conference - Volume B (SPLC '20), October 19–23, 2020, MONTREAL, QC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3382026.3431246

1 INTRODUCTION

The threat level of a software system is determined by its vulnerabilities, which are the main reason for an exploitation [36, 46]. A cyber criminal attacks an existing vulnerability and tries to exploit a software, and in consequence the whole system, to perform unauthorized actions, like information theft or sabotage. Vulnerabilities are identified and published every day by different entities, for example security researchers or software vendors. The same way, new exploits and detailed information about the exploitation become public. As long as a vulnerability is not closed, by patches or reconfiguration, the software remains vulnerable and endangers the entire system or the whole IT infrastructure.

Many databases provide extensive information about vulnerabilities, exploits, and other security issues, for example:

- Common Vulnerabilities and Exposures (CVE),[1]
- National Vulnerability Databases (e.g., the US NVD[2]),
- Exploit Database,[3] and
- rapid7 Vulnerability and Exploit Database[4]

Any of these databases can comprise information about a software's security problems, such as the vulnerability (Which software is exploited?), exploit (How is a vulnerability exploited?), configurations (What version is affected?), and risk and vulnerability management (How to deal with the risk?). So, this information is highly important for any security-related stakeholder that is involved in the software's life cycle. However, this information is scattered across these databases, describes often only a single stakeholder's perspective [48], and fluctuates in quality and granularity [43]. This makes it difficult to determine the threat level of a software system, which is also influenced by many other factors. Already during the analysis of a single software system, we have to consider many dependencies. This includes all vulnerabilities that exist in a particular version of a software as well as in the environment it interacts with (i.e., other software systems or libraries) or is operated in (i.e., the operating system). If we want to identify all affected configurations, especially in variant-rich software system, we are faced with a huge variability within this vulnerability information, for which a manual management is not feasible. To overcome these limitations, we propose an approach that focuses on the acquisition of all existing information about vulnerabilities in a model that is able to capture

[1] https://cve.mitre.org/
[2] https://nvd.nist.gov
[3] https://www.exploit-db.com
[4] https://www.rapid7.com/db

this variability. In software product line engineering, feature models have become established to manage the variability of software systems, wherein features are used to describe certain user-visible aspects, quality or characteristics [3, 10, 25, 42]. We propose a *vulnerability feature model* that provides a homogeneous view of all vulnerabilities in a software system, considering the interactions with its environment. Instead of focusing only on a specific vulnerable version, our model allows us to determine the threat level of all vulnerable configurations, including vulnerabilities that result from their relationships and dependencies. We are not focusing on the security of software product lines (SPLs) [3, 47], but on all software systems that are documented as vulnerable, wherein we make use of SPL techniques and infrastructures to handle the configuration of software landscapes. Our model enables each security-related stakeholder, to analyze and manage the variability of vulnerabilities in an effective way. Depending on a stakeholder's perspective (e.g., assessing the security of the infrastructure), our model is the basis for customized views and analyses that provide additional, novel information within vulnerability management. To this end, we want to address various scenarios, for example to support the vulnerability management of a company, wherein our model is used during the (re)configuration of software systems or landscapes and providing analyses of the threat level, risk and severity as well as alternative suggestions. Especially, in more extensive scenarios, like a service provider of cloud-based services, we have to deal with various system configurations that are tailored to a customer's specific needs (i.e., software systems a particular version). We also plan to investigate the management of evolutionary changes and the discovery of unknown vulnerabilities. These approaches are only made possible by the vulnerability feature model, and are based on the information and dependencies stored in it. Finally, our approach shall help to facilitate the analysis activities of each stakeholder, improve existing analysis results or provide new ones, and thus contribute to improving vulnerability management.

The remaining content of the paper is structured as follows. In Section 2, we introduce relevant terms related to information about vulnerabilities in software systems, and feature models as a representation of variability. We summarize our previous case study and its implications on the research plan in Section 3. In Section 4, we show related work, before concluding this paper in Section 5.

2 BACKGROUND

In this section, we describe software vulnerabilities and exploits, how both are organized in specialized databases, and feature models that we want to use as representation of variability in our project. For better understanding of terms and relationships in the area of vulnerability information, we provide an overview of security concepts and databases in the upper part of Figure 1.

Vulnerability. Mistakes in the life cycle of a software system, for example, in specification, configuration or development, cause weaknesses in the security system, which allow attackers to gain access to a system and manipulate it [36, 46]. This can lead to behavior that has an impact on the security or survivability of a software system [4] and is expressed by access to software itself, the internally stored data or the system beyond.

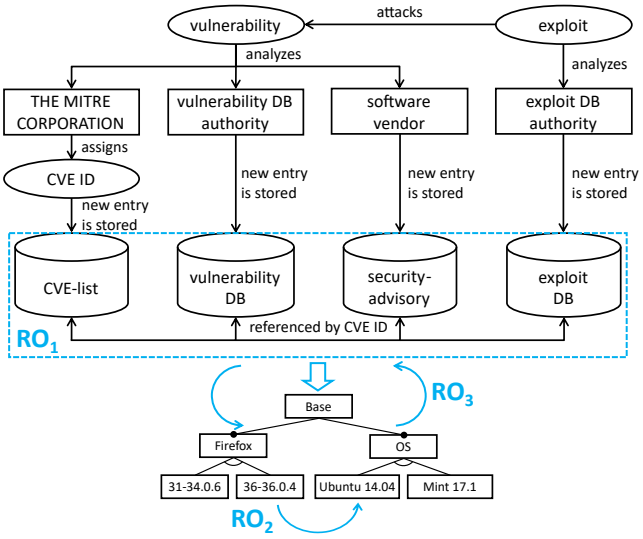


Figure 1: Overview of security concepts, databases and research objectives.

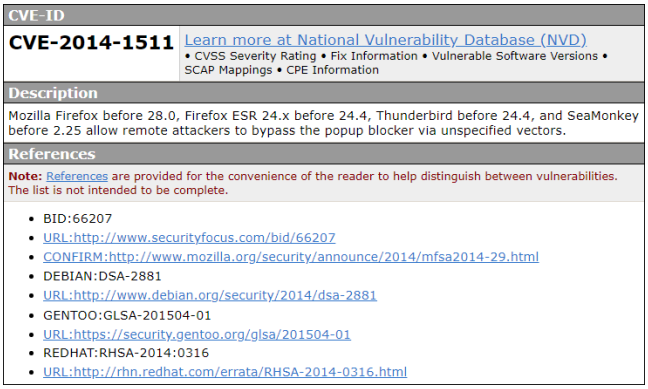
Stolen or cracked login data is usually the first entry point into a corporate network for hackers. This is accomplished through botnet-based remote attacks or malware stored in compromised emails. Afterwards, the execution of malicious code on a local system (e.g., via SQL injections [15], or buffer-overflows [9]) allows an attacker to perform further actions. While this is only a small part of a hacker's portfolio, the presence of exploitable vulnerabilities is critical to the success of a cyber attack.

Exploit. The second important requirement after the vulnerability itself, is the presence of an exploit, of which more and more are identified every day. An exploit is a systematic way to make use of vulnerabilities in a system's architecture [36]. Those exploits attempt to force a break at the vulnerable parts. As our first case study also confirmed, an attacker needs detailed information about the victim system and its environment [26].

CVE. Vulnerabilities and exploits are important elements for the data acquisition and analysis. Very common is a unique identifier, which is used to map related content through databases. CVE has become the default industry standard and is also established in academia—with The MITRE Corporation operating the corresponding information system. All entries together form the CVE-list, a dictionary of publicly disclosed vulnerabilities that actually contains more than 137 thousand[5] entries. After a vulnerability is discovered and analyzed within an additional audit process, The MITRE Corporation creates a new CVE-Identifier (CVE-ID) and assigns it [37] (see Figure 1, left). Figure 2 shows an CVE-entry `CVE-2014-1511`,[6], which we used in our previous case study [26].

Each entry contains the CVE-ID, a brief textual description with facts about the vulnerability, and a reference list with links to data sources with additional information. Such links include various types of websites, such as the developers' security advisory or

[5] manually determined from http://cve.mitre.org (2020-06-17)
[6] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1511



CVE-ID

CVE-2014-1511 Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

Mozilla Firefox before 28.0, Firefox ESR 24.x before 24.4, Thunderbird before 24.4, and SeaMonkey before 2.25 allow remote attackers to bypass the popup blocker via unspecified vectors.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- BID:66207
- URL:http://www.securityfocus.com/bid/66207
- CONFIRM:http://www.mozilla.org/security/announce/2014/mfsa2014-29.html
- DEBIAN:DSA-2881
- URL:http://www.debian.org/security/2014/dsa-2881
- GENTOO:GLSA-201504-01
- URL:https://security.gentoo.org/glsa/201504-01
- REDHAT:RHSA-2014:0316
- URL:http://rhn.redhat.com/errata/RHSA-2014-0316.html

Figure 2: Excerpt of CVE-2014-1511.[6]

bug tracker, confirming the vulnerability, as well as other vulnerability and exploit databases. Operating system vendors, in this case Debian, Gentoo, and RedHat, also analyze vulnerabilities and provide additional information if the vulnerability is part of their distributions and is delivered through them (see Figure 1, right).

Vulnerability Databases. Vulnerabilities are tracked in different databases, such as CVE [1] or NVD [2]. The provided information varies depending on the database. Often, the required software variant, version, and operating system —also with variant and version— is included. In most cases, however, this information is stored in pure textual form (see Figure 2). In addition, metrics, such as the Common Vulnerability Scoring System (CVSS) metrics [37, 49] of the NVD are used to describe and assess characteristics of a vulnerability and help to quantify the severity of potential exploitation.

Exploit Databases. Exploit databases report vulnerabilities in terms of how a vulnerability in a software system can be exploited, providing malicious code or even detailed instructions. This information is also provided in plain text and often has a back reference to a CVE-ID. If the exploit specializes on a single scenario or a group of scenarios, detailed specifications are provided. These specifications indicate, which combinations of a software system and operating system have been successfully exploited.

Exploit databases also contain entries that actually belong to separate data sets with ready-to-use attack scenarios. They are embedded into frameworks that allow the automatic simulation of attack scenarios for a given vulnerability, including the provision of the necessary infrastructure. The MetaSploit Framework (MSF) [35] is the most prominent example where more than 3,800 modules have been assembled. Each of them allows to execute attacks on a software system.

Vulnerability Management. Vulnerability management plays a central role for the security posture of an organization [57] and is defined as cyclical practice of identification, classification, prioritization, remediation, and mitigation of vulnerabilities in a software system or landscape [12].

Variability in Software Systems. Variability is described as the ability of a software system to be tailored for the use in a particular context by its efficient extension, changing, customization or configuration [52]. The management of a software system's commonality and variability has evolved as a central concern in the engineering of software product lines [3, 47]. Variability is described by features of the software system that represent certain user-visible aspects, quality or characteristics and are used to define variants [25].

Variability Model. Variability models, and especially feature models, have become established in academia and industry [3, 10, 25, 42]. Feature models have been introduced in the FODA method [25] and are used to organize the commonalities and variabilities of a software system in an abstract manner, using features to represent different variants of a software product line.

In the basic notation, feature models describe all valid variants of an SPL in a hierarchical, tree-based structure using four relations between parent features and one or more child features and cross-tree constraint for additional dependencies. In addition, many extensions to feature models have been proposed, such as extended feature models [6], which introduce one or more attribute relations to features to find an optimal variant by evaluating those attributes, and non-functional properties [50], which define specific characteristics in addition to functionality and are evaluate during optimization. Extended feature models allow to store additional information to a feature, for example, CVSS metrics.

3 RESEARCH PLAN

A complete analysis of vulnerabilities in software systems to determine the threat level is made difficult by various restrictions. Currently, such analyses usually focus on a certain level, for example, the network layer or all accessible ports and services. Detailed analyzes are not provided and each vulnerability is evaluated on its own, without considering other relationships and dependencies.

However, to effectively deal with vulnerabilities and reduce the risks of cyber attacks, the threat level of a software system and its impacts on the system's security must be fully understood.

3.1 Case Study and First Results

In our previous case study [26], we evaluated detailed information about the characteristics of vulnerabilities, their exploitation and compatible attack scenarios. We extracted this information from various database and used it to virtualize vulnerable systems in Docker containers. The MSF contains a number of attack scenarios as ready-to-use scripts that allowed us to attack these virtualized systems. This way, we evaluated the suitability of the extracted information and analyzed the Firefox architecture under attack.

The case study indicates that all relevant information about software vulnerabilities and exploits can be extracted form existing databases by consequently following links, for example, the CVE-ID. Although we evaluated only 18 attack scenarios for 46 configurations, we had to manually determine all artifacts of information, which required a lot of effort and was error-prone. We evaluated almost all available databases to fully describe a configuration, for example, to determine all affected versions of Ubuntu. Our extraction and analysis showed that further automation and the inclusion of other information sources are necessary.

In our case study, we identified objectives that are subsequently used to define the research plan. These include the design of vulnerability feature models (RO_1), their application in the vulnerability management process (RO_2), and the impact of evolution, discovery, and verification of vulnerabilities (RO_3).

3.2 RO_1: Modeling Vulnerabilities

With regard to vulnerability modeling, we want to design and develop techniques that allow us to synthesize vulnerability feature models, which is the currently planned basis, but we will also investigate alternative model forms. Our first objective is to analyze and extract the variability of software vulnerabilities that is described by many information artifacts scattered across multiple databases (see Figure 1). A comparative survey ($RO_{1.1}$) must be carried out, which expands the knowledge from the case study and structures the information about the available databases, in order to be able to fully define the starting point for the model synthesis ($RO_{1.2}$). Following objectives are addressed:

$RO_{1.1}$: Conduct a Comparative Survey of Vulnerability Databases
$RO_{1.2}$: Develop Techniques to Synthesize Vulnerability Feature Models

$RO_{1.1}$: Conduct a Comparative Survey of Vulnerability Databases. The evaluation of a single database will not be sufficient to fully describe a vulnerability and, moreover, the threat level for a software system. Other databases for vulnerabilities and exploits and any kind of related source (i.e., security advisories, bug tracker) must also be evaluated. We will perform a comprehensive survey of the database landscape, which will help us to determine the significance of each database, and whether we need to consider them in the context of our model synthesis. This will allow us to evaluate the characteristic of each database, the data stored in it, and the dependencies between the databases as well as within the data. In our comparative survey, we address the following topics:

- databases and dependencies
- information composition and structure
- metrics (i.e. CVSS) and enumerations (i.e. CPE)[7]
- verification and history of vulnerabilities
- options for automatized extraction

The resulting comparative survey will provide necessary details about the existing databases and the data stored. According to a first literature search, there are approaches where one or more databases are used as data sources. As far as we know, a few surveys exist that analyze the internal structures of these databases [34, 44, 53]. However, they only focus on a single database, or if multiple databases are analyzed, they are considered separately without evaluating connections or dependencies. The comparative survey is the result of $RO_{1.1}$ and is needed immediately for $RO_{1.2}$.

$RO_{1.2}$: Develop Techniques to Synthesize Vulnerability Feature Models. For the synthesis of vulnerability feature models, all relevant information must be extracted from the selected databases. The comparative survey will give us guidance on which database we have to evaluate first and which characteristics will be relevant. Based on this, we have to develop a technique that is able to:

(1) automatically extract and link vulnerability information,
(2) semi-automatically connect the information and synthesize a vulnerability feature model.

In order to achieve these requirements, we provide advanced analysis techniques to evaluate all necessary information stored as artifacts scattered across databases. In the first step, we follow the comparative survey to build an extraction mechanism. By using explicit information (i.e., CVE-ID, external references), we are able to map all of this information, but we will be faced with many different kinds of information (i.e., natural language descriptions and exploit guidelines) that need to be extracted automatically. The composition of the information is influenced by our other two research objectives (RO_2 and RO_3), since these require an enlarged spectrum of information, in order to be able to carry out, for example, risk ($RO_{2.2}$) or history ($RO_{3.1}$) analyses. This will also influence the structure of the vulnerability feature model and the way in which further information can be stored and evaluated later. Extended feature models [6] or non-functional properties [50] seem to be good opportunities to advance in this direction.

Independently of this decision, the information is connected in the second step and used to synthesize a vulnerability feature model. We assume that the connection step must be carried out semi-automatically, and that we have to provide mechanisms for manual corrections. After completing this work, we can provide a mechanism to generate vulnerability feature models.

> RO_1: Modeling Vulnerabilities
> Evaluate exisiting databases for vulnerabilities in a comparative survey ($RO_{1.1}$). The resulting knowledge is essential for the automatic extraction of vulnerability information in $RO_{1.2}$. Using the extraction and connection mechanisms, we can generate vulnerability feature models semi-automatically.

3.3 RO_2: Vulnerability Management

After we synthesize a vulnerability feature model, various research topics in the handling of vulnerabilities and the resulting threat level will be addressed, expecting many advantages from our model-based information processing (see Figure 1). The interrelated knowledge about vulnerabilities in software systems, as provided by our approach of vulnerability feature model, is of particular interest in vulnerability management which we want to support and improve by the following objectives:

$RO_{2.1}$: Determine the Threat Level of Software Systems
$RO_{2.2}$: Analyze Risk and Severity of Existing Vulnerabilities
$RO_{2.3}$: Optimize the Threat Level Through Alternative Configurations

Each of these objectives can be assigned to one or more disciplines of the general vulnerability management process, as shown below.

$RO_{2.1}$: Determine the Threat Level of Software Systems. We can determine the threat level of a software system or landscape (i.e., the IT infrastructure of an organization) by mapping them to our vulnerability feature model. This concerns the first two steps of the vulnerability management process, namely the identification and classification of the vulnerabilities of a software system. In our approach, the vulnerabilities for a software system are identified by creating a configuration for the vulnerability feature model, which describes the configuration of the system (i.e., underlying operating system and the installed software). The configuration must be precisely defined in order to identify certain version-specific vulnerabilities. The second step is to evaluate the exploitability of the vulnerabilities and classify their severity. With our approach, this information does not have to be determined manually, as the

[7] https://nvd.nist.gov/products/cpe

relevant information is extracted from exploit databases and stored in the vulnerability feature model and linked to the vulnerability.

The information provided in vulnerability feature model enables security-related stakeholder to achieve the remediation and mitigation steps of the vulnerability management process. This way, concrete activities can be selected to secure the software system or landscape or at least protect it from known vulnerabilities. Simple and straightforward activities, such as installing patches, including the operating system and other applications, can be used here. Furthermore, the vulnerability feature model can help to detect vulnerabilities that are caused by dependencies to other software systems used in the same context. Such vulnerabilities are not addressed by a separate consideration of a single software system.

$RO_{2.2}$: Analyze Risk and Severity of Existing Vulnerabilities. Risk and severity analyses are important techniques used in the steps classification and prioritization of the vulnerability management process. A severity analysis helps to understand the level of risk that may be caused by a vulnerability. The risk analysis includes a prioritization by which a sequence for addressing the vulnerabilities in the steps remediation and mitigation is defined. Usually, the CVSS metrics are used to provide an estimation of severity and is also used in the risk assessment.

Following the approach of extended feature models [6], CVSS or any other metric can be stored in the vulnerability feature model as attributes and thus can be evaluated in our approach. Risk estimates and severity analyses can be directly supported by the model, taking into account such metric values for each vulnerability of an analyzed software system. The vulnerability feature model will also provide an expanded, possibly improved, view of risk and severity. Our approach focuses not only on a specific vulnerability, but on the entire threat level of the software system, which is determined by various factors. We can consider every vulnerability for a particular software configuration, including the context, for example, the operating system or a dependent software systems. This way, we develop techniques to determine the overall risk and prevent misjudgments, that occur when considering the risk of vulnerabilities individually. Once the remediation step is completed, the vulnerability feature model can be used to determine the threat level for the updated system. At this point, optimization problems, such as those in feature models with attributes or non-functional properties, should also be examined in order to choose the right strategy for multiple upgrade options and reduce the risk as much as possible. Existing research on extended feature models and non-functional properties seems to be a good opportunity to advance in this direction [5, 16].

$RO_{2.3}$: Optimize the Threat Level Through Alternative Configurations. We offer alternative configurations that support the remediation and mitigation steps in the vulnerability management process to optimize the threat level. These steps determine which update must be performed or which countermeasure is applied if the software system cannot be updated or the risk is not sufficiently reduced after an update.

Our approach reduces the risk from vulnerabilities by providing alternative configurations. We optimize the current configuration of a software system or a landscape and provide a new form of countermeasure in the remediation and mitigation. Such optimizations are very valuable for vulnerability management and offer a promising alternative to approaches, such as encapsulating an unpatchable but necessary system or purchasing special security systems. Alternatives may include changing the operating system (e.g., Linux instead of Windows), replacing or removing parts of the software stack (e.g., HTML5 instead of Adobe Flash), and switching to alternative software systems in the same range of application (e.g., Google Chrome instead of Mozilla Firefox). In research on feature modeling, extension concepts have emerged, such as optimization of configurations, which are the foundations on which we can rely to provide alternative configurations [45, 50]. In industry, there are scenarios that define specific optimization requirements. Some parts of the software stack may be mandatory in a certain version or configuration and may not be updated, switched or reconfigured.

RO_2: Vulnerability Management

Using the vulnerability feature model to determine the threat level ($RO_{2.1}$) will improve the identification and classification steps in the vulnerability management process. Risk and severity analyses ($RO_{2.2}$) can be improved by assessing the threat level for all vulnerabilities of a software and its environment instead of a separate evaluation. Alternative configurations ($RO_{2.3}$) optimize the feature selection in the vulnerability feature model and provide a new form of countermeasurement that helps to reduce the risk by suggesting modifications or replacements to the software stack.

3.4 RO_3: Evolution, Discovery, and Verification

Both, software systems and vulnerability information, evolve over time and will influence our vulnerability feature model, for example, by adding further vulnerable configurations. The evolution may force us to improve the model itself and the mechanisms behind it, but the analysis of this evolution may also help us to discover unknown vulnerabilities, missing configurations, and verify existing ones. This results in the following three objectives, which are discussed separately to examine the implications for vulnerability feature models and analyses techniques (see Figure 1).

$RO_{3.1}$: Manage Evolution of Software Systems and Vulnerability Information
$RO_{3.2}$: Discover Unknown Vulnerabilities
$RO_{3.3}$: Verify Existing Vulnerabilities

$RO_{3.1}$: Manage Evolution of Software Systems and Vulnerability Information. New vulnerabilities or exploits are disclosed with a high frequency and are subject to constant change. This includes editing and correcting existing information or extending it, for example by adding links to other advisories or new types of vulnerable configurations. These changes describe the evolution of a vulnerability over time and affect the vulnerability feature model, techniques based on it, and analyses that have already been performed ($RO_{2.1}$). The CVSS metrics are recalculated based on changes and over time, which affects risk and severity considerations ($RO_{2.2}$) as well as alternative configurations ($RO_{2.3}$).

In order to reflect these changes in the vulnerability feature model, appropriate change tracking and management techniques must be established. This also requires a validation step in which we analyze the nature of the change and determine the associated

impact in advance. Approaches for software product lines must also deal with evolution and offer different solutions [29, 39, 59]. Logic feature mapping or model slices could be options for dealing with evolution. In our approach, we have to consider model changes and the impact on existing techniques and analyses, rather than the evolutionary impact at code level.

$RO_{3.2}$: Discover Unknown Vulnerabilities. Using the feature model as the basic notation allows us to rely on existing tools and analysis mechanisms. Those mechanisms may help us to find inconsistencies while evaluating the logical expression that represents the model or parts of it. Given the information about a software system's vulnerabilities caused by its environmental dependencies (i.e., libraries), we may be able to apply this knowledge to software systems with identical or similar dependencies, possibly also for transitive relationships. Those analyses may point to configurations of our vulnerability feature model that are also vulnerable, even if no information about them is stored in the databases.

The discovery of unknown vulnerabilities would be an added value for vulnerability management, which could be achieved by using our vulnerability feature model. It would also allow us to test the reliability of our approach. An unknown vulnerability would not be integrated directly in our model, but would be processed separately. Additional verification steps must be performed, before this information is passed to a vulnerability authority.

$RO_{3.3}$: Verify Existing Vulnerabilities. Similar to the discovery of unknown vulnerabilities, we also want to support the verification of existing vulnerabilities. Our case study and other research [43] have identified inaccuracies. We will use existing tools and analysis mechanisms for feature models in a specialized form to identify conspicuous patterns or sub-trees that could indicate consistency problems in the databases. If such problems are identified, the information in the databases can be corrected or stored as additional information in the vulnerability feature model to be available in our analyses. Overall, this would improve the data quality and enhance our analyses ($RO_{2.1–2.3}$).

> RO_3: Vulnerability Evolution, Discovery, and Verification
> Software systems and vulnerabilities evolve over time, which is also reflected in the information and its changes. To ensure the consistency of our analyses, all associated impacts must be considered ($RO_{3.1}$). Tools and analysis mechanisms for feature models can use the information about the evolution to discover missing vulnerabilities ($RO_{3.2}$), verify existing vulnerabilities ($RO_{3.3}$), and improve the quality of our model and analyses.

3.5 Unaddressed Related Prospects

In our case study, we identified other perspectives that are not addressed in our project. For the sake of completeness, these approaches are only briefly explained below.

Tailoring Penetration Testing. A vulnerability feature model enables the creation of tailor-made penetration tests for a software system or software landscape that has been hardened according to the threat level. This means that the success of the security hardening can then be examined by attacking the closed vulnerabilities.

Tailoring Vulnerable Victim Systems. The creation of intentionally vulnerable systems, as we have created in our case study, can also be supported. This may range from a simple list or guideline for manual implementation to a fully automated technique, which is entirely conceivable, but also presents challenges in many areas.

4 RELATED WORK

We are aware of related works, but none of them provides a model-based approach for vulnerability management taking all vulnerable configurations and its environmental dependencies into account.

Vulnerability Model, Ontology, and Taxonomy. In the research area of vulnerabilities, models, ontologies or taxonomies are proposed, which are related to our project.
Vulnerability Model. Model-based approaches are used to address the causes and characteristics of vulnerabilities, to detect them on code level [31], to predict them using a linear vulnerability model [2], or for testing [30].
Ontology. In the domain of software vulnerabilities, ontologies are used to formalize the knowledge about concepts, such as vulnerabilities and software systems, and their relationships. The ontology for vulnerability management (OVM) bases on the NVD and its standards (i.e., CWE, CPE), and captures the relationships between IT products, vulnerabilities, attackers, security metrics, countermeasures, and other relevant concepts [57]. Using this ontology, instances can be retrieved and used for vulnerability management or assessment, providing, for example, the vulnerabilities for a single software system or even a specific version. A comprehensive analysis of the threat level does not seem to be possible, so that the vulnerabilities caused by dependencies in the environment cannot be considered simultaneously. This requires a new instance in which, for example, the operating system is examined separately.
Taxonomy. Cyber security taxonomies are used on different levels of abstraction [21, 54] to describe attacks in general or specific forms (i.e., web attacks), using details of vulnerabilities, such as the weakness type (CWE), attack type or its impact, to describe the nature of the attack [18, 28]. There are also taxonomies dealing with specific areas, such as flaws in software systems (i.e., multi-perspective program flaws [55]) or networks (i.e., threats to networks [58]).

Existing approaches for vulnerability models differ completely from our approach and do not deal with capturing all vulnerable configurations or their dependencies on the environment. Similar to our approach, ontologies, such as the OVM, relate different vulnerability information, but a software system can only be analyzed separate without considering any of its dependencies, such as the operating system. Instead of determining the resulting threat level of a software system by analyzing its vulnerabilities, taxonomies classify objects (i.e., cyber attacks) according to specific criteria.

Information Processing from Vulnerability Databases. Various vulnerability databases have been used to analyze and process the information they contain for various purposes.
Prediction. Han et al. [17] and Chen et al. [8] use this information to predict vulnerabilities based on machine-learning. Zhang et al. [60] conduct an empirical study on the NVD to predict zero-day exploits, but conclude that a prediction is impossible.

Reliability. The NVD is evaluated from various perspectives to check the reliability of the stored information, such as version data [43], and the CVSS scoring [22].
Centralization. Kim et al. [27] and Rahman et al. [48] focus on different types of databases to propose a model for a centralized database that contains aggregated vulnerability information and can improve vulnerability management.
(Sub-)Domain-specific Analysis. Databases are analyzed to evaluate the patch behavior of software vendors [41]. Frei et al. [13] perform a comprehensive analysis to examine the life cycle of a vulnerability by focusing on the information in its timeline. Massacci and Nguyen [34] analyze existing databases and propose an new classification scheme that groups information by vendor.

Existing approaches analyze and process only a certain amount of vulnerability information. Non of them focuses on the acquisition of all existing information about vulnerabilities, as our vulnerability feature model does. A combination of information from several databases is proposed, but without supporting or improving the threat level assessment. In contrast to our approach, the information is only examined individually, without considering the relationships and dependencies, such as the environment of the software system.

Vulnerability Assessment. There are solutions that can be used to identify the vulnerabilities of software systems or landscapes. They are limited to a specific domain (i.e., networks infrastructures, code, or web applications) for which vulnerabilities are identified using information from vulnerability databases.
Network Infrastructures. Starting from a network, of, for example, a company or authority, scanners are used to determine the visible attack surface from the network's perspective [33].

For each client, the accessible ports, protocols and hosted services are identified, including platform data for the host's operating system and for the services provided. In OpenVAS[8] or Nessus[9], for example, this information is collected and compared to platform data stored in individual entries of the NVD [19]. This way, all currently existing vulnerabilities for the network infrastructure are identified and made available for further assessment steps.
Code. As part of the test phase during development, methods of static code analysis are used to search for suspicious patterns in the source code of a software in order to identify vulnerabilities [7, 14]. Weaknesses such as buffer overflows, which are often used to crash applications in cyber attacks, occur in certain patterns depending on the programming language, which complicates the static analyses and requires special tools [32, 56]. Tools, such as SonarQube[10] or FindBugs/Spotbugs,[11] are used during development to identify vulnerabilities that may later lead to vulnerabilities when the software is released and used in practice [1].
Web Applications. For web application, Cross Site Scripting (XSS) and various types of injections such as SQL, header or command are the most dangerous attacks [11, 24]. These weaknesses are identified during development by analyzing the code base [23]. Software and penetration testing is used to determine weaknesses software stack on both the server and the user side [20]. Scanners, such as Vega[12] and Wapiti,[13] scan web applications for known security problems.

Existing approaches, solutions, and tools only focus on the analysis of a limited or specific part of the attack surface that a software system can offer. They usually only try to prevent vulnerabilities during development, or work at a certain level of abstraction, for example, systems in a network infrastructure. In our project we want to focus on the entire threat level of a software system or a landscape. Furthermore, we want to consider all affected configurations and dependencies. Although web application scanners analyze the server and installed software components, only those components that are required for the web application are considered.

Security in Software Product Lines. The consideration of security is also highly relevant for SPLs, because even a single vulnerability can cause issues in all derived products. Mellado et al. [38] provide an approach to manage security requirements and the variable artifacts already in the early stages of SPL development. A uniform conceptual framework is necessary to address security concerns in software families with variants of security solutions and different security goals per product [51]. Myllärniemi et al. [40] describe an approach for modeling functional and security variability with security engineering concepts. In our approach we want to analyze the information about vulnerable software systems using techniques of SPL engineering instead of focusing on an SPLs' security.

5 CONCLUSION

Information about vulnerabilities in software systems is scattered as artifacts across different databases and shows fluctuations in quality and granularity. Even the vulnerability analysis of a single software system generates a huge variability space of vulnerabilities, if all affected configurations and dependencies are considered. It is a challenging task for any security-related stakeholder to determine the threat level, as a comprehensive aggregation of information is required. In this paper, we described vulnerability feature models that help to overcome the limitations of current databases and provide a homogeneous view on all vulnerabilities, configurations and dependencies, and allow the management of the resulting variability. We reported on a research plan for our project on the model-based evaluation of vulnerabilities. This includes the definition of objectives, focusing on different research areas related to vulnerability feature models and their application in the vulnerability management process. In the first objective, we want to analyze and extract the variability of vulnerabilities so that we can design and develop a technique to synthesize vulnerability feature models. The second objective comprises the application of our model in the vulnerability management process. To support different disciplines, we evaluate the determination of the threat level of software systems, the analysis of risk and severity of vulnerabilities and optimizations that are used to reduce risk and severity through alternative configurations. In our third objective, we want examine how the evolution affects the model and the analyses based on it, and how this information is used together with the tools and analysis mechanisms available for feature models to detect missing vulnerabilities and verify existing vulnerabilities.

[8] www.OpenVAS.org
[9] www.tenable.com/products/nessus/nessus-professional
[10] www.sonarqube.org
[11] spotbugs.github.io
[12] subgraph.com/vega
[13] wapiti.sourceforge.io

Acknowledgments.. This research is partly supported by the German Federal Ministry of Education and Research (16KIS0526).

REFERENCES

[1] Hamda Hasan AlBreiki and Qusay H. Mahmoud. 2014. Evaluation of Static Analysis Tools for Software Security. In *IIT*. IEEE.

[2] O.H. Alhazmi, Y.K. Malaiya, and I. Ray. 2007. Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems. *Computers & Security* 26, 3 (2007).

[3] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2016. *Feature-Oriented Software Product Lines*. Springer.

[4] William A. Arbaugh, William L. Fithen, and John McHugh. 2000. Windows of Vulnerability: A Case Study Analysis. *Computer* 33, 12 (2000).

[5] Mohsen Asadi, Samaneh Soltani, Dragan Gasevic, Marek Hatala, and Ebrahim Bagheri. 2014. Toward Automated Feature Model Configuration with Optimizing Non-Functional Requirements. *Information and Software Technology* 56, 9 (2014).

[6] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. 2005. Automated Reasoning on Feature Models. In *CAiSE*. Springer.

[7] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. 2010. A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World. *Communications of the ACM* 53, 2 (2010).

[8] Qiuyuan Chen, Lingfeng Bao, Li Li, Xin Xia, and Liang Cai. 2018. Categorizing and Predicting Invalid Vulnerabilities on Common Vulnerabilities and Exposures. In *APSEC*. IEEE.

[9] Crispin Cowan, F. Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. 2000. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. In *DISCEX*. IEEE.

[10] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *VaMoS*. ACM.

[11] Jose Fonseca, Marco Vieira, and Henrique Madeira. 2007. Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks. In *PRDC*. IEEE.

[12] Park Foreman. 2019. *Vulnerability Management*. CRC Press.

[13] Stefan Frei, Martin May, Ulrich Fiedler, and Bernhard Plattner. 2006. Large-Scale Vulnerability Analysis. In *LSAD*. ACM.

[14] Katerina Goseva-Popstojanova and Andrei Perhinschi. 2015. On the Capability of Static Code Analysis to Detect Security Vulnerabilities. *Information and Software Technology* 68 (2015).

[15] William G. Halfond, Jeremy Viegas, and Alessandro Orso. 2006. A Classification of SQL-Injection Attacks and Countermeasures. In *SSSE*. IEEE.

[16] Fatima Zahra Hammani. 2014. Survey of Non-Functional Requirements Modeling and Verification of Software Product Lines. In *RCIS*. IEEE.

[17] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. 2017. Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description. In *ICSME*. IEEE.

[18] Simon Hansman and Ray Hunt. 2005. A Taxonomy of Network and Computer Attacks. *Computers & Security* 24, 1 (2005).

[19] Hannes Holm, Teodor Sommestad, Jonas Almroth, and Mats Persson. 2011. A Quantitative Evaluation of Vulnerability Scanning. *Information Management & Computer Security* 19, 4 (2011).

[20] Yao-Wen Huang, Shih-Kun Huang, Tsung-Po Lin, and Chung-Hung Tsai. 2003. Web Application Security Assessment by Fault Injection and Behavior Monitoring. In *WWW*. ACM.

[21] Vinay Igure and Ronald Williams. 2008. Taxonomies of Attacks and Vulnerabilities in Computer Systems. *IEEE Communications Surveys & Tutorials* 10, 1 (2008).

[22] Pontus Johnson, Robert Lagerstrom, Mathias Ekstedt, and Ulrik Franke. 2018. Can the Common Vulnerability Scoring System Be Trusted? A Bayesian Analysis. *IEEE Transactions on Dependable and Secure Computing* 15, 6 (2018).

[23] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. 2006. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities. In *SP*. ACM.

[24] Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad Jovanovic. 2006. SecuBat: A Web Vulnerability Scanner. In *WWW*. ACM.

[25] Kyo C. Kang, Sholom G Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-021. Carnegie-Mellon University.

[26] Andy Kenner, Stephan Dassow, Christian Lausberger, Jacob Krüger, and Thomas Leich. 2020. Using Variability Modeling to Support Security Evaluations: Virtualizing the Right Attack Scenarios. In *VaMoS*. ACM.

[27] GeonLyang Kim, JinTae Oh, DongIl Seo, and JeongNyeo Kim. 2013. The Design of Vulnerability Management System. *International Journal of Computer Science and Network Security* 13, 4 (2013).

[28] Maria Kjaerland. 2006. A Taxonomy and Comparison of Computer Security Incidents from the Commercial and Government Sectors. *Computers & Security* 25, 7 (2006).

[29] Sebastian Krieter, Reimar Schröter, Thomas Thüm, Wolfram Fenske, and Gunter Saake. 2016. Comparing Algorithms for Efficient Feature-Model Slicing. In *SPLC*. ACM.

[30] Franck Lebeau, Bruno Legeard, Fabien Peureux, and Alexandre Vernotte. 2013. Model-Based Vulnerability Testing for Web Applications. In *ICSTW*. IEEE.

[31] Xiang Li, Jinfu Chen, Zhechao Lin, Lin Zhang, Zibin Wang, Minmin Zhou, and Wanggen Xie. 2018. A Vulnerability Model Construction Method Based on Chemical Abstract Machine. *Wuhan University Journal of Natural Sciences* 23, 2 (2018).

[32] Matti Mantere, Ilkka Uusitalo, and Juha Roning. 2009. Comparison of Static Code Analysis Tools. In *SECURWARE*. IEEE.

[33] Steve Manzuik, André Gold, and Chris Gatford. 2007. *Network Security Assessment: From Vulnerability to Patch*. Syngress.

[34] Fabio Massacci and Viet Hung Nguyen. 2010. Which Is the Right Source for Vulnerability Studies?: An Empirical Analysis on Mozilla Firefox. In *MetriSec*. ACM.

[35] David Maynor. 2011. *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. Elsevier.

[36] Gary McGraw. 2006. *Software Security: Building Security*. Addison-Wesley.

[37] Peter Mell, Karen Scarfone, and Sasha Romanosky. 2006. Common Vulnerability Scoring System. *IEEE Security and Privacy Magazine* 4, 6 (2006).

[38] Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. 2010. Security Requirements Engineering Framework for Software Product Lines. *Information and Software Technology* 52, 10 (2010).

[39] Ralf Mitschke and Michael Eichberg. 2008. Supporting the Evolution of Software Product Lines. In *ECMDA-TW*.

[40] Varvana Myllärniemi, Mikko Raatikainen, and Tomi Männistö. 2015. Representing and Configuring Security Variability in Software Product Lines. In *QoSA*. ACM.

[41] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. 2015. The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching. In *SP*. IEEE.

[42] Damir Nešić, Jacob Krüger, Stefan Stănciulescu, and Thorsten Berger. 2019. Principles of Feature Modeling. In *ESEC/FSE*. ACM.

[43] Viet Hung Nguyen and Fabio Massacci. 2013. The (Un)Reliability of NVD Vulnerable Versions Data: An Empirical Experiment on Google Chrome Vulnerabilities. In *ASIA CCS*. ACM.

[44] Virginia Nunes Leal Franqueira and Maurice van Keulen. 2008. *Analysis of the NIST Database towards the Composition of Vulnerabilities in Attack Scenarios*. Technical Report TR-CTIT-08-08. University of Twente.

[45] Rafael Olaechea, Derek Rayside, Jianmei Guo, and Krzysztof Czarnecki. 2014. Comparison of Exact and Approximate Multi-Objective Optimization for Software Product Lines. In *SPLC*. ACM.

[46] Charles P. Pfleeger and Shari L. Pfleeger. 2002. *Security in Computing*. Prentice Hall.

[47] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer.

[48] Munib ur Rahman, Vikas Deep, and Santosh Multhalli. 2016. Centralized Vulnerability Database for Organization Specific Automated Vulnerabilities Discovery and Supervision. In *RAINS*. IEEE.

[49] Karen Scarfone and Peter Mell. 2009. An Analysis of CVSS Version 2 Vulnerability Scoring. In *ESEM*. IEEE.

[50] Norbert Siegmund. 2019. Challenges and Insights from Optimizing Configurable Software Systems. In *VAMOS*. ACM.

[51] Laurens Sion, Dimitri Van Landuyt, Koen Yskout, and Wouter Joosen. 2016. Towards Systematically Addressing Security Variability in Software Product Lines. In *SPLC*. ACM.

[52] Mikael Svahnberg, Jilles van Gurp, and Jan Bosch. 2005. A Taxonomy of Variability Realization Techniques. *Software: Practice and Experience* 35, 8 (2005).

[53] Tiantian-Tan, Baosheng-Wang, Yong-Tang, Xu-Zhou, and Jingwen-Han. 2019. A Method for Vulnerability Database Quantitative Evaluation. *Computers, Materials & Continua* 61, 3 (2019).

[54] Anshu Tripathi and Umesh Kumar Singh. 2010. Towards Standardization of Vulnerability Taxonomy. In *ICCTD*. IEEE.

[55] K. Tsipenyuk, B. Chess, and G. McGraw. 2005. Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors. *IEEE Security Privacy* 3, 6 (2005).

[56] Andreas Wagner and Johannes Sametinger. 2014. Using the Juliet Test Suite to Compare Static Security Scanners:. In *SECRYPT*. SCITEPRESS.

[57] Ju An Wang and Minzhe Guo. 2009. OVM: An Ontology for Vulnerability Management. In *CSIIRW*. ACM.

[58] D. Welch and S. Lathrop. 2003. Wireless Security Threat Taxonomy. In *SMCSIA*. IEEE.

[59] Jules White, José A. Galindo, Tripti Saxena, Brian Dougherty, David Benavides, and Douglas C. Schmidt. 2014. Evolving Feature Model Configurations in Software Product Lines. *Journal of Systems and Software* 87 (2014).

[60] Su Zhang, Doina Caragea, and Xinming Ou. 2011. An Empirical Study on Using the National Vulnerability Database to Predict Software Vulnerabilities. In *Database and Expert Systems Applications*. Vol. 6860. Springer.