AMADEUS: Towards the AutoMAteD secUrity teSting

Ángel Jesús Varela-Vaca, Rafael M. Gasca, Jose Antonio Carmona-Fombella
María Teresa Gómez-López
Universidad de Sevilla
Seville, Spain
{ajvarela,gasca,maytegomez}@us.es;joscarfom@alum.us.es

ABSTRACT

The proper configuration of systems has become a fundamental factor to avoid cybersecurity risks. Thereby, the analysis of cybersecurity vulnerabilities is a mandatory task, but the number of vulnerabilities and system configurations that can be threatened is extremely high. In this paper, we propose a method that uses software product line techniques to analyse the vulnerable configuration of the systems. We propose a solution, entitled AMADEUS, to enable and support the automatic analysis and testing of cybersecurity vulnerabilities of configuration systems based on feature models. AMADEUS is a holistic solution that is able to automate the analysis of the specific infrastructures in the organisations, the existing vulnerabilities, and the possible configurations extracted from the vulnerability repositories. By using this information, AMADEUS generates automatically the feature models, that are used for reasoning capabilities to extract knowledge, such as to determine attack vectors with certain features. AMADEUS has been validated by demonstrating the capacities of feature models to support the threat scenario, in which a wide variety of vulnerabilities extracted from a real repository are involved. Furthermore, we open the door to new applications where software product line engineering and cybersecurity can be empowered.

CCS CONCEPTS

• **Software and its engineering → Software product lines.**

KEYWORDS

cybersecurity, , feature model,

reasoning, vulnerabilities,

vulnerable configuration,

testing, pentesting

1 INTRODUCTION

The analysis of cybersecurity vulnerabilities is a crucial task to reduce the exposition of end-users and organisations to risks, for instance, Broken Authentication, Code Injection, Insecure References, etc. Nowadays, some of the most important vulnerabilities are the misconfiguration or improper security configuration of systems [35]. Attack vectors are defined as the set of means that enable attackers to exploit system vulnerabilities. In general, the use of default or vulnerable configurations (i.e., misconfiguration) of systems is the first attack vectors used for attackers [28]. To reinforce this, OWASP project [8] establishes the "Security Misconfiguration" and "Using Components with Known Vulnerabilities", as two of the top-10 vulnerabilities for Web systems.

The lack of vulnerability detection in configuration systems before they are exploited might increase the weaknesses of the organisations. To reduce the risks, the cybersecurity community is making an important effort to catalogue vulnerabilities on open-access databases, such as the National Vulnerability Database (NVD) [6] by the National Institute of Standards and Technology (NIST). These databases provide information about the attack vectors to be used for each vulnerability. However, massive are the possible vulnerabilities and configurations that systems might be affected. This variability depends on the wide range of the target systems, such as databases, Web systems, Web repositories, devices, networks, etc.

Although there exist vulnerability analysis tools, e.g., OpenVAS, they focused on finding out vulnerabilities concerning network assets. An in-depth vulnerability analysis of system configurations required to be performed in a manual way based on expert knowledge and developed less formally. This scenario makes unapproachable a systematic analysis of the vulnerabilities of a system, since thousands of features can be involved, with a huge number of configurations that could or not produce a vulnerable system object of a later attack. This is where feature models can help to model the variability of the vulnerable configurations.

Feature Models (FMs), in the context of Software Product Lines (SPLs), are used to represent and reason about the possible configurations of the systems represented in a compact way. Variability models, such as FMs [21], describe commonalities and variabilities in SPLs and are used along with all the SPL development processes.

The extremely high possible combinations of the features involved in the configuration and how they could produce a vulnerability (i.e., attack vector) bring about the proper context where FMs and their advantages can be applied. Nevertheless, the problem is how these FMs could be created, that implies to solve different challenges as the following: (i) the management of the various set of sources of vulnerabilities [24]; (ii) the extraction of the high number

of features involved in vulnerabilities; (iii) inferring the relation between the features analysing the vulnerabilities; and, (iv) facilitating the security testing according to the inferred knowledge.

To facilitate the automatic analysis and testing of cybersecurity vulnerabilities, in this paper, we propose AMADEUS, a framework to guide the creation and reasoning over the feature models extracted from the analysis of the organisation infrastructure and the vulnerabilities retrieved by the vulnerability repositories. The contributions of the proposal are focused on: (1) definition and implementation of the necessary phases to manage the vulnerable configurations with FMs; (2) integration with the existing analysis tools for the extraction of the relevant information from the systems; (3) integration with external vulnerability repositories, catalogues, and databases; (4) scrapping the external repositories to obtain the affected vulnerabilities of the analysed systems, and extract the information related to the vulnerable configurations; (5) inferring FMs automatically according to the identified vulnerabilities and vulnerable configurations; (6) reasoning over the FMs to facilitate the generation of attack vectors for security testing; and, (7) evaluating AMADEUS in a scenario where services and applications of a real system with at least 20 vulnerabilities and thousands of vulnerable configurations are analysed.

The remainder of this paper is organised as follows: Section 2 introduces the needed concepts about vulnerabilities in cybersecurity to understand the proposal; Section 3 describes the proposal, detailing the integrated modules necessary to achieve the objectives and the steps of the methodology; Section 4 details how to formalise FMs from extracted information; Section 5 studies how the created FM can be used for reasoning. Section 6 evaluates the proposal with a real repository; Section 7 analyses the previous related work; and, finally, the paper is concluded, opening some evolution of the proposal for future work.

2 FOUNDATIONS

In order to understand the proposal, it is necessary to introduce some terms related to the vulnerabilities in cybersecurity, i.e., the existing repositories, the structure of the known vulnerabilities, and the scenarios where they might occur.

2.1 Repositories of Vulnerabilities

Derived from the high number of vulnerabilities that can affect a system, there exist catalogues, repositories, and databases. Some examples are the NVD [6], the US-CERT Vulnerability Notes Database [10], the IBM's XFORCE [4], and the VulDB [9]. In general, these databases provide information about the attack vectors to be used in each vulnerability. In this paper, we focused on NVD which contains information related to vulnerabilities and cybersecurity-related issues, it is widely used and contains a trusted source of information since it is maintained up to date. Moreover, it provides a web tool that allows to search vulnerabilities.

Due to the wide range of the target systems (e.g., databases, Web systems, Web repositories, devices, etc.), massive are the possible vulnerabilities and configurations that systems might be affected.

Vuln. ID	Summary	CVSS Severity
CVE-2020-1933	A XSS vulnerability was found in ...	*V3.0*: 6.1 *V2.0*: 4.3
CVE-2020-1928	An information disclosure vulnerability was ...	*V3.0*: 5.3 *V2.0*: 5.0

Table 1: NVD results for "*Apache NiFi 1.10*" query.

2.2 Vulnerabilities

To automate the analysis of vulnerabilities, the cybersecurity community has made several efforts to make uniform how the vulnerabilities are represented. Common Vulnerabilities and Exposures (CVEs) [2] is the de-facto standard used by NVD and other repositories to represent the vulnerabilities. CVE is a reference method to publish the known vulnerabilities in a structural way to facilitate its management and sharing. A CVE is formed of a list of vulnerabilities that includes, among other, the following information: (i) the CVE ID, the identifier of the vulnerabilities, that is mandatory information; (ii) the summary to describe the vulnerability textually; (iii) the impact of the vulnerability, following the standard CVSS (Common Vulnerability Score System) [3], for assessing the severity of the vulnerability. CVSS in each of its different versions (until the current 3.1) proposes a formula that returns a value between 0 and 10, to represent the lowest and highest severity respectively. Table 1 is an example of two CVEs related to Apache NiFi 1.10.

2.3 Vulnerable Configurations

The CVE vulnerabilities are formed of a set of CPEs (Common Platform Enumeration), $\{cpe_1, cpe_2, \cdots, cpe_n\}$, that describe the products and scenarios in which a vulnerability might occur, represented through a set of Known Affected Software Configurations (hereinafter *Configurations*). The CPE standard [1] by MITRE is used to formalise these possible configurations. CPE defines a standardised approach for the identification of the features of the contexts where the vulnerabilities might be exploited. They provide key information in the definition, enforcement and verification of IT policies, such as vulnerability or configurations.

CPEs are represented by using a modular naming specification known as Well-Formed Name (WFN), introduced in the CPE 2.3 version [1]. WFN permits to represent abstract logical *conceptual* expressions, that unambiguously specify desired implementations and behaviours. WFN follows the structure of a list of pairs: attribute (a) and value (v).

Definition 2.1. Let $\langle a, v \rangle$ be a pair of an attribute a, with a value v, where WFNs can be expressed as an unordered set of n valid pairs, that describe the characteristics of a *cpe*.

A *cpe* is valid when each attribute (a_i) appears only once, and belongs to the following list:

- *part* describes the scope of applicability: hardware (h), software (a), or operating system (o).
- *vendor* describes the organisation that distributes the product, e.g., *apache*.
- *product* identifies the product affected, e.g., *nifi*.
- *version* is a vendor-specific alphanumeric string that characterises the release version of the product, e.g., 1.0.1.
- *update* is a specific alphanumeric string that characterises the update version of the product affected, e.g., update 256.

- *edition* captures edition-related terms applied by the vendor to the product.
- *language* defines the language supported by the product, e.g., ES.
- *sw_edition* describes how the product is tailored to a particular market.
- *target_sw* defines the software environment within the product operates, e.g., windows.
- *target_hw* characterises the architecture, e.g., x86.
- *other* describes any other information.

Regarding to their values (v_i), they are commonly UTF-8 strings, but they can also be assigned two logical values: *ANY*, there is no restrictions applicable for the attribute; and, *NA* (Not Applicable), there is no valid value for the attribute. Thereby, a CPE which follows WFN can be represented as follows:

$$cpe_x = \{\langle part, v_1\rangle, \langle vendor, v_2\rangle, \langle product, v_3\rangle \ldots, \langle other, v_n\rangle\} \quad (1)$$

The reference cpe_x is used just as a manner to easily identify and differentiate the CPEs between them. WFNs are not intended to be a data format, encoding, or any other kind of machine-readable representation. To deterministically transform a logical construct into a machine-readable representation, there exist the so-called bindings. The one this article uses receives the name of *Formatted String Binding* (FSB), and consists of a colon-delimited list of attributes[1] as follows:

$$cpe : 2.3 : part : vendor : product : version : update : edition : language : sw\_edition : target\_sw : target\_hw : other \quad (2)$$

FSB adds prefixes and binds the attributes in a WFN in a fixed order and separated by the colon character. Note that all eleven attribute values must appear in the FSB, such as:

$$cpe : 2.3 : o : linux : linux\_kernel : 2.6.0 : * : * : * : * : * : * : * : * \quad (3)$$

The previous example in WFN format, for the CPE 2.3 version, can be represent as following:

$$\{\langle part, o\rangle, \langle vendor, linux\rangle, \langle product, linux\_kernel\rangle, \langle version, 2.6.0\rangle, \langle update, ANY\rangle, \langle edition, ANY\rangle, \langle language, ANY\rangle, \langle sw\_edition, ANY\rangle, \langle target\_sw, ANY\rangle, \langle target\_hw, ANY\rangle, \langle other, ANY\rangle\} \quad (4)$$

Analysing each attribute, the list describes a vulnerable configuration in an Operating System (*part=o*), released by Linux (*vendor*), named Linux Kernel (*product*) and at version 2.6.0 (*version*). The rest of the attributes take the wildcard value (*) in FSB, which is how the logical value *ANY* in WFN. Remark, we ignored the first pair (cpe:2.3) since it only describes the CPE format used.

Regarding NVD, the CVEs that represent vulnerabilities are formed of a set of vulnerable contexts, so-called *Configurations*. A *Configuration* is composed, in turn, of a list of vulnerable CPEs, $\{cpe_1, cpe_2, cpe_3, \ldots, cpe_n\}$, and optionally, a set of *Running Configurations* (RC) can be included as a set of CPEs $\{cpe_{n+1}, cpe_{n+2}, cpe_{n+3}, \ldots, cpe_{n+m}\}$ specifying the concrete executions environments in which the vulnerability may be reproduced. In presence of RCs, we have to take into account the combinations of the CPEs regarding each RC separately. Table 2 is an example of configurations for the 'CVE-2020-1933' vulnerability associated to the Cross-Site Scripting in the Apache NiFi for versions 1.0.0 to 1.10.0. In this example, there is only one RC, thereby, cpe_1 can occur with cpe_{58}; cpe_2 can occur with cpe_{58}; so until cover all the combinations.

[1]The first pair indicates the standard of CPE version used.

Configuration 1
List of CPEs
cpe_1 : *cpe:2.3:a:apache:nifi:1.0.0:beta-rc1:*:*:*:*:*:**
cpe_2 : *cpe:2.3:a:apache:nifi:1.0.0:rc1:*:*:*:*:*:**
cpe_3 : *cpe:2.3:a:apache:nifi:1.0.0:-:*:*:*:*:*:**
... (+54 results)
Running Configurations
cpe_{58} : *cpe:2.3:a:mozilla:firefox:-:*:*:*:*:*:*:**

Table 2: List of CPEs for the vulnerability *CVE-2020-1933* from NVD.

3 AMADEUS

The deep analysis of the possible security vulnerabilities based on the configuration of the systems can facilitate the detection of possible attack vectors [34, 41]. AMADEUS aims to get attack vectors automatically as the baseline for conducting a complete and congruent security test for a specific environment. Therefore, AMADEUS is conceived as a framework to discover and analyse vulnerable configurations within IT-resources of an ecosystem, based on the process shown in Figure 1. AMADEUS can be positioned in the reconnaissance phase and the enumeration in an ethical hacking process [20]. Thus, the framework collects information relative to the known vulnerabilities stored in public vulnerability databases, to create automatically the attack vectors derived from these vulnerabilities to detect scenarios where the vulnerabilities might occur. In the AMADEUS process the activities have been marked as manual by a hand symbol, as automatic by an engine symbol, and semi-automatic by an engine-hand symbol. The manual activities and semi-automatic require certain human intervention. Each step is described in the following subsections.

3.1 Analysis of Infrastructure

To extract the vulnerabilities of a system, it is necessary to analyse the features that are configured on it. This is why AMADEUS starts analysing a group of systems and/or devices (cf., Analyse Infrastructure or Provide Terms) that pinpoints some of their key characteristics.

The identification of the particular attack vectors or the definition of pentesting techniques to be applied [14, 38] depends on the configuration of the systems (e.g., software, hardware, network, and people). To gather and recovery the configuration used in organisations, different solutions exists, both active and passive analysis tools. One of them is the Configuration Management Database (CMDB) [31, 40, 48], an IT infrastructure database that stores the configurations of their assets, so-called configuration items. Concerning cybersecurity, CMDBs are frequently used to perform impact analysis and the seek of the root cause under a failure. On the other hand, active tools, such as Lynis [5] and Nmap (Network Mapper) [7], let to audit systems to detect vulnerabilities, executing penetration testing, of performing system hardening. Nmap is a very well-known tool widely used for auditing the security of firewalls, networks, measuring the traffic of a network or the detection

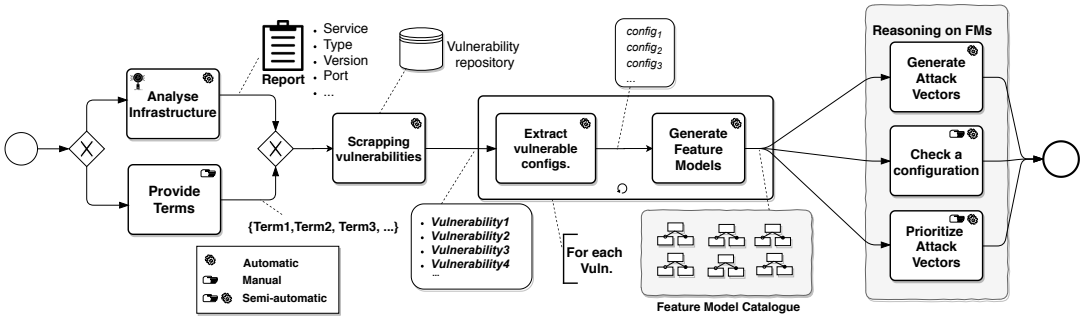

Figure 1: AMADEUS Framework overview.

Service	Version
Mozilla, Firefox	-
OpenSSH	7.7
Apache HTTP Server, OpenSSL	-
Adobe Flash	32.0.0.238
OpenVPN	2.3.17

Table 3: Example of extracted terms.

of vulnerabilities. Due to the popularity in the security community, Nmap is the solution integrated into the implementation of AMADEUS, albeit others could be adapted as well.

AMADEUS[2] has been developed with two operation modes: custom and automatic. In the custom mode, AMADEUS allows users to provide a list of terms and keywords for a set of target systems. In the automatic mode, AMADEUS invokes an analysis tool (Nmap in our case) with a set of target systems. The information retrieved from this tool is tackled by AMADEUS as a list of terms and keywords, where tuples as $\langle service, version \rangle$ are returned, as shown in Table 3.

3.2 Scrapping Vulnerabilities

The terms related to the running services, versions, active ports, etc. found in the reports obtained from the Analyse infrastructure task or provided by the users, are essential in the search of related vulnerabilities. This quest can be then performed against a vulnerability database, where a scrapper (cf., Scrapping Vulnerabilities) can be used to extract the vulnerabilities.

AMADEUS integrates the NVD [6] solution. Due to the intrinsic web nature of NVD repository, the best approach to automatise the search and extraction of such information are employing a scrapper, reason why AMADEUS integrates a web scraper module. This scrapper analyses the HTML structure similar to the shown in Table 1, and replicates the results by retaining only specific and relevant information, such as the CVE ID.

3.3 Extraction of the Vulnerable Configurations

Afterwards, each of the extracted vulnerabilities might be used for ascertaining the vulnerable configurations (cf., Extract vulnerable configs.)

[2]https://github.com/IDEA-Research-Group/AMADEUS

Once the vulnerabilities represented by the CVEs are gathered, the possible features of the scenarios where these vulnerabilities can be exploited are analysed. To do that, it is necessary to ascertain how these scenarios are described. For example, the vulnerability 'CVE-2020-1933' describes that malicious scripts can be injected in Apache NiFi 1.10. However, several questions arise, such as: *in which specific software configuration is this vulnerability applicable*; *whether it might be related to software, hardware, application or for an Operating System*, and; *whether this vulnerability might exist for every version or release.*

In this stage, AMADEUS, by using the same scrapper, extracts the different sets of vulnerable configurations (represented by CPEs) for every vulnerability (represented by CVEs) that were discovered in the previous step.

3.4 Generation of Feature Models and Reasoning

With these items as a basis, AMADEUS attempts to infer/build some valid feature models (cf., Generate FMs) of the possible configurations that may lead to an attack, generating a feature model catalogue of vulnerabilities [44]. In this paper, we propose an algorithm to create feature models adapted to the vulnerability context. These feature models can be used to create a catalogue of scenarios where attacks can occurs for the known vulnerabilities. This catalogue can be used in an immense variety of test generation scenarios reasoning about them (cf., Reasoning on FMs). Figure 1 shows some of the utilities that can be applied over these aggregation of models, ranging from the generation of attack vectors to the detection of vulnerable components amongst others.

The details about how the feature models are created and the possible reasoning are found in Sections 4 and 5.

4 GENERATION OF FEATURE MODELS OF VULNERABILITIES

As commented, the high variability defined by the number of possible vulnerabilities and the vulnerable configurations makes it extremely complex to manage the potential threats. The creation of FMs that covers this vulnerability information could facilitate the vulnerability analysis.

A FM combines a set of relevant features and their dependencies. There exist various notations to design FMs [15], although the most widely used is that proposed by Czarnecki [16], that describes the set of features and the constraints (relations) between them.

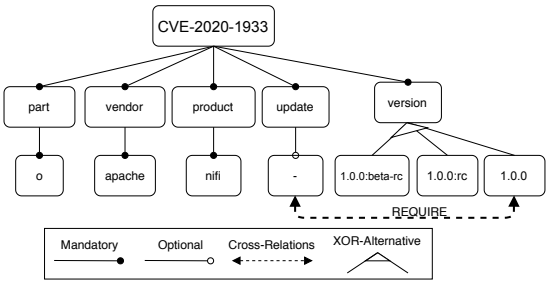

Figure 2: Example of FM for the *CVE-2020-1933*.

Bringing FMs to the problem of vulnerability analysis at hand, we would need to represent every vulnerability (CVE) and its vulnerable configurations (CPEs) in a FM. To do that, it is necessary to determine the features and the relations among them. Following with the example of Table 2, a quick first approximation might be to use the attributes of the CPEs as the features with a mandatory relation with their values, as shown in Figure 2. This is a simplified FM since it only includes three of the 58 CPEs for this vulnerability. The problem arises when several and different vendors, products, versions, etc; with different values are involved in the same vulnerability. How to generalise the creation of FMs to enable an automatic and uniform construction is one of the purposes of AMADEUS.

AMADEUS creates a FM for each CVE (vulnerability), including every CPE of their configurations. Therefore, every configuration produced by the FM is a vulnerable configuration according to the NVD vulnerabilities.

Definition 4.1. Let *CPEs* be a list of vulnerable configurations and running configuration environments $\{cpe_1, cpe_2, cpe_3, \ldots, cpe_n\}$, a FM is equivalent to a list of CPEs, iff the set of products of FM is equal to the CPEs described in each vulnerability.

$$FM \equiv CPEs \iff products(FM) = \{cpe_1, cpe_2, cpe_3, \ldots, cpe_n\} \quad (5)$$

In our approach, the construction of the so-mentioned FM is divided into two main stages: (1) Retrieve an unrestricted FM that contains only information regarding every $\langle attribute, value \rangle$ pair within the set of CPEs; and, (2) include restrictions into the form of cross-tree relations to the previous FM to make it equivalent to the CPEs, avoiding possible configurations that could be produced by the unrestricted FM.

4.1 Retrieving Unrestricted Feature Model from CPEs

The well-known as reverse engineering in SPLs [29, 30, 39] provides mechanisms to generate FMs according to a set of configurations. In the context of cybersecurity and vulnerable configurations, the reverse engineering applied is relatively bounded, since only 12 attributes may occur, and some of them cannot take every configuration of values. This is the case of the attribute *product*, that determines the *vendor* and the *part*. It means that the same *product* cannot belong to two different vendors or two different parts. Moreover, these three attributes must be mandatorily fulfilled, being impossible an 'ANY' label value associated with them. This is the main motivation to propose a specific algorithm to incorporate these restrictions in the generation of the FM. To exemplify each part of the algorithm, the running example of Table 4 is used. The

Configuration 1
List of CPEs
cpe_1 : cpe:2.3:a:olearni:civet:1.0.0:*:*:fr:*:*:*
cpe_2 : cpe:2.3:a:olearni:civet:1.0.1:*:*:*:*:*:*:*
cpe_3 : cpe:2.3:a:olearni:civet:1.0.2:*:*:*:*:*:*:*
Running Configurations
-
Configuration 2
List of CPEs
cpe_4 : cpe:2.3:a:oteachy:lynx:*:*:*:es:*:*:*:*
cpe_5 : cpe:2.3:a:oteachy:ocelot:*:*:*:*:*:*:*:*
Running Configurations
cpe_6 : cpe:2.3:a:origin:iberian:-:*:*:*:*:*:*:*

Table 4: Running example of CPEs for a vulnerability.

example represents a vulnerability encompassed of two configurations (cf., Configuration 1 and 2), each with a list of CPEs and a list of running configuration, empty for the *Configuration 1*.

The idea of the algorithm to create the FM (cf., Algorithm 1) is based on three parts: (1) the creation of a sub-FM for each vendor; (2) the creation of sub-FM for the running configurations; and, (3) the integration of these sub-FMs in a single FM tree. Figure 3 represents the different steps for the running example of Table 4, but for a better understanding of each part of the algorithm, some concepts must be introduced.

Let L be the list of n configurations (CPEs), of a given CVE. L could be regarded as a composition of two smaller lists, L_{VUL}, $\{cpe_1, cpe_2, cpe_3, \ldots, cpe_j\}$; and, L_{RC}, $\{cpe_{j+1}, cpe_{j2}, \ldots, cpe_n\}$, containing vulnerable configurations and execution environments, respectively. Regarding the running example in Table 4, $L_{VUL} = \{cpe_1, cpe_2, cpe_3, cpe_4, cpe_5\}$ and $L_{RC} = \{cpe_6\}$.

Derived from the mentioned special characteristics of the CPE attributes: *product, vendor, and part*, some functions are described as following:

- $getVendors(L)$ returns the vendors associated to the list L of CPEs. For the example, *getVendors(L) = {'oteachy', 'olearni', 'origin'}*.
- $getProducts(L, v_i)$ returns a list of the products for a vendor v_i for a given list L of CPEs. For the example, *getProducts(L, 'oteachy') = {'lynx', 'ocelot'}*.
- $getAttributes(L, p_i)$ returns a list of attributes that are relevant for the product p_i because of they do not have '*' in every CPE of L. For the example, *getAttributes(L, 'civet')={'version', 'language'}*.
- $getValues(L, p_i, a_j)$ returns a list of values for the attribute a_i for a product p_j in a list L of CPEs. For the example, *getValues(L, 'civet', 'version')={'1.0.0', '1.0.1', '1.0.2'}*.

Other operators have been defined for developing the algorithms. The operators are grouped into two categories:

(1) Operators to get information from L:
   - $vul(l)$ takes a list of CPEs (l) as input and returns the list of vulnerable configurations, L_{VUL}. For the example, *vul(L)={cpe1, cpe2, cpe3, cpe4, cpe5}*.
   - $rc(l)$ takes a list of CPEs (l) as input and returns the map (list of pairs key→value) of the running environment configurations, L_{RC}. For the example, *rc(L)=['rc_1' → {cpe6}]*.

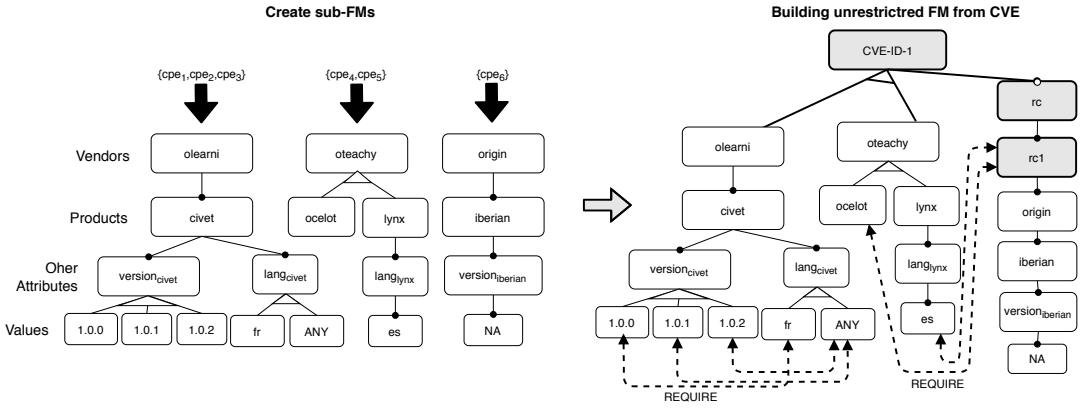

Figure 3: Process of construction of the FM for the running example.

- $getRC(L, rc_i)$ returns a list of CPEs associated to the rc_i in the running configurations L. For the example, *getValues*(L_{RC}, 'rc_1')={cpe6}.

(2) Operators to build FM structures:

- $createRootF(FM, n)$ creates a new feature in the FM named n and establishes it as root.
- $man(FM, f_1, f_2)$ creates two new features, if they do not exist, and a mandatory relation between them.
- $opt(FM, f_1, f_2)$ creates two new features, if they do not exist, and an optional relation between them.
- $xor(FM, f, A)$ creates a new feature f in FM, if it does not exist, and an XOR-Alternative relation between it and the set of alternative features $A \subset FM$.
- $children(FM, f, C)$creates a new feature f in FM, if it does not exist, and a relation with a set of children features $C \subset FM$:
  - If $|C| = 1$, a new mandatory relation is added between f and $c \in C$; i.e., $man(FM, f, c)$.
  - If $|C| > 1$, a new XOR-Alternative relation is added between r and $\forall c \in C$; i.e., $xor(FM, f, C)$.
- $merge(FM, f, S)$ creates a new feature f in FM, if it does not exist, and a relation with set S of FMs. Let R be the set of roots $\forall FM_i \in S$, the operator *merge* creates a new relation between f and every $root_j \in R$; i.e., $children(FM, f, R)$.

The concrete specification of the solution is given in Algorithm 1. The algorithm is derived into three parts: (1) **initialisation of the lists (lines 1-2),** to prepare the needed structures to create the list of sub-FMs of the vulnerable configuration according to the vendors ($listOfFM_{VUL}$) and running configurations ($listOfFM_{RC}$) according to L_{RC}; (2) **creation of the sub-FMs for each vendor and their integration in a single FM (lines 4-10),** where the root is 'CVE-ID' and the branches are the sub-FMs obtained from *createSubFMs*, detailed in Algorithm 2; and, (3) **creation of a FM of the running configuration (lines 11-22)**, with a root 'rc' as an optional relation with the whole FM (since running configuration can or cannot appear).

In the right part of Figure 3, we can see how the three sub-FMs, of the left part, are combined to create the complete FM for the example.

Algorithm 1: Build unrestricted FM from a CVE.

Input: CVE-ID, $L : \{cpe_1, cpe_2, cpe_3, \ldots, cpe_n\}$
Result: fm: Feature Model

1 $listOfFM_{VUL} \leftarrow \{\}$; $listOfFM_{RC} \leftarrow \{\}$; $fm \leftarrow \{\}$;
2 $L_{VUL} = vul(L)$; $L_{RC} = rc(L)$;
3 /* Create the root of FM */
4 $createRootF(fm$, CVE-ID);
5 /* Create FMs for the list of CPEs */
6 $listOfFM_{VUL} \leftarrow createSubFMs(L_{VUL})$;
7 /* Create a FM for each Vendor */
8 **for** $fm_{vul_i} \in listOfFM_{VUL}$ **do**
9 $\quad merge(fm$, CVE-ID, $fm_{vul_i})$;
10 **end**
11 **if** $|L_{RC}| > 0$ **then**
12 $\quad$ /* Create a node that will contain all RCs */
13 $\quad opt(fm$, CVE-ID, "rc");
14 $\quad$ /* Create a FM for each RC */
15 $\quad$ **for** $rc_i \in L_{RC}$ **do**
16 $\quad\quad listOfFM_{VUL} \leftarrow createSubFMs(getRC(L_{RC}, rc_i))$;
17 $\quad\quad$ /* Merge FMs together */
18 $\quad\quad$ **for** $fm_{rc_i} \in listOfFM_{RC}$ **do**
19 $\quad\quad\quad merge(fm, rc_i, fm_{rc_i})$;
20 $\quad\quad$ **end**
21 $\quad$ **end**
22 **end**

Algorithm 2 is responsible of creating a list of FMs (*createSubFMs*), one for each **vendor** (line 3) with the vendor as root (line 6). Iteratively, the **products** of each vendor (line 7), and the **attributes** of each product (line 8). The possible **attribute values** are also included and related in the model **(lines 9-10)**. Finally, the **relations** between features of **products and attributes (line 12)**, and **products and vendors (line 14)** are included.

4.2 Include cross-tree constraints to the FM

Up to this point, the FM obtained encapsulates every attribute and values for the CPEs of a CVE. As aforementioned, the set of CPEs does not usually include such high variability, and the existence of certain of its components is intrinsically related to the appearance

Algorithm 2: Create sub-FMs.

Input: $L : \{cpe_1, cpe_2, cpe_3, \ldots, cpe_n\}$
Result: $listOfFM$: List of FMs

```
1  listOfFM ← {};
2  /* Create new FM representing each vendor */
3  for v_i ∈ getVendors(L) do
4      fm ← {};
5      /* Include all vendors as root feature */
6      createRootF(fm, v_i);
7      for p_j ∈ getProducts(L, v_i) do
8          for a_k ∈ getAttributes(L, p_j) do
9              /* Create the features and the relation between them,
                 representing the values a_k that the attributes may
                 take */
10             children(fm, a_k, getValues(L, a_k, p_j));
11         end
12         children(fm, p_j, getAttributes(L, p_j));
13     end
14     children(fm, v_i, getProducts(L, v_i));
15     listOfFM ← fm;
16 end
```

of others. That is the reason why the inference of a set of constraints over the FM is necessary, to overcome this situation and restrict the number of feasible combinations, adjusting it. In this stage, AMADEUS derives a set of constraints using Algorithm 3 to adjust the variability of the FM according to the restriction of the CPEs attributes and the running configurations.

The cross-tree constraints are derived from an analysis performed over the original list of CPEs, which is a clear descriptor of the possible valid configurations. Any other combinations will generate a configuration that is not included in the list, hence considered as spurious. We may recall that the whole aim of this algorithm is to build a FM that can produce the same set of items contained in the original list of CPEs. Therefore, we only use two types of cross-tree constraints (i.e., the *Require* and the *XOR-require* [26, 37]) modelled by means of a single operator (cf., explained bellow *const(FM,f,C)*). The *Require* constraint is used when the feature requires the existence of other features with non-direct family relation (e.g., $f1 \rightarrow f2$). On the other hand, the *XOR-require* constraint ($\oplus$) involves a list with a minimum of three features. *XOR-require* constraint establishes a required relation between one feature and the set of other features, permitting that only one appears at the same time. A constraint $f1 \oplus \{f2, f3\}$ is equivalent to:

$$((f1 \rightarrow f2) \wedge \neg(f1 \rightarrow f3)) \vee (\neg(f1 \rightarrow f2) \wedge (f1 \rightarrow f3)) \quad (6)$$

A set of operators are introduced to facilitate the understanding of Algorithm 3:

- $getLeaves(FM)$ takes a feature model FM, and returns the set of leaves of the model FM, that are the values of the relevant attributes. For the example, $getLeaves(FM)$ = {'1.0.0', '1.0.1', '1.0.2', *'fr'*, *'ANY'*, *'es'*, *'NA'*}.
- $isRC(L, f)$ takes a list of CPEs L, and a value f which represents a feature, that returns a *true* value if f belongs to any running configuration of L, *false* otherwise. For the example, $isRC(L, `fr') = false$ or $isRC(L, `NA') = true$.
- $getSiblings(L, f)$ takes a list of CPEs L, a value f which represents a feature, and returns a list of values that are sibling of it and belongs to the same product in L. For the example, $getSiblings(L, `1.0.0')$ = '$fr$' or $getSiblings(L, `1.0.1')$ = 'ANY' or $getSiblings(L, `es')$ = {}.
- $getRelatedRC(L, f)$ takes a list of CPEs L, a value f which represents a feature, and returns a list of values that represent the related running configurations in L. For the example, $getRelatedRC(L, `1.0.0')$ = {} or $getSiblings(L, `ocelot')$ = '$rc1$' or $getSiblings(L, `es')$ = {'$rc1$'}.
- $const(FM, f, C)$ takes a source feature $f \in FM$ and a set of target features $C \subset FM$:
  - If $|C| = 1$, a new *Require constraint* relation is added between f and $c \in C$.
  - If $|C| > 1$, a new *XOR-require* constraint is added between f and $\forall c \in C$.

Algorithm 3 is encompassed of the two following parts: (1) Creation of cross-tree constraints between feature leaves of the same product (between values of the relevant attributes of the same sub-FM) (lines 6-11); and, (2) Creation of cross-tree constraints between feature leaves of products and running configurations (between values of the relevant attributes and a root of sub-FM of a running configuration) (lines 12-17).

Considering again the list of CPEs for the running example in Table 4 and the generated FM of Figure 3, it is possible to find out several required cross-tree constraints. The cases are: (1) between the attributes of the *civet* product, the required between '1.0.0' and 'fr' features to enforce the achievement of the $cpe1$; (2) the two required between '1.0.1' and '1.0.2', and 'ANY' features to enforce the $cpe2$ and $cpe3$; (3) the required relation between '*ocelot*' and '$rc1$' features to enforce the occurring of the running configuration features for the $cpe4$; and, (4) the required relation between '*es*' feature and the '$rc1$', to enforce $cpe5$. These five constraints are included to complete the FM.

5 REASONING ON THE FEATURE MODELS

The use of FM provides a way to represent the vulnerabilities structurally, dispelling the generic and meaningless structures of conventional representations, such as lists or tables. An additional advantage derived from the use of FM, is the storage of the vulnerabilities as a catalogue. AMADEUS is empowered thanks to the definition of a catalogue of FMs that could, in some manner, be regarded as an interactive entity that supports a wide range of queries and reasoning. The catalogue stores the same information as the databases of vulnerabilities, as NVD, but additionally, it unleashes a range of valuable utilisation of features for the analysis of vulnerabilities, that are:

1. Generate attack vectors: Every FM holds information about vulnerable configurations, by obtaining the set of all products of the model, we are able to generate the attack vectors. For instance, for the running example the products are: *Product* 1: {CVE-ID-1, olearni, civet, $version_{civet}$, 1.0.0, $lang_{civet}$, fr}; *Product* 2: {CVE-ID-1, olearni, civet, $version_{civet}$, 1.0.1, $lang_{civet}$, ANY}; *Product* 3: {CVE-ID-1, olearni, civet, $version_{civet}$, 1.0.2, $lang_{civet}$, ANY}; *Product* 4: {CVE-ID-1, otecachy, ocelot, rc, rc1, origin, iberian, $version_{iberian}$, NA};

Algorithm 3: Create Cross-tree Constraints for FM.

Input: $L : \{cpe_1, cpe_2, cpe_3, \ldots, cpe_n\}$, FM: Feature Model
Result: FM : Feature Model with Constraints

```
/* Obtain the leaves of the FM */
leaves ← getLeaves(FM);
/* For each leaf */
for leaf ∈ leaves do
    if ¬isRC(L, leaf) then
        /* Get other leaves related to the same CPE */
        listSiblings ← getSiblings(L, leaf);
        /* Include a new cross-tree for each relative leaf */
        for s_i ∈ listSiblingsAttr do
            const(FM, leaf, s_i);
        end
        /* Get RC related to the leaf */
        listRelatedRC ← getRelatedRC(L, leaf);
        /* Include a new cross-tree for each relative RC */
        for rc_i ∈ listRelatedRC do
            const(FM, leaf, rc_i);
        end
    end
end
```

Product 5: {CVE-ID-1, oteachy, lynx, $lang_{lynx}$, es, rc, rc1, origin, iberian, $version_{iberian}$, NA}.

2. Obtain/Check a specific configuration: Given details about a specific configuration, AMADEUS is able to determine whether it is vulnerable or not, by diagnosing the configuration concerning the FMs. Furthermore, it can pinpoint which products would make a valid vulnerable configuration. For instance, we would like to diagnose the configuration: {*olearni*,*lynx*,*es*} for the running example. The diagnosis of this configuration can return that to deselect the feature *olearni*, and to select features {*CVE-ID-1*, $oteachy$, $lang_{lynx}$}.

3. Obtain a prioritised set of attack vectors: A reordering/ prioritisation using a specific criteria [45] could be applied to the generated attack vectors. For instance, we would like to prioritise the attack vectors with Spanish language (es): 1) *Product 1:* {CVE-ID-1, oteachy, lynx, $lang_{lynx}$, es, rc, rc1, origin, iberian, $version_{iberian}$, NA}; 2) *Product 2:* {CVE-ID-1, olearni, civet, $version_{civet}$, 1.0.1, $lang_{civet}$, ANY}; 3) *Product 3:* {CVE-ID-1, olearni, civet, $version_{civet}$, 1.0.2, $lang_{civet}$, ANY}; 4) *Product 4:* {CVE-ID-1, olearni, civet, $lang_{civet}$, fr, $version_{civet}$, 1.0.0, }; 5) *Product 5:* {CVE-ID-1, otecachy, ocelot, rc, rc1, origin, iberian, $version_{iberian}$, NA}.

The mentioned types of analysis are ground-breaking proposals in the combination of cybersecurity and software product lines. And they are just a few examples of the potential use of FMs in the field of cybersecurity, leaving for the further work in the matter the inclusion of much more functionalities.

6 EVALUATION

To evaluate the feasibility of AMADEUS, we propose the answering of the following research questions: **RQ1.** Can we automatically extract vulnerable configurations from real scenarios? **RQ2.** Can we infer automatically FMs in an acceptable time? **RQ3.** Can we reason on the FMs extracted to determine, i.e., generating attack vectors?

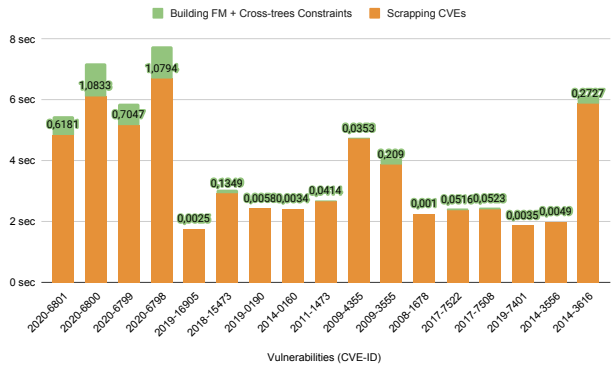

Figure 4: Time consumed in the whole process (Scrapping, Building FM, and Cross-tree Constrains).

To conduct **RQ1**, we have used one synthetic threat scenario which considers a workstation in a corporate network, which includes the following applications and services: *Firefox*[3] as Internet browsers; *Adobe Flash*[4] plugins for those browsers; *OpenSSH* server to allow external connections; *Apache HTTP server* with an *OpenSSL* as SSL/TLS provider to support secure connections; Nginx as alternative of Web server; finally, OpenVPN as client/server enables to secure external connections.

AMADEUS has automatically determined certain terms after scanning the workstation with the Nmap tool, see the bold ones in the column Terms of Table 5. The rest of the terms have been included manually, i.e., OpenSSL. Furthermore, Table 5 shows the information retrieved from the NVD by using the terms, the CVE identification, the number of CPEs, and the Running Configurations (RC). The CVEs have been automatically extracted by AMADEUS from the NVD. Although only some CVEs are shown, since the complete list is formed of thousands of elements, e.g., *Mozilla Firefox* has more than 2,000 vulnerabilities (CVEs). Nevertheless, the chosen CVEs cover 5.000 vulnerable configurations (CPEs) approximately, giving an idea of the complexity of the scenario to answer RQ1.

For each CVE, AMADEUS automatically infers a FM (**RQ2**) as explained in Section 4 by using FaMa [17]. The inferred FM models[5] for the evaluation and the source code of AMADEUS implementation are free available[6]. To analysis the key characteristics of the FM, we provide information about the number of features, the number of relations (mandatory, optional, and XOR), and the number of cross-tree constraints.

To evaluate the performance on inferring the FMs we analyse each phase presented in Figure 1 (i.e., Scrapping Vulnerabilities, Extract CPEs and Generate FM). The designed benchmark consists of the execution of each phase several times and the calculation of the average time (in seconds) spent on each one. The benchmark wants to demonstrate (**RQ2**) that the generation of FMs requires

[3]Firefox vulnerabilities: https://www.cvedetails.com/product/3264/Mozilla-Firefox.html?vendor_id=452
[4]List of Adobe Flash vulnerabilities: https://www.cvedetails.com/product/6761/Adobe-Flash-Player.html?vendor_id=53
[5]Due to the limitation of the basic FaMa formalisation to support certain cross-tree constraints (i.e., XOR cross-relation), we generate the cross-tree constraint apart from FM file in a generic formulation that can be easily adapted to any formalism.
[6]https://github.com/IDEA-Research-Group/AMADEUS

Terms	CVE Id.	#CPEs	#RCs	#Features	#Mandatory	#Optional	#XOR	#Constraints	Valid	# Attack Vectors
Mozilla, Firefox	CVE-2020-6801	442	0	446	5	0	3	422	✓	442
	CVE-2020-6800	978	0	849	8	0	6	790	✓	978
	CVE-2020-6799	581	1	585	9	1	6	552	✓	581
	CVE-2020-6798	978	0	849	8	0	6	790	✓	978
OpenSSH, 7.7	CVE-2019-16905	8	0	11	4	0	2	0	✓	8
	CVE-2018-15473	259	3	172	29	1	8	108	✓	259
Adobe Flash, 32.0.0.238	CVE-2019-8070	157	24	133	15	1	11	90	✓	1975
	CVE-2019-8069	157	24	133	15	1	11	90	✓	1975
Apache HTTP Server, OpenSSL	CVE-2019-0190	21	0	34	10	0	3	9	✓	21
	CVE-2014-0160	14	0	17	4	0	2	7	✓	14
	CVE-2011-1473	86	0	71	4	0	2	57	✓	86
	CVE-2009-4355	169	0	68	5	0	4	48	✓	169
	CVE-2009-3555	396	0	406	20	0	9	173	✓	396
	CVE-2008-1678	3	0	7	3	0	1	0	✓	3
OpenVPN, 2.3.17	CVE-2017-7522	102	0	99	4	0	2	63	✓	102
	CVE-2017-7508	102	0	99	4	0	2	63	✓	102
nginx, 1.7	CVE-2019-7401	13	0	17	3	0	1	0	✓	13
	CVE-2014-3556	19	0	23	3	0	1	0	✓	19
	CVE-2014-3616	279	0	283	3	0	1	0	✓	279

Table 5: Analysis of FMs generated by AMADEUS.

acceptable time (sub-linear time) in the general case. The evaluation times are shown in Figure 4 and 5. As can be observed in Figure 4, the consumed time is mostly used for the scrapping, in comparison with the time to build the FMs (labelled in the figure because sometimes is very small to be distinguished). It is relevant to bear in mind that the scrapping is quite affected by the Internet time response of the NVD repository and the size of the CVE to be extracted. Focusing on generation FMs, Figure 5 represents the percentage of consumed time for the creation of the unrestricted FMs and the creation of the cross-tree constraints, that correspond with Algorithm 1 and 3 respectively. The results demonstrate that in the CVEs with more number of features the consumed time for the cross-time constraints is higher.

To conduct **RQ3**, the FMs have been analysed in twofold: (1) *validating the model* and calculating *the number of products* (cf., # Attack vectors) operation; and, (2) reasoning about the attack vectors and checking configurations.

The valid operation demonstrates that is correct in terms of obtaining at least one valid product. That is, from the perspective of security testing the FM will create at least one valid attack vector. The number of products operation represents the number of valid attack vectors that can be generated from the FM. The number of products can be used as validation operation, in this respect, in our case, the number of products helps as a correctness metric to measure precision and recall [29]. Thus, if the number of products differs from the expected combination of CPEs and RCs, the FM is not equivalent (cf., Definition 4.1), thereby, invalid. For instance, the CVE-2020-6801 has 442 CPEs and 0 RCs hence the expected number of attack vectors is 442 as we can check in Table 5. In the presence of RCs, we have to bear in mind the number of combinations of the CPEs regarding each RC separately, such as the case of *Adobe Flash*. For instance, for the vulnerability CVE-2019-8070 the number of attack vectors is 1975, as the result of: $33*3+55*4+47*24+22*24 = 1975$. The first term $(33 * 3)$ means that there are 33 CPEs with 3 possible RCs which implies that we must consider all possible combinations of CPE and RC.

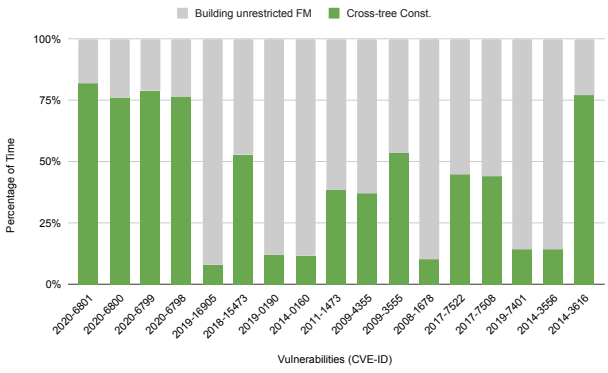

Figure 5: Time consumed in the Building unrestricted FMs and Adding Cross-tree constraints.

Regarding the reasoning capabilities, they can be used from two perspectives: (1) users who want to analyse and manage the security on the system configurations; and, (2) security testers who want to check and test the security of the systems. As aforementioned, we could generate a set of attack vectors for the service Apache HTTP Server with OpenSSL. We can use AMADEUS by introducing the terms, and the attack vectors can be obtained, for instance, from the CVE-2019-0190 (cf., Table 5). This CVE enables that remote attackers send a request that would cause mod_ssl to enter a loop leading to a denial of service. We can generate a sample of attack vectors by filtering some features (e.g., apache_http_server and openssl): (1) {apache, apache_http_server, version, 2.4.37}; (2) {openssl, openssl, version, 1.1.1, update, pre7}; (3) {openssl, openssl, version, 1.1.1, update, pre5}; (4) {openssl, openssl, version, 1.1.1b, update, ANY}; and, (5) {openssl, openssl, version, a.00.09.07l, update, ANY}. This is only one example of an attack vector, others can be calculated applying different filters. This example provides valuable information to security testers about the scenario to be checked and better planning the pentesting, focusing on the vulnerabilities and specific configurations.

7 RELATED WORK

The analysis of vulnerabilities of systems is a well-known problem to manage the system risks [25, 42]. To reduce risks, the vulnerabilities must be gathered and analysed to ascertain the possible attacks. About the gathering of vulnerabilities, some approaches to identify and extract them can be found in the literature. In [36] the authors

provided a tool, IVA, to automate the process of finding possible vulnerabilities in software products installed inside an organisation. This approach depends on an inventory of assets, but our approach is decoupled of the infrastructure since AMADEUS supports analysis tools such as Nmap [7] which enables us to discover assets and services automatically without an inventory.

Pseudo-formal structures, such as software requirement specifications written in Structured Object-oriented Formal Language (SOFL), have been used to identify vulnerabilities [19]. Other approaches [46] used Natural Language and Machine Learning (ML) techniques to extract useful information from vulnerability databases, as NVD [6], that can subsequently be utilised by applications, such as vulnerability scanners and security monitoring tools. In [32], a framework is presented to detect and extract information about vulnerabilities and attacks from Web text.

Once the vulnerabilities are identified, various are the techniques to analyse the possible risks [12]. In [32], an ontology of terms is created for identifying future vulnerability terms for querying the NVD. In [24], ML techniques are used in a cybersecurity knowledge base to extract entities and build an ontology to obtain a cybersecurity knowledge base. New rules are then deduced by calculating formulas and using the path-ranking algorithm. The use of the knowledge base implies to maintain this structure up-to-date in case of new terminology appears, and not to analyse in deep the set of the vulnerabilities of a system, providing less customised solutions. However, our approach is focused on the analysis of the vulnerabilities of a system according to its components, configurations or the terms introduced by the experts.

The use of FMs is introduced as a means to manage the variability of vulnerability and configuration of systems in a reasoning manner. After a FM is defined, products can be configured, derived or validated. In the configuration and derivation process, the users select and deselect features using a configurator. FMs have been demonstrated very useful in the security field where they are applied to model the domain of configuration of security systems [44] and for the selection of configurations to reduce the security risk in the selection of configurations [50]. The idea of using FMs to represent the vulnerabilities of systems already existed in the literature [27]. The authors analyse the vulnerabilities to create synthetic attack scenarios. However, the main methodology used to derive these models remains manual. In contrast with this, our approach provides a new automatic method able to outperform current human-oriented methods. We define a consistent and homogeneous structure by using FMs for the representation of vulnerable configurations, but AMADEUS also provides a solution that includes every phase from the vulnerability extraction to the FMs creation and reasoning.

The extraction of the FMs from existing systems is a topic already tackled in SPL by reverse engineering techniques. The reverse engineering techniques are used in many directions but basically to determine features, feature constrains, or generating complete feature models. An approach [49] is presented to extracting complex feature correlations from existing product configurations using association mining techniques. In [33] a reverse engineering technique is proposed to automatically build up language product lines from exiting DSL variants. [13] provides an approach to mine a feature model based on the formal analysis of conceptual models and configurations. SPL-XFactor [43] is an end-to-end search-based framework for reverse engineering feature models.

Several are the techniques applied to do reverse engineering in SPL: search-based techniques [29]; using propositional logic [18]; natural language requirements [47]; ad-hoc algorithms [11, 22, 23]; and, configuration scripts [39].

Most of the reverse engineering approaches are focused on the application of different topics of software engineering, but they are away from the special characteristics of cybersecurity and vulnerability issues. The reason why the extraction of FMs from vulnerabilities has been considered in this paper.

8 CONCLUDING REMARKS & FUTURE DIRECTIONS

The management and testing of cybersecurity vulnerabilities have become crucial for organisations to avoid security risks. Vulnerabilities databases have emerged as the cornerstone in vulnerability management and security testing. However, the massive information joined with the huge complexity and variability of the system configuration makes very complex to provide efficient solutions. In this paper, AMADEUS is presented as a holistic solution to reduce the gap between security testing and vulnerability management in the organisations. It is performed by enabling the vulnerability scanning of the systems and deriving FMs to provide reasoning capabilities to security testers. FMs models are useful for reasoning about attack vectors for the security testers according to the scenario.

Even though the experiments presented in this paper provide pieces of evidence for validation, we discuss the different threats to validity that affect the approach: (1) *Internal validity*, the analysis done in the evaluation reveals different properties of the vulnerabilities and vulnerable configurations, however, there might be characteristics that are not revealed, e.g., the most prominent vulnerable feature. (2) *External validity*, although the evaluation covers a large number of vulnerable configurations we cannot generalise the conclusions about the precision of the FMs because an extended study is needed to avoid some bias. AMADEUS is useful for different security stakeholders to reveal reasoning capabilities on vulnerability information that currently are not exploited. (3) *Conclusion validity*, anybody can replicate the experiments since we provide a repository with AMADEUS and the models extracted.

As future work directions, AMADEUS has many potential extensions. AMADEUS can be extended with prioritising techniques for the generation of attack vectors analysing the FMs and the threat scenario, and the detection of inconsistencies in the repositories of vulnerabilities. Implementation can be extended integrating other analysis tools (e.g., Lynis); the integration of other vulnerability databases (e.g., US-Cert); and, the integration with security testing tools (e.g., Wapiti).

ACKNOWLEDGMENTS

This work has been partially funded by ECLIPSE (RTI2018-094283-B-C33), COPERNICA, and METAMORFOSIS projects; and by ERDF/FEDER Fund. Special thanks to David Benavides and José A. Galindo for their support in this work.

REFERENCES

[1] 2020. Common Platform Enumeration. Available from MITRE. https://cpe.mitre.org/

[2] 2020. Common Vulnerability Exposure. Available from MITRE. http://cve.mitre.org/

[3] 2020. Common Vulnerability Scoring System SIG. Available from FIRST. https://www.first.org/cvss/

[4] 2020. Internet security systems x-force security threats. Available from IBM. https://exchange.xforce.ibmcloud.com/

[5] 2020. Lynis Audit Tool. Available from CISOFY. https://cisofy.com/lynis/

[6] 2020. National Vulnerability Database. Available from NIST. https://nvd.nist.gov/

[7] 2020. NMAP The Network Mapper. Available from NMAP. https://nmap.org/

[8] 2020. OWASP Top Ten. Available from OWASP. https://owasp.org/www-project-top-ten/

[9] 2020. The CommunityDriven Vulnerability Database. Available from VULDB. https://vuldb.com/

[10] 2020. Vulnerability notes database. Available from US-CERT. https://www.kb.cert.org/vuls/

[11] M Acher, A Cleve, G Perrouin, P Heymans, C Vanbeneden, P Collet, and P.c Lahire. 2012. On extracting feature models from product descriptions. In *VAMOS*. 45–54. https://doi.org/10.1145/2110147.2110153

[12] Adedayo Oyelakin Adetoye, Sadie Creese, and Michael Goldsmith. 2012. Reasoning about Vulnerabilities in Dependent Information Infrastructures: A Cyber Range Experiment. In *Critical Information Infrastructures Security - 7th International Workshop, CRITIS 2012, Lillehammer, Norway, September 17-18, 2012, Revised Selected Papers (Lecture Notes in Computer Science)*, Bernhard M. Hämmerli, Nils Kalstad Svendsen, and Javier López (Eds.), Vol. 7722. Springer, 155–167. https://doi.org/10.1007/978-3-642-41485-5_14

[13] Ra'Fat Al-Msie'deen, Marianne Huchard, Abdelhak Seriai, Christelle Urtado, and Sylvain Vauttier. 2014. Reverse Engineering Feature Models from Software Configurations using Formal Concept Analysis. In *Proceedings of the Eleventh International Conference on Concept Lattices and Their Applications, Košice, Slovakia, October 7-10, 2014 (CEUR Workshop Proceedings)*, Karell Bertet and Sebastian Rudolph (Eds.), Vol. 1252. CEUR-WS.org, 95–106. http://ceur-ws.org/Vol-1252/cla2014_submission_13.pdf

[14] Michael Backes, Jörg Hoffmann, Robert Künnemann, Patrick Speicher, and Marcel Steinmetz. 2017. Simulated Penetration Testing and Mitigation Analysis. *CoRR* abs/1705.05088 (2017). arXiv:1705.05088 http://arxiv.org/abs/1705.05088

[15] Don Batory. 2005. Feature models, grammars, and propositional formulas. In *International Conference on Software Product Lines*. Springer, 7–20.

[16] David Benavides, Sergio Segura, and Antonio Ruiz Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.* 35, 6 (2010), 615–636. https://doi.org/10.1016/j.is.2010.01.001

[17] David Benavides, Pablo Trinidad, Antonio Ruiz Cortés, and Sergio Segura. 2013. *FaMa*. Springer Berlin Heidelberg, Chapter FaMa, 163–171. https://doi.org/10.1007/978-3-642-36583-6-11

[18] Krzysztof Czarnecki and Andrzej Wasowski. 2007. Feature diagrams and logics: There and back again. In *11th International Software Product Line Conference (SPLC 2007)*. IEEE, 23–34.

[19] B. O. Emeka and S. Liu. 2018. Assessing and extracting software security vulnerabilities in SOFL formal specifications. In *2018 International Conference on Electronics, Information, and Communication (ICEIC)*. 1–4.

[20] Patrick Engebretson. 2013. *The basics of hacking and penetration testing: ethical hacking and penetration testing made easy*. Elsevier.

[21] José A. Galindo, David Benavides, Pablo Trinidad, Antonio-Manuel Gutiérrez-Fernández, and Antonio Ruiz-Cortés. 2018. Automated analysis of feature models: Quo vadis? *Computing* (11 Aug 2018). https://doi.org/10.1007/s00607-018-0646-1

[22] Evelyn Nicole Haslinger, Roberto E Lopez-Herrejon, and Alexander Egyed. 2011. Reverse engineering feature models from programs' feature sets. In *2011 18th Working Conference on Reverse Engineering*. IEEE, 308–312.

[23] Evelyn Nicole Haslinger, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2013. On extracting feature models from sets of valid feature combinations. In *FASE*. Springer, 53–67. https://doi.org/10.1007/978-3-642-37057-1_5

[24] Yan Jia, Yulu Qi, Huaijun Shang, Rong Jiang, and Aiping Li. 2018. A Practical Approach to Constructing a Knowledge Graph for Cybersecurity. *Engineering* 4, 1 (2018), 53 – 60. https://doi.org/10.1016/j.eng.2018.01.004 Cybersecurity.

[25] Matthieu Jimenez, Renaud Rwemalika, Mike Papadakis, Federica Sarro, Yves Le Traon, and Mark Harman. 2019. The importance of accounting for real-world labelling when predicting software vulnerabilities. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 695–705. https://doi.org/10.1145/3338906.3338941

[26] Ahmet Serkan Karataş, Halit Oğuztüzün, and Ali Doğru. 2013. From extended feature models to constraint logic programming. *Science of Computer Programming* 78, 12 (2013), 2295 – 2312. https://doi.org/10.1016/j.scico.2012.06.004 Special Section on International Software Product Line Conference 2010 and Fundamentals of Software Engineering (selected papers of FSEN 2011).

[27] 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 '20: 14th International Working Conference on Variability Modelling of Software-Intensive Systems, Magdeburg Germany, February 5-7, 2020*. 10:1–10:9. https://doi.org/10.1145/3377024.3377026

[28] Xiaowei Li and Yuan Xue. 2014. A Survey on Server-Side Approaches to Securing Web Applications. *ACM Comput. Surv.* 46, 4, Article Article 54 (March 2014), 29 pages. https://doi.org/10.1145/2541315

[29] R.E Lopez-Herrejon, L Linsbauer, J.A Galindo, J.A Parejo, D Benavides, S Segura, and A.a Egyed. 2015. An assessment of search-based techniques for reverse engineering feature models. *JSS* 103 (2015), 353–369. https://doi.org/10.1016/j.jss.2014.10.037

[30] Roberto Erick Lopez-Herrejon, José A. Galindo, David Benavides, Sergio Segura, and Alexander Egyed. 2012. Reverse Engineering Feature Models with Evolutionary Algorithms: An Exploratory Study. In *Search Based Software Engineering*, Gordon Fraser and Jerffeson Teixeira de Souza (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 168–182.

[31] Hari Madduri, Shepherd S. B. Shi, Ron Baker, Naga Ayachitula, Laura Shwartz, Maheswaran Surendra, Carole Corley, Messaoud Benantar, and Sushma Patel. 2007. A configuration management database architecture in support of IBM Service Management. *IBM Syst. J.* 46, 3 (2007), 441–458. https://doi.org/10.1147/sj.463.0441

[32] V. Mulwad, W. Li, A. Joshi, T. Finin, and K. Viswanathan. 2011. Extracting Information about Security Vulnerabilities from Web Text. In *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, Vol. 3. 257–260.

[33] David Méndez-Acuña, José A. Galindo, Benoît Combemale, Arnaud Blouin, and Benoît Baudry. 2017. Reverse engineering language product lines from existing DSL variants. *Journal of Systems and Software* 133 (2017), 145 – 158. https://doi.org/10.1016/j.jss.2017.05.042

[34] Alex Oyler and Hossein Saiedian. 2016. Security in automotive telematics: a survey of threats and risk mitigation strategies to counter the existing and emerging attack vectors. *Security and Communication Networks* 9, 17 (2016), 4330–4340. https://doi.org/10.1002/sec.1610

[35] Salvador Martínez Perez, Valerio Cosentino, and Jordi Cabot. 2017. Model-based analysis of Java EE web security misconfigurations. *Comput. Lang. Syst. Struct.* 49 (2017), 36–61. https://doi.org/10.1016/j.cl.2017.02.001

[36] Luis Alberto Benthin Sanguino and Rafael Uetz. 2017. Software Vulnerability Analysis Using CPE and CVE. arXiv:cs.CR/1705.05347

[37] Christoph Seidl, Tim Winkelmann, and Ina Schaefer. 2016. A software product line of feature modeling notations and cross-tree constraint languages. In *Modellierung 2016*, Andreas Oberweis and Ralf Reussner (Eds.). Gesellschaft für Informatik e.V., Bonn, 157–172.

[38] Sugandh Shah and Babu M. Mehtre. 2015. An overview of vulnerability assessment and penetration testing techniques. *J. Comput. Virol. Hacking Tech.* 11, 1 (2015), 27–49. https://doi.org/10.1007/s11416-014-0231-x

[39] S She, R Lotufo, T Berger, A Wa¸sowski, and K.a Czarnecki. 2011. Reverse engineering feature models, In ICSE. *ICSE*, 461–470. https://doi.org/10.1145/1985793.1985856

[40] Edgar H. Sibley, P. Gerard Scallan, and Eric K. Clemons. 1981. The software configuration management database. In *American Federation of Information Processing Societies: 1981 National Computer Conference, 4-7 May 1981, Chicago, Illinois, USA (AFIPS Conference Proceedings)*, Vol. 50. AFIPS Press, 249–255. https://doi.org/10.1145/1500412.1500448

[41] Florian Skopik, Roman Fiedler, and Otmar Lendl. 2014. Cyber Attack Information Sharing. *Datenschutz und Datensicherheit* 38, 4 (2014), 251–256. https://doi.org/10.1007/s11623-014-0101-1

[42] Pierantonia Sterlini, Fabio Massacci, Natalia Kadenko, Tobias Fiebig, and Michel van Eeten. 2020. Governance Challenges for European Cybersecurity Policies: Stakeholder Views. *IEEE Secur. Priv.* 18, 1 (2020), 46–54. https://doi.org/10.1109/MSEC.2019.2945309

[43] Thammasak Thianniwet. 2016. SPL-XFactor: A framework for reverse engineering feature models. (01 2016).

[44] Ángel Jesús Varela-Vaca, Rafael M. Gasca, Rafael Ceballos, María Teresa Gómez-López, and Pedro Bernáldez Torres. 2019. CyberSPL: A Framework for the Verification of Cybersecurity Policy Compliance of System Configurations Using Software Product Lines. *Applied Sciences* 9, 24 (2019). https://doi.org/10.3390/app9245364

[45] Shuai Wang, David Buchmann, Shaukat Ali, Arnaud Gotlieb, Dipesh Pradhan, and Marius Liaaen. 2014. Multi-Objective Test Prioritization in Software Product Line Testing: An Industrial Case Study. In *Proceedings of the 18th International Software Product Line Conference - Volume 1 (SPLC '14)*. Association for Computing Machinery, New York, NY, USA, 32–41. https://doi.org/10.1145/2648511.2648515

[46] Sachini S. Weerawardhana, Subhojeet Mukherjee, Indrajit Ray, and Adele E. Howe. 2014. Automated Extraction of Vulnerability Information for Home Computer Security. In *FPS*.

[47] Nathan Weston, Ruzanna Chitchyan, and Awais Rashid. 2009. A framework for constructing semantically composable feature models from natural language requirements. In *Proceedings of the 13th International Software Product Line Conference*. 211–220.

[48] Hiroshi Yamada, Takeshi Yada, and Hiroto Nomura. 2013. Developing network configuration management database system and its application - data federation for network management. *Telecommunication Systems* 52, 2 (2013), 993–1000. https://doi.org/10.1007/s11235-011-9607-0

[49] B. Zhang and M. Becker. 2014. Reverse Engineering Complex Feature Correlations for Product Line Configuration Improvement. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. 320–327.

[50] Ángel Jesús Varela-Vaca and Rafael M. Gasca. 2013. Towards the automatic and optimal selection of risk treatments for business processes using a constraint programming approach. *Information & Software Technology* 55, 11 (2013), 1948–1973.