Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Feature models to boost the vulnerability management process[☆]

Ángel Jesús Varela-Vaca [*], Diana Borrego, María Teresa Gómez-López, Rafael M. Gasca, A. German Márquez

Data-Centric Computing Research Hub (IDEA), Universidad de Sevilla, Av. Reina Mercedes, Seville, 41012, Spain

A R T I C L E   I N F O

Article history:
Received 18 February 2022
Received in revised form 26 September 2022
Accepted 7 October 2022
Available online 18 October 2022

Dataset link: https://zenodo.org/record/7072369

Keywords:
Cybersecurity
Feature model
Vulnerability
Exploits
Reasoning
Vulnerable management process

A B S T R A C T

Vulnerability management is a critical and very challenging process that allows organisations to design a procedure to identify potential vulnerabilities, assess the level of risk, and define remediation mechanisms to address threats. Thus, the large number of configuration options in systems makes it extremely difficult to identify which configurations are affected by vulnerabilities and even assess how systems may be affected. There are several repositories to store information on systems, software vulnerabilities, and exploits. However, they are largely scattered, offer different formats and information, and their use has limitations, complicating vulnerability management automation. For this reason, we introduce a discussion concerning modelling in vulnerability management and the proposal of feature models as a means to collect the variability of software and system configurations to facilitate the vulnerability management process. This paper presents AMADEUS-Exploit, a feature model-based solution that provides query and reasoning mechanisms that make it easier for vulnerability management experts. The power of AMADEUS-Exploit is shown and evaluated in three different ways: first, the solution is compared with other vulnerability management tools; second, the solution is faced with another in a complex scenario with 4,000 vulnerabilities and 700 exploits; and finally, our solution was used in a real project demonstrating the usability of reasoning operations to determine potential vulnerabilities.

1. Introduction

Vulnerability management (Foreman, 2009) is a critical process that allows organisations to identify potential vulnerabilities, assess the level of risk, and define remediation mechanisms to address threats. However, current cyberattack chains (attack chains) used by attackers to penetrate systems are becoming increasingly sophisticated (Yadav and Rao, 2015). Therefore, attackers use a wide variety of attack vectors to exploit system vulnerabilities. According to the definition of the European Union Agency for Cybersecurity (ENISA),[1] "an attack vector is a means by which a threat agent can abuse weaknesses or vulnerabilities in assets to achieve a specific outcome". For example, a misconfiguration (Perez et al., 2017) of a software component can be used as an entry point (attack vector) for an attacker. Due to the wide variety of existing configuration options for software and hardware systems and the increasing number of vulnerabilities, vulnerability management becomes a very difficult process (Morrison et al., 2018), from identification to assessment (Dass and Namin, 2020; Murthy and Shilpa, 2018). Therefore, designing appropriate mechanisms to drive the vulnerability management process is crucial to minimise the exposure of end-users and organisations to external threats.

The first stage of a vulnerability management process is to inventory software and systems, and then identify vulnerabilities and exploits that may affect them (Engebretson, 2013). To do so, the elements involved and their characteristics (i.e. service names, ports, software versions, etc.) must be identified, and which known vulnerabilities and exploits may affect them. Currently, there are vulnerability catalogues/repositories, such as the National Vulnerability Database (hereinafter NVD) (Anon., 2020a). These catalogues provide information related to vulnerabilities, associating these vulnerabilities with the products they affect (software, hardware, operating systems, etc.). This information is crucial to determine whether a vulnerability can be used as an attack vector and should or should not be taken into account for assessment. However, vulnerability repositories may have poor quality (Kuehn et al., 2021), limitations that hinder their use (Zhang et al., 2015) -such as a limited number of

[☆] Editor: Prof. Raffaela Mirandola.
[*] Corresponding author.
E-mail addresses: ajvarela@us.es (Á.J. Varela-Vaca), dianabn@us.es (D. Borrego), maytegomez@us.es (M.T. Gómez-López), gasca@us.es (R.M. Gasca), amtrujillo@us.es (A.G. Márquez).
[1] ENISA Threat Landscape https://www.enisa.europa.eu/news/enisa-news/enisa-report-the-2017-cyber-threat-landscape.

https://doi.org/10.1016/j.jss.2022.111541
0164-1212/© 2022 Elsevier Inc. All rights reserved.

searches or hidden information retrieved-, or even their vulnerability information may be unlinked from exploits. In fact, automatic detection of system features and vulnerabilities remains an open problem (Gawron et al., 2015; Tommy et al., 2017). Current vulnerability management tools (e.g. OpenVAS) are very limited in the kind of operational capabilities they offer (they only prioritise by vulnerability impact) to carry out fine-grained vulnerability management. For instance, identifying a specific attack vector related to a combination of software version, platform, and operating system may be inferred from a vulnerability. These drawbacks make it even more difficult to assess vulnerabilities to obtain adequate vulnerability coverage (Dass and Namin, 2020). Therefore, it is essential to provide models that collect information from vulnerability and exploit databases and offer automatic analysis mechanisms to support the definition of accurate vulnerability identification and assessment.

The definition of models that enable automation and standardisation for the retrieval, identification, and assessment of vulnerabilities is one of the main challenges in the vulnerability management process (Wang and Guo, 2009a; Palmaers, 2021). Most academic approaches focus on the definition of semantic models (Wang and Guo, 2009a,b; Syed, 2020; Jia et al., 2018). Ontologies and knowledge graphs are used to link semantically related concepts that are generally unrelated. For instance, the CVO ontology (Syed, 2020) related to the concepts of NIST, CERT/CSS, and CVSS (Common Vulnerability Score System) (Anon., 2020b). Afterward, the ontology is used to infer certain information from different data sources. This ontology was used to identify tweets that mentioned any vulnerability. Vulnerabilities and exploits may affect different parts of a system at different levels of granularity, e.g., operating systems, platforms, applications, components, versions, etc. This creates a controversy over the variability, i.e., which combination of those parts represents a vulnerability for the target system or infers whether any variety of those parts may affect the target system. Due to the high variability in both systems, vulnerabilities, and exploits, the interest in applying configuration models to analyse vulnerability emerged (Kenner et al., 2020).

In previous work, we presented AMADEUS (Varela-Vaca et al., 2020) as a solution that uses feature models (hereinafter FMs) as formal models to gather the variability of known affected elements (software, hardware, operating systems, etc.) represented in vulnerabilities. The main advantage of using FMs is that they can help us in two ways: firstly, by bringing together all the elements represented in a unified model; and, secondly, the use of FMs opens up the possibility of using automatic analysis mechanisms to support the definition of appropriate security tests. However, we did not address some limitations: (1) AMADEUS only supports one vulnerability repository, lacking the integration of vulnerability and exploit repositories; (2) AMADEUS generates FMs but without taking exploits into account; (3) cross-tree constraints are generated in a separate file of the FMs file, and this limited the use of the reasoner; (4) the reasoning capabilities are not fully explored.

In this paper, our aim is to extend the previous work (Varela-Vaca et al., 2020) by improving AMADEUS empowered by the following items:

1. In the previous work, AMADEUS only integrated one vulnerability repository. In this paper, we propose to integrate more vulnerability repositories and also integrate exploit repositories.
2. In the previous work, AMADEUS only considered vulnerability information. In the new approach, we redefine FM generation algorithms to take vulnerabilities and exploits information into account.
3. In the previous work, AMADEUS was defined on top of the old-fashion FAMA Framework. In the new approach, the core of AMADEUS Exploit has been completely reimplemented to support a new FM engine to facilitate reasoning capabilities.
4. In the previous work, AMADEUS provided only a few operations and was very limited. In the new approach, we propose new FM reasoning capabilities with new operators to facilitate vulnerability analysis.
5. AMADEUS-Exploit has positioned itself against a wide range of vulnerability management tools to demonstrate the extent of the functionalities available on the market and the feasibility of the solution when applied in real contexts.
6. In the previous work, AMADEUS was tested with a bunch of vulnerabilities. In the new approach, AMADEUS-Exploit is evaluated in three different ways: (1) it is compared with other vulnerability management tools concerning certain capabilities for the identification and reasoning of vulnerabilities and exploits; (2) it is applied in a synthetic scenario consisting of several applications and services affected by 4000 vulnerabilities and 674 exploits to demonstrate the ability to generate a large number of models; and (3) a real scenario in a security project is used, in which we apply our approach to recognise services, vulnerabilities, and exploits, and to apply reasoning operations to define a concrete list for prioritising vulnerability assessment.

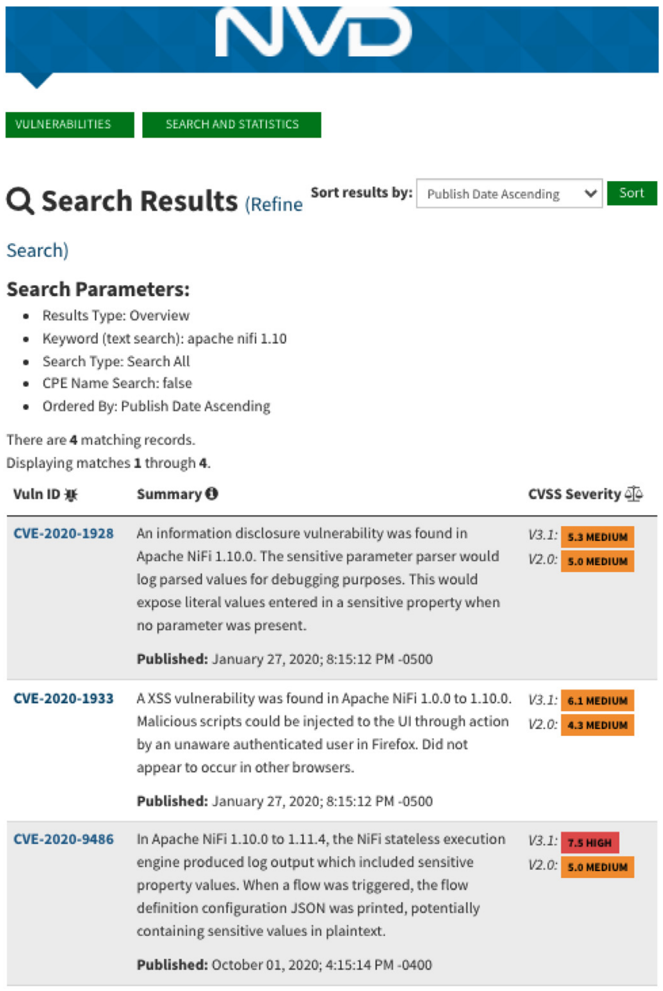
In summary, we present AMADEUS-Exploit[2] as a new solution to cover the limitations of AMADEUS and other commercial tools, increasing the functionalities. AMADEUS-Exploit allows the automatic generation of FMs from different vulnerability and exploit repositories, enabling an improved vulnerability management process boosted by automatic analysis mechanisms, making it easier for vulnerability management experts to identify potentially vulnerable software and system configurations or to prioritise vulnerabilities to be assessed. Thus, AMADEUS-Exploit is conceived to assist/support experts in the process of discovering, identifying, and assessing vulnerabilities. Therefore, AMADEUS-Exploit provides queries and reasoning operations to support and assist this crucial task. The current vulnerability and exploit databases enable for specific search capabilities. However, this search capability is limited to specific terms and information, and more sophisticated operations that address both are not available. Our approach tries to provide these types of operations.

The rest of the paper is organised as follows. Section 2 presents the basics on feature modelling, vulnerabilities, and exploits to better understand the proposal. Section 3 introduces the proposal, describing the integrated modules to achieve the objectives set out in the methodology. Section 4 details the formalisation of FMs. Section 5 illustrates how FMs can be used to reason and achieve better results in security testing. Section 6 compares our solutions with other commercial tools, and assesses the feasibility of our approach on real scenarios. Section 7 summarises previous proposals in the area. Section 8 analyses the threats to the validity of this study; and finally, conclusions are drawn and future work is outlined in Section 9.

2. Foundations

This section introduces some terms related to cybersecurity vulnerabilities, exploits, and feature modelling to facilitate the understanding of the proposal.

2 https://doi.org/10.5281/zenodo.7072369



Search Results (Refine Search)

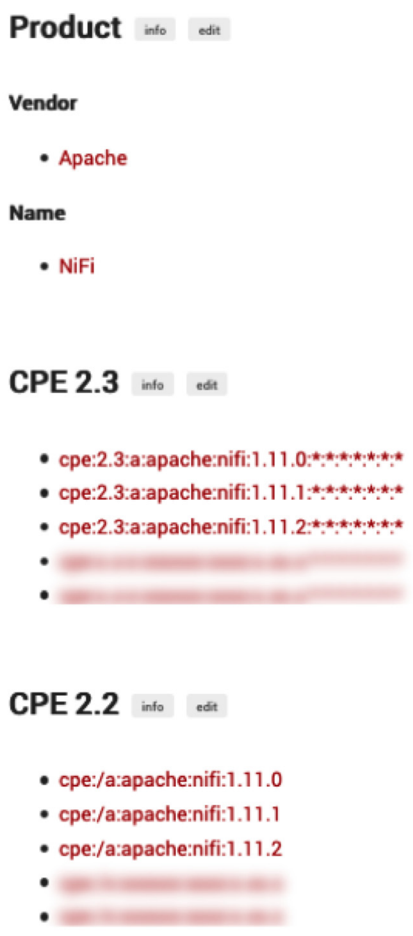
Search Parameters:

- Results Type: Overview
- Keyword (text search): apache nifi 1.10
- Search Type: Search All
- CPE Name Search: false
- Ordered By: Publish Date Ascending

There are **4** matching records.
Displaying matches **1** through **4**.

Vuln ID	Summary	CVSS Severity
CVE-2020-1928	An information disclosure vulnerability was found in Apache NiFi 1.10.0. The sensitive parameter parser would log parsed values for debugging purposes. This would expose literal values entered in a sensitive property when no parameter was present. **Published:** January 27, 2020; 8:15:12 PM -0500	V3.1: 5.3 MEDIUM V2.0: 5.0 MEDIUM
CVE-2020-1933	A XSS vulnerability was found in Apache NiFi 1.0.0 to 1.10.0. Malicious scripts could be injected to the UI through action by an unaware authenticated user in Firefox. Did not appear to occur in other browsers. **Published:** January 27, 2020; 8:15:12 PM -0500	V3.1: 6.1 MEDIUM V2.0: 4.3 MEDIUM
CVE-2020-9486	In Apache NiFi 1.10.0 to 1.11.4, the NiFi stateless execution engine produced log output which included sensitive property values. When a flow was triggered, the flow definition configuration JSON was printed, potentially containing sensitive values in plaintext. **Published:** October 01, 2020; 4:15:14 PM -0400	V3.1: 7.5 HIGH V2.0: 5.0 MEDIUM

(a) Apache NiFi 1.10 search in NVD.



Product

Vendor

- Apache

Name

- NiFi

CPE 2.3

- cpe:2.3:a:apache:nifi:1.11.0:*:*:*:*:*:*:*
- cpe:2.3:a:apache:nifi:1.11.1:*:*:*:*:*:*:*
- cpe:2.3:a:apache:nifi:1.11.2:*:*:*:*:*:*:*
- [redacted]
- [redacted]

CPE 2.2

- cpe:/a:apache:nifi:1.11.0
- cpe:/a:apache:nifi:1.11.1
- cpe:/a:apache:nifi:1.11.2
- [redacted]
- [redacted]

(b) Apache NiFi 1.10 vulnerability in VulDB.

Fig. 1. Example of vulnerabilities in different databases.

2.1. Vulnerability repositories

Several catalogues and repositories collect vulnerabilities that characterise different systems. They also provide information on how attack vectors and vulnerabilities interact. Among all these repositories, some stand out, such as NVD (Anon., 2020a), the US-CERT[3] vulnerability notes database, VulDB[4] or IBM X-FORCE.[5] This paper focusses on NVD and VulDB databases due to their wide use, continuous data update, and reliability. Furthermore, they have web tools that provide vulnerability search mechanisms, as illustrated in Fig. 1(a). This picture shows the indexed results related to the query about vulnerabilities of Apache Nifi 1.10. The query yielded three records, CVE-2020-9486,[6] CVE-2020-1933, and CVE-2020-1928.

Due to the wide range of target systems and configurations, they can easily be affected by vulnerabilities. An example of the extremely high number of vulnerabilities is the 160,732 vulnerabilities registered in NVD[7] (13,761 new vulnerabilities added in 2021), affecting 1824 vendors and 5999 products. However, the use of these repositories (e.g., NVD and VulDB) may have usage limitations, such as VulDB enabling 50 results of vulnerabilities per search. For example, the free version of VulDB provides limited information on vulnerabilities. For the CVE-2020-9485 obtained in the previous example (cf., Fig. 1(a)), we obtained the information shown in Fig. 1(b). However, NVD provides comprehensive information about vulnerabilities, including JSON-based feeds that can be consumed for offline use of the database.

2.2. Known affected configurations (CPE)

Using the terminology in NVD, the Known Affected Configurations can be described through a set of Common Platform Enumerations (CPE) (Parmelee et al., 2011) $\{cpe_1, cpe_2, \ldots, cpe_n\}$.

Definition 1 (*CPE*)**.** A CPE cpe_i represents a configuration of a system by a list of pairs $\langle a, v\rangle$ attribute-value that describe the products and scenarios in which vulnerabilities may occur.

In turn, these CPEs are represented by a set of Known Affected Software Configurations (hereinafter *Configurations*). To formalise the possible configurations, the CPE standard (Parmelee et al., 2011) created by the MITRE Corporation is used. It identifies the features of the contexts in which vulnerabilities could be

[3] Vulnerability notes database: https://www.kb.cert.org/vuls/.
[4] The Community-Driven Vulnerability Database: https://vuldb.com/
[5] Internet security systems x-force security threats: https://exchange.xforce.ibmcloud.com/.
[6] Acronym for Common Vulnerabilities, and Exposures (CVE).
[7] Data obtained from CVE Details: https://www.cvedetails.com/.

exploited, providing key information in the definition, enforcement, and verification of IT policies, such as vulnerabilities or configurations.

For a cpe_i[8] to be valid, each attribute (a_i) must appear only once, from the following options:

- *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 in which the product operates, e.g., Windows.
- *target_hw* characterises the architecture, e.g., x86.
- *other* describes any other information.

The value field (v_i) associated with each attribute (a_i) is usually a UTF-8 string. However, there are two logical values that can also be assigned to indicate, respectively, that there are no restrictions applicable to that attribute (value *ANY*) or that there is no valid value (value *NA*, Not Applicable). Thus, a CPE 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 identifier cpe_x is used to quickly identify and differentiate CPEs from each other. This paper uses *Formatted String Binding* (FSB), which consists of a list of attributes delimited by colons[9] 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 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 for the CPE 2.3 can be represented as:

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

The values of the attributes describe a configuration with vulnerability in an operating system (*part* = o), released by Linux (*vendor*), named Linux Kernel (*product*) at version 2.6.0 (*version*). The remaining attributes take the wildcard value ($*$) in FSB, which is the logical value *ANY*. As can be seen, the first pair (cpe:2.3) is ignored, as it only points out the CPE format.

[8] Considering CPE 2.3 specification (Parmelee et al., 2011).

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

2.3. Vulnerabilities

A vulnerability is defined by ISO/IEC 27005:2008 as "a weakness of an asset or group of assets that can be exploited by one or more threats, where an asset is anything that has value to the organisation, its business operations, and their continuity, including information resources that support the organisation's mission". With the idea of automating vulnerability scanning, the cybersecurity community has made several efforts to standardise the way vulnerabilities are represented. To this end, NVD, Vulners, VulDB, and other repositories use the de facto standard to represent vulnerabilities, Common Vulnerabilities, and Exposures (CVE) (Anon., 2020c). CVE can be defined as a reference method for structural publication of known vulnerabilities for easy management and sharing.

Definition 2 (*CVE*). A CVE is a tuple $\langle CVE\_id, description, impact,$ CPEs$\rangle$ of information about a vulnerability, where:

1. *CVE_id* is the mandatory identifier of each vulnerability.
2. *description* is the summary to describe the vulnerability textually.
3. *impact* of the vulnerability, following the CVSS standard (Anon., 2020b) to assess the severity of the vulnerability. CVSS in its different versions (up to current 3.1) proposes a formula that returns a value between 0 and 10 to represent the lowest and highest severity.
4. *CPEs* is a set $\{cpe_1, cpe_2, \ldots, cpe_n\}$.

Table 1 shows an example of two CVEs related to Apache NiFi 1.10 from a query obtained for NVD, as shown in Fig. 1(a). These represent two different vulnerabilities that affect Apache Nifi in versions 1.0.0 and 1.10, as shown in the CPE column.

As in the case of NVD, CVEs representing vulnerabilities are made up of a set of vulnerable contexts, the so-called *Configurations*. A *Configuration* is, in turn, composed of a list of vulnerable CPEs $\{cpe_1, cpe_2, cpe_3, \ldots, cpe_n\}$. Also, optionally, to specify concrete runtime environments in which the vulnerability can be reproduced, a set of *Running Configurations* (RC) can be included as a set of extra CPEs $\{cpe_{n+1}, cpe_{n+2}, cpe_{n+3}, \ldots, cpe_{n+m}\}$. An RC is a "special" type of CPE, since each combination of CPEs is considered with respect to each RC separately. Therefore, RC establishes certain environmental conditions under CPE in which RC can perform. Table 2 shows an example of RC (cpe_{58}) which indicates Mozilla Firefox as the running environment for which configurations (cpe_1, cpe_2, cpe_3, ...) can be exploited. In the presence of RCs, combinations of CPEs must be considered with respect to each RC separately. Table 2 shows a piece of an example of configurations for the vulnerability CVE-2020-1933 associated with Cross-Site Scripting in Apache NiFi for versions 1.0.0 to 1.10.0. In this example, there is only one RC, so cpe_1 can occur with cpe_{58}; cpe_2 can occur with cpe_{58}; ... and so on until all combinations are covered. In summary, Apache Nifi version 1.0.0 beta-rc1 and the others in the table under the environment of Mozilla Firefox (in any version) are affected by the Cross-Site Scripting vulnerability CVE-2020-1933.

2.4. Security exploits

In general, a security exploit is a fragment of software used to attack a software or hardware system by leveraging a vulnerability. Exploits are designed to cause damage to systems in order to change their behaviour and derive some benefit for the attacker. Examples are pieces of software that attempt to produce arbitrary code executions, a denial of service, or a privilege granted. There is no standard way to represent exploits, but typically they can be described in the following information.

Table 1
NVD results for “*Apache NiFi 1.10*” query.

Vuln. ID	Summary	CVSS Severity	CPEs
CVE-2020–1933	A XSS vulnerability was found in ...	*V3.0*: 6.1 *V2.0*: 4.3	{cpe:2.3:a:apache:nifi:1.0.0:..., ...}
CVE-2020–1928	An information disclosure vulnerability was ...	*V3.0*: 5.3 *V2.0*: 5.0	{cpe:2.3:a:apache:nifi:1.10.0:... ...}

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

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:–:*:*:*:*:*:*:**

- *Exp_Id* is the identifier of the exploit.
- List of *CVEs* with which the vulnerability is associated (if applicable).
- *date* of publication of the exploit.
- *type* of exploit, e.g., webapp, shellcode, remote, papers.
- *platf* is the platform affected by the exploit, e.g., Hardware, PHP, Linux, Windows.
- The author of the exploit published.
- *app:* Vulnerable app that provides a link to the downloadable version of the platform.
- *sc:* The source code or instructions which constitute the exploit itself.

There are repositories of exploits, such as Exploit-DB (Anon., 2021) by Offensive Security Community, which provide a significant number of exploits associated with vulnerabilities. Specifically, this repository provides 42,802 exploits.[10] Following the example in Fig. 1(a), there are no exploits published for Apache Nifi. An illustrative example is the exploit 27,227 shown in Fig. 2. This exploit is related to the vulnerability CVE-2006-0733 for the WordPress Core 2.0 component concerning HTML Injection.

2.5. Modelling vulnerabilities and exploits: Feature modelling and automatic analysis

We start with a consideration of the approaches for vulnerability and exploit modelling in the context of cybersecurity and vulnerability management.

As mentioned in the introduction, the use of models to represent vulnerabilities is a challenge for the vulnerability management process (Palmaers, 2021; Wang and Guo, 2009a). Threat modelling (Xiong and Lagerström, 2019) is widespread in cybersecurity as a discipline for assessing and identifying potential vulnerabilities. However, threat modelling currently has several challenges to face (Xiong and Lagerström, 2019): (1) automate security analysis and modelling, and (2) integrate with threat and vulnerability databases. Several academic approaches define semantic models (i.e., ontologies and knowledge graphs) (Wang and Guo, 2009a,b; Syed, 2020; Jia et al., 2018) to homogenise and interrelate concepts of vulnerabilities and others. However, vulnerability and exploit information are often addressed separately. The high range of information and variability of vulnerabilities and exploits (e.g., devices, operating systems, platforms, applications, components, versions, configurations, source code, etc.) makes it difficult to find a model that enables reasoning and represents relations, variability, and commonalities. For example, any variety of software versions of a component may affect the targeting system due to certain vulnerabilities, and this may be inferred. Due to that, interest has arisen in applying configuration models to analyse vulnerabilities (Kenner et al., 2020).

FMs are a widely used technique to represent software product lines (SPLs) (Clements and Northrop, 2002) in tree-like structures. Although there are other representations (e.g. OVM Roos Frantz et al., 2009), FMs have become the de facto standard for representing common and variable characteristics in an SPL. In general, an FM is a model that defines features and their relationships. FMs can be defined in many ways (i.e., textual, formal, graphical, etc.) albeit the most widely used is the one proposed by Czarnecki (Benavides et al., 2010), exemplified in Fig. 3.

Around FMs a field related to the Automatic Analysis of Feature Models (AAFM) (Benavides et al., 2010) has emerged. AAFM aims to extract information from the models by using some logic or reasoning mechanisms, e.g., determining product configurations or tests.

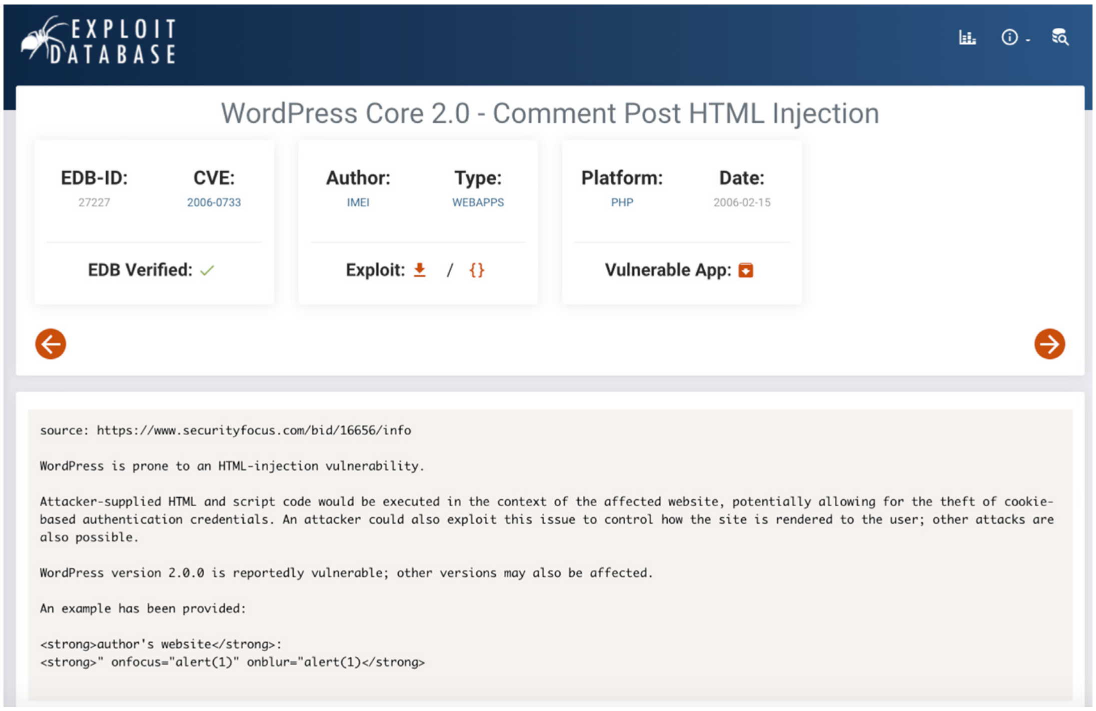
Regarding the tools, there are many of them that allow FMs to be defined and provide some automatic analysis mechanisms. Some examples of tools are: FAMILIAR (Acher et al., 2013), FeatureIDE (Thüm et al., 2014), Gears,[11] FaMa (Benavides et al., 2013), FaMaPy (Galindo and Benavides, 2020), SPLOT (Mendonca et al., 2009), pure::variants,[12] VariaMos (Mazo et al., 2015) or Glencoe (Schmitt et al., 2018). The new approach presented in this paper is powered by the FaMaPy framework. FaMaPy is a Python-based AAFM framework that enables multi-solver and multi-metamodel support for the integration of AAFM tools into the Python ecosystem. FaMaPy supports multiple solvers (e.g., Glucose or Minisat) and multiple variability models, such as the FaMa format Benavides et al. (2013). FaMaPy defines an FM-metamodel that allows and provides transformations from different formats to the FaMaPy metamodel. In terms of reasoning capabilities, FaMaPy provides more than ten operations for cardinality-based feature models, e.g., valid model, valid product, error detection, error diagnosis, etc.

To perform an efficient vulnerability management process, it is crucial to choose the appropriate vulnerabilities (i.e, “vulnerability coverage”) and the elements of the systems and software that need to be checked (Dass and Namin, 2020). As mentioned above, vulnerability and exploit repositories offer search engines to extract information about them. However, these searches are sometimes limited, as the information is only available for a fee, and it is not always possible to secure complete information (Kuehn et al., 2021; Zhang et al., 2015). Moreover, the amount of extracted information can be unmanageable, which is a crucial problem since this information is essential to identify which elements (parts, vendors, versions, OS, etc.) of our systems and software need to be checked in security testing. Therefore, we

[10] Data obtained from Exploit-DB: https://www.exploit-db.com/exploit-database-statistics.

[11] Gears: www.biglever.com.

[12] pure-systems: www.pure-systems.com.



```
source: https://www.securityfocus.com/bid/16656/info

WordPress is prone to an HTML-injection vulnerability.

Attacker-supplied HTML and script code would be executed in the context of the affected website, potentially allowing for the theft of cookie-
based authentication credentials. An attacker could also exploit this issue to control how the site is rendered to the user; other attacks are
also possible.

WordPress version 2.0.0 is reportedly vulnerable; other versions may also be affected.

An example has been provided:

<strong>author's website</strong>:
<strong>" onfocus="alert(1)" onblur="alert(1)</strong>
```

Fig. 2. Example of exploit associated to a CVE.

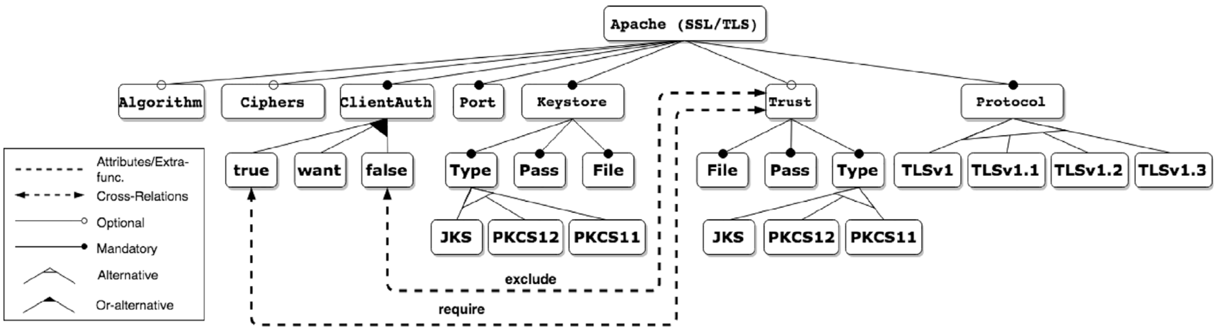


Fig. 3. Example of FM for Apache security configuration (Varela-Vaca et al., 2019).

propose to search and extract information from multiple vulnerability and exploit repositories and provide a unified model that helps to define appropriate security tests. Indeed, FMs are an interesting approach to represent the variability of elements within CVEs, CPEs and exploits. The main advantage of using FMs is that they can help us in two ways: first, by bringing together all the elements represented in a unified model; and, secondly, the use of FMs opens up the possibility of using automatic analysis mechanisms to support the definition of appropriate security tests.

3. AMADEUS-exploit

An in-depth analysis of potential security vulnerabilities can facilitate a proper vulnerability management process based on possible attack vectors and their exploits (Dass and Namin, 2020; Oyler and Saiedian, 2016; Skopik et al., 2014).

As mentioned above, AMADEUS was presented in a previous work (Varela-Vaca et al., 2020) as a methodology for automatically creating FMs by integrating information from the vulnerability repository and reasoning to determine attack vectors with certain features. However, certain aspects were left pending, such as the subsequent extraction of exploits to assess vulnerabilities, the incorporation of new vulnerability repositories, and the improvement of reasoning about the models. These tasks, among others, are crucial to complete the task of the vulnerability management process, i.e., to enable the discovery and analysis of configurations with vulnerabilities and exploits available for testing within the software and hardware resources of an ecosystem. The new proposed framework, AMADEUS-Exploit follows the process shown in Fig. 4, which describes the workflow, where the white boxes represent the different tasks that are performed, such as to 'Analyse infrastructure'. Attached to the tasks by dashed arrows, it can be found a description of the data generated or consumed by each task, e.g., the list of terms generated by 'Provide Terms'. Bold arrows show in which order the gateways and tasks are reached and performed. For instance, there is an OR-gateway (X-diamond symbol) to choose the path to execute and AND-gateways (+ -diamond symbol) to allow parallel execution of tasks. Certain tasks are grouped into stages labelled as grey boxes for ease of understanding, e.g., 'Discover target elements' involves 'Analyse the infrastructure' and 'Provide terms'. These stages are explained in the following subsections.

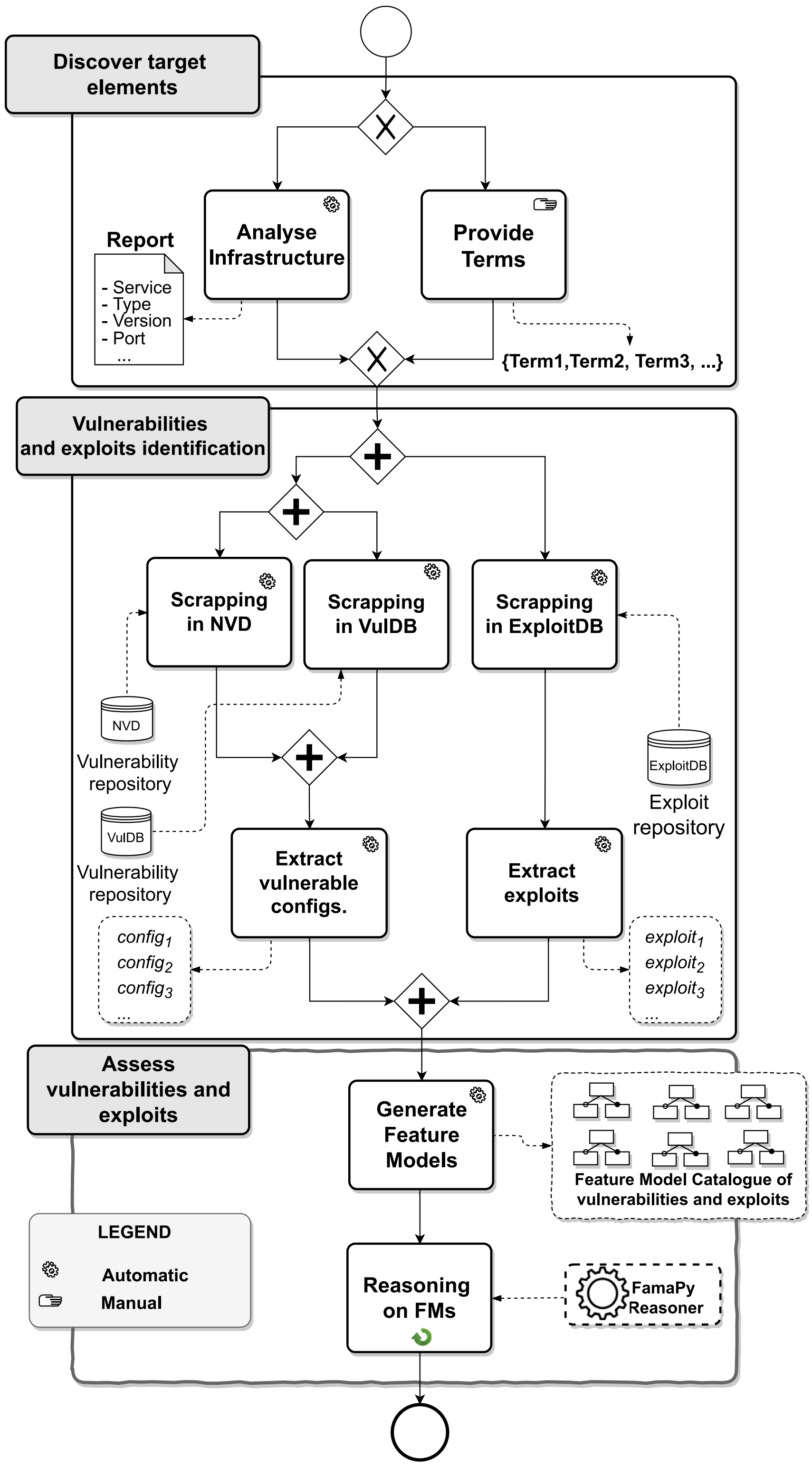


Fig. 4. AMADEUS-Exploit framework overview. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

AMADEUS-Exploit can be placed between the preparation, discovery, and scanning phases in the vulnerability management process (Palmaers, 2021; Foreman, 2019). These phases aim to analyse and gather information about potential targets, and to identify particular aspects of those targets, e.g., exposed services, open ports, operating system names, vulnerabilities, etc.

To understand the framework, the tasks that make up the AMADEUS-Exploit process are marked in the workflow of Fig. 4 as manual (hand symbol), and automatic (engine symbol). Manual tasks require unavoidable human intervention.

As mentioned, the workflow consists of three stages: (1) discovering the target elements to be analysed; (2) the automatic extraction of information from both vulnerabilities and exploits (Vulnerability and Exploit identification); (3) vulnerability and exploits assessment in which feature models are generated for each vulnerability and reasoning on FMs as the application of reasoning techniques on the obtained FMs. Each part is described in detail below.

3.1. Discover target elements

Depending on the system configurations (software, hardware, network, and others), specific vulnerabilities and exploits can be identified, or specific assessment techniques can be applied (Shah and Mehtre, 2015; Backes et al., 2017).

Therefore, the first step is to define the scope of the analysis by discovering the elements involved in the analysis. This scope enables the establishment of the boundaries and goals of the analysis. There are several solutions to collect and retrieve the configuration used in organisations, including active and passive analysis tools. In our proposal, the systems or devices involved can be derived from the infrastructure analysis or provided by experts through a set of terms (cf., Analyse Infrastructure and Provide Terms).

As for infrastructure analysis (cf. Analyse Infrastructure), systems can be audited using active tools, such as Lynis[13] and Nmap (Network Mapper).[14] Nmap is a well-known tool widely used to audit the security of firewalls, networks, measure network traffic, or detect vulnerabilities. Due to its popularity in the security community, Nmap is integrated in the AMADEUS-Exploit implementation, although others could be adapted as well.

However, the user can include a list of terms (cf. Provide Terms) to analyse the vulnerabilities of a system. For this reason, AMADEUS-Exploit works in two modes of operation, custom and automatic. The custom mode allows users to provide a list of terms and keywords for a set of target systems, e.g., the terms OpenSSH and version 7.7. The automatic mode invokes an analysis tool (Nmap in our case) on a set of target systems. AMADEUS-Exploit addresses the information retrieved from this tool as a list of terms and keywords, where the tuples are returned as $\langle service, version\rangle$, for example: $\langle OpenSSH, 7.7\rangle$, $\langle ApacheHTTPServer, -\rangle$, $\langle OpenVPN, 2.3.17\rangle$.

3.2. Vulnerabilities and exploits identification

From the information extracted in the previous step (report or list of terms), possible configurations with vulnerabilities can be analysed. Therefore, terms related to running services, versions, active ports, etc, are used to search for vulnerabilities (CVE) and exploits. These searches can be performed on repositories, where the information can be extracted using a scrapper (cf. Scrapping NVD, VulDB, and Exploit-DB). AMADEUS-Exploit integrates three main data sources: NVD (Anon., 2020a), VulDB (Anon., 2020d) and Exploit-DB (Anon., 2021). The integration is possible thanks to the implementation of a Web scraper module that allows the automatic search and extraction of information in these repositories. The scrapper analyses structures similar to Figs. 1(a) and 1(b) and collects data, keeping only specific and relevant information, such as the CVE ID, description, CPEs. As shown in Fig. 4, the scrapping activities can be run in parallel as different repositories are accessed. Similarly, since vulnerability and exploit extraction are independent tasks, they can also be executed in parallel.

After gathering the vulnerabilities represented by the CVE and exploits, it is time to analyse the possible features of the scenarios in which these vulnerabilities can be exploited. Therefore, AMADEUS-Exploit extracts different sets of CPEs (cf. Extract Vulnerable Configurations), represented, for each vulnerability (CVE). For example, the CVE-2020-1933 vulnerability describes the malicious scripts that can be injected into Apache NiFi 1.10. However, several questions arise, such as *on which specific software configuration this vulnerability applies*, *whether it can be related to software, hardware, application or an operating system*, or *whether this vulnerability exists for each version or release*. For instance, the CVE-2020-1933 contains 57 CPEs describing 57 different versions of Apache NiFi running in Mozilla Firefox affected by this vulnerability.

Similarly, information about exploits (cf. Extract exploits) is extracted from ExploitDB (Anon., 2021). In this way, the CVE IDs are used as necessary information to search for direct exploits related to those vulnerabilities. We retain all valuable features (i.e., Exploit ID, platform, etcetera) of each exploit obtained for use in subsequent feature model generation. Bear in mind that some vulnerabilities may have one or more exploits, but others do not. Therefore, (i) vulnerabilities 'with exploits' can be directly related to possible exploits to be used in a future security test, and (ii) vulnerabilities 'without exploits' should be known as they may be potential security issues or challenging vulnerabilities to be tested.

For example, the aforementioned vulnerability CVE-2020-1933 has no exploit, whereas the vulnerability CVE-2009-3555 (associated with Apache HTTP servers and OpenSSL) has two exploits. Therefore, we have two exploits that can be tested against all the CPE covered by the CVEs.

3.3. Assess vulnerabilities and exploits

Using these concepts as a basis, the AMADEUS-Exploit framework attempts to obtain valid FMs (cf. Generate Feature Models) of the discovered target systems. All these generated FMs constitute a catalogue (Varela-Vaca et al., 2019). The contribution in this paper proposes a set of algorithms that create FMs adapted to the vulnerability context, giving rise to a catalogue of scenarios that collects the attack scenarios that may occur depending on the vulnerabilities. This catalogue can be used in many scenarios by reasoning over the models (cf., Reasoning on FMs). The reasoning task in Fig. 4 is represented as an iterative task (cf., green arrow) since the customer will require to apply multiple operations depending on the task at hand.

Taking into account that FMs represent a catalogue of vulnerabilities, including their configurations and exploits, various reasoning operations can be developed. Some examples are: the generation of attack vectors, the extraction of exploits, the extraction of vulnerabilities, the verification of a configuration, or the determination of the lack of exploits necessary to test a vulnerability, among others.

Sections 4 and 5 detail how FMs are created and the possible reasoning mentioned.

4. Generation of feature models

As discussed, the high variability – due to the large number of potential vulnerabilities, affected configurations, and exploits – makes the management of potential threats too complicated. The creation of FMs that gather and structure this information makes vulnerability analysis easier and more automated. This section describes how an FM of vulnerabilities and exploits is, and how an FM catalogue can be created using the vulnerabilities and exploits sources.

The inference of FMs from vulnerabilities was introduced in the previous work (Varela-Vaca et al., 2020). In that approach,

[13] Lynis: https://cisofy.com/lynis/.

[14] NMAP: https://nmap.org/.

we inferred an FM for a CVE, but it was built considering only a vulnerability database and omitting exploit information. In this paper, AMADEUS-Exploit extends the previous approach by pursuing the construction of an FM catalogue that gathers vulnerabilities extracted from various repositories and integrates them with exploits extracted from others. AMADEUS-Exploit generates an FM for each CVE (vulnerability), including each CPE of its configurations and the associated exploits for each CVE. Therefore, each configuration collected in the FM is vulnerable according to NVD or VulDB vulnerabilities and can be associated with some exploits extracted from Exploit-DB.

Definition 3 (*FM of Vulnerabilities and Exploits*)**.** Let *CPEs* be a list of known affected configurations and running configuration environments $\{cpe_1, cpe_2, cpe_3, \ldots, cpe_n\}$ and *EXP* be a set of exploits $\{exp_1, exp_2, exp_3, \ldots, exp_m\}$. An FM of vulnerabilities and exploits is an equivalent representation of all combinations[15] of each CPE and the EXP described in each vulnerability.

$$FM \equiv CPEs \ltimes_{pred} EXP \iff products(FM)$$
$$= \{(cpe_1, exp_1), (cpe_2, exp_1), \ldots \quad (5)$$
$$(cpe_1, exp_2), (cpe_2, exp_2), \ldots, (cpe_n, exp_m)\}$$

In our approach, the above-mentioned FM generation is carried out in two main phases:

1. Retrieval of an unrestricted FM containing information only within the CPE and EXP sets.
2. Inclusion of restrictions in the form of cross-tree relations in that FM, avoiding possible configurations that the FM could generate without restrictions.

One of the main concerns in the proposed algorithms is correctness. The valid operation can demonstrate that the generated model is correct; therefore, it can obtain at least one valid product. In addition to model validation, the number of products can be used as a validation operation. In this sense, the number of products helps as a correctness metric to measure accuracy and recall (Lopez-Herrejon et al., 2015). Therefore, if the number of products differs from the expected combination of CPE, RC, and EXP, FM is not equivalent (see Definition 3). The correctness of our algorithms has been studied and proven in previous work (Varela-Vaca and Gasca, 2013). Although new features and relations are included in FMs (according to EXP), the assumption regarding the number of products remains valid, since the new features are included through mandatory relations to the root; therefore, the number of products should remain unaltered with regard to the previous one.

4.1. Retrieving unrestricted feature model from CPEs and exploits

The so-called reverse engineering in SPLs (She et al., 2011; Lopez-Herrejon et al., 2015) provides mechanisms to generate FMs from a set of configurations. Reverse engineering that can be applied in this context of cybersecurity is relatively limited, with just 12 attributes to describe CPEs and running configurations, and a set of exploits. This is the case, for example, with the *product* attribute, which determines *vendor* and *part*, not being possible for the same *product* to come from two different *vendors* or *parts*. In addition, these three attributes must have a specific value, as it is impossible to assign them the value '*ANY*'. These particularities are the main motivation to propose a specific algorithm to create FMs and to include these restrictions in FM generation.

[15] We have used $\ltimes_{pred}$ with the semantic of the left join operator.

Table 3
Running example of CPEs and exploits for a vulnerability.

CVE-ID-1
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:-:*:*:*:*:*:*:**
Exploits
$\langle exp_1, CVE\text{-}ID\text{-}1, \ldots\rangle$ $\langle exp_2, CVE\text{-}ID\text{-}1, \ldots\rangle$ $\langle exp_3, CVE\text{-}ID\text{-}1, \ldots\rangle$

The running example in Table 3 is used to illustrate each part of the proposed algorithms. It represents a vulnerability `CVE-ID-1` that encompasses two configurations, each with a CPE list $\{\{cpe_1, cpe_2, cpe_3\},\{cpe_4, cpe_5\}\}$ and a running configuration list $\{cpe_6\}$, empty for *Configuration 1*. Furthermore, we assume that this vulnerability can be used with three different exploits $\{exp_1, exp_2, exp_3\}$.

The feature modelling algorithm is based on three steps: (1) creation of a sub-FM for each vendor and a sub-FM with every exploit; (2) creation of a sub-FM for each running configuration, and; (3) integration of these sub-FMs into a single FM tree, one for each CVE:

1. **Creation of a sub-FM for each vendor and a sub-FM with each exploit**. For example, we create an FM for the vendors olearni (cpe_1, cpe_2, cpe_3) and oteachy (cpe_4 and cpe_5), and similarly for the exploits exp_1, exp_2, and exp_3.
2. **Creation of a sub-FM for every running configuration**. For example, we create an FM for the vendor in the running configuration cpe_6.
3. **Integration of sub-FMs into a single FM**, i.e. integration of these sub-FMs (for vendors and exploits) into a single FM tree, one for each CVE. The 'rc' feature is included as an optional relation to the whole FM (as the running configuration may or may not appear).

Fig. 5 illustrates these steps for the running example in Table 3, connecting the CPE lists with blue lines, running configurations with green lines, and exploits with red dashed lines. In Fig. 6, it can be seen how the four sub-FMs are combined to create the complete FM for the example. The inclusion of cross-tree constraints in the unrestricted FM is described below. Algorithms 1 and 2 in the Appendix describe the concrete specification for the inference of FMs.

4.2. Include cross-tree constraints in the FM

Up to this point, the FM obtained encapsulates all attributes and values of the CPEs of a CVE and the related exploits. As mentioned above, the set of CPEs does not usually include such high variability, and the existence of some of its components is intrinsically related to the occurrence of others. Therefore, the inference of a set of constraints on an FM is necessary to

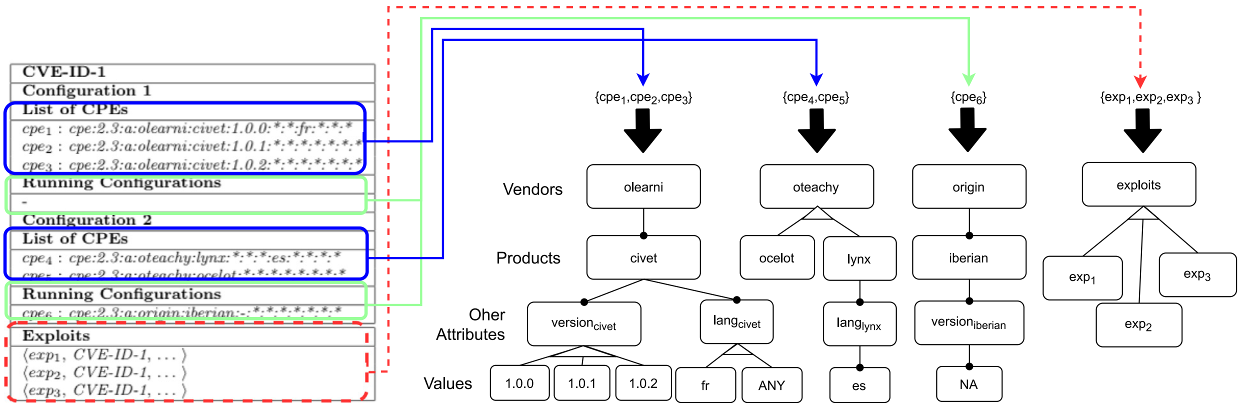


Fig. 5. Process of construction of the FM for the running example.

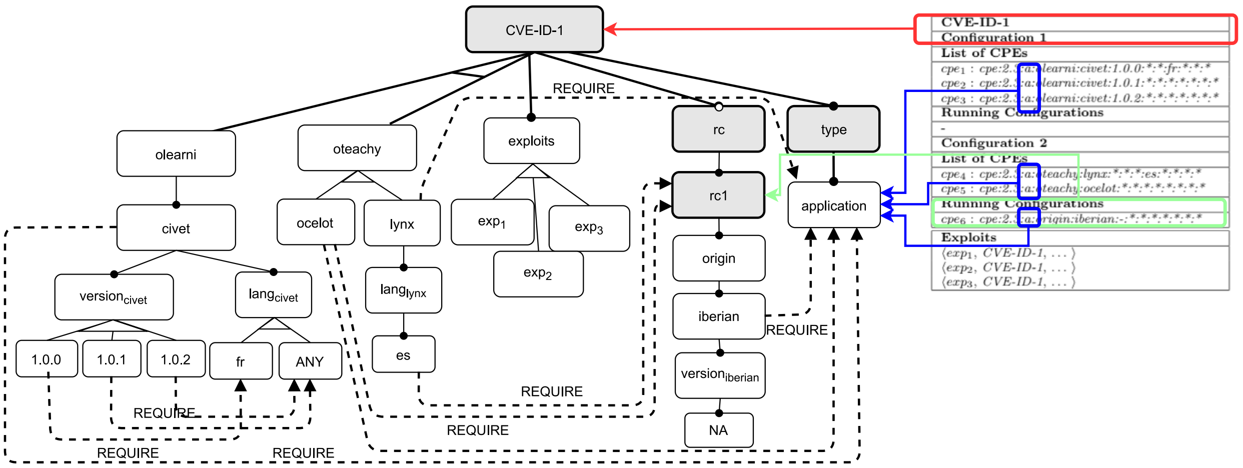


Fig. 6. Process of construction of the FM for the running example.

overcome this situation and restrict the number of feasible combinations by adjusting it. At this stage, AMADEUS-Exploit derives a set of constraints to adjust the FM variability according to the restrictions of the CPE attributes and the running configurations. As mentioned above, Fig. 6 is the adjusted version of the running example in Table 3. We should clarify that it is unnecessary to include specific cross-tree constraints between exploits and CPE features. In general, the exploits can be related to the CVE, referring to all the CPEs of the CVE.

The cross-tree constraints are derived from an analysis of the original list of CPEs, a clear descriptor of the possible valid configurations. Any other combination would result in an unlisted configuration and is therefore considered spurious. Recall that the whole point of this algorithm is to build an FM that can produce the same set of items contained in the original CPE list. Therefore, we only use two types of cross-tree constraints (*Require* and *XOR-require* Karataş and Oğuztüzün, 2013; Seidl et al., 2016). *Require* constraint is used when a feature requires other features with a non-direct family relation (e.g., $f1 \rightarrow f2$). On the other hand, *XOR-require* constraint establishes a required relation between a feature and a set of other features, allowing only one to appear at a time. A $f1$ XOR-require $\{f2, f3\}$ constraint is equivalent to:

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

The cross-tree derivation consists of the following three parts:

1. Creation of cross-tree constraints between the products and their associated type.
2. Creation of cross-tree constraints between feature leaves of the same product (between relevant attribute values of the same sub-FM).
3. Creation of cross-tree constraints between feature leaves of products and running configurations (between relevant attribute values and a sub-FM root of a running configuration).

A detailed specification of cross-tree derivation is defined in the Algorithm 3 in the Appendix section. Following the example in Table 3 and the FM generated in Fig. 5, several cross-tree constraints are found by applying the Algorithm 3. According to each of the three parts of the algorithm, the constraints found are as follows:

1. Cross-tree among features of the same vendor: relation among the *civet* product attributes, the constraints required between the features '1.0.0' and '*fr*' to enforce the achievement of *cpe*1, plus the two required relations between '1.0.1' and '1.0.2', and '*ANY*' features to enforce *cpe*2 and *cpe*3;
2. Cross-tree among products and types: the *civet*, *ocelot*, *lynx* products require the same type, thus, *application*;
3. Cross-tree among feature leaves and running configurations: the required relation between '*ocelot*' and '*rc*1' features to enforce the occurrence of running configuration features for *cpe*4, and the required relation between '*es*' feature and the '*rc*1' to enforce *cpe*5.

These cross-tree constraints are included to complete the FM, as shown in Fig. 6.

5. Reasoning on feature models

AMADEUS-Exploit finds FMs as a way of representing vulnerabilities and exploit information discarding the generic and conventional representations, such as lists or repository tables. An additional advantage derived from the use of FMs is the ability to store both vulnerabilities and their exploits as a catalogue (Varela-Vaca et al., 2019). AMADEUS-Exploit has been enhanced by the definition of an FM catalogue, which, in a way, could be considered as an interactive entity supporting a wide range of queries and reasoning operations. These operations are part of the classically automated analysis of FMs (Benavides et al., 2010), i.e. determine if a product is valid, obtain all products, validate the model, detect and explain errors, etc.

Please note that AMADEUS-Exploit is conceived to assist/support experts in the vulnerability management process. That is, to discover, identify, and assess vulnerabilities. Therefore, the queries and reasoning operations provided by AMADEUS-Exploit should be orientated to assist in this crucial task. Current vulnerability and exploit databases enable specific search capabilities, such as searching for vulnerabilities for CVE or CPE identifiers. However, this search capacity is limited to particular terms and information, and vulnerabilities and exploits are unlinked. No more sophisticated operations related to both information are available. Our approach tries to provide these types of operations. For instance, it is impossible to perform a complex search that describes a partial configuration of a CPE that can be affected by exploits. For a given exploit, discover whether partial elements that define CPEs are involved or obtain all possible CPEs that are affected by exploits and vulnerabilities. Some of the possible reasoning operations applied to any FMs are explained in the following subsections.

5.1. Reasoning about attack vectors

As mentioned above, attack vectors are a means by which a threat actor can abuse the weaknesses or vulnerabilities of assets to achieve a specific result. Therefore, attack vectors are necessary to assess a vulnerability. From the FM perspective, attack vectors represent the selection of features in the FM related to products, vendors, OS, version, running configurations, exploits, etc. that describe a known affected configuration by a vulnerability. One of our goals is to support security testing to assess a set of vulnerabilities that adequately covers the identified vulnerabilities. Using the reasoning capabilities provided by AMADEUS-Exploit, which was integrated with FaMaPy (Galindo and Benavides, 2020), we can apply certain operations tailored to the problem at hand. In particular, by obtaining the set of all products of the FMs (i.e., all attack vectors) or by applying a filter (i.e., completing an attack vector), we can generate useful information to obtain the attack vectors. From a security testing point of view, if the expert knows which specific vulnerability configuration to test, we could simply query the FM by fetching the products or applying a filter to it. In practise, the generation of all attack vectors from an FM can help to know the configuration space that we have to check to test all possibilities of a vulnerability. Therefore, it helps to know the configuration space that represents the vulnerability and to decide how to assess it. For example, FM is built for the vulnerability CVE-2018-15473, which affects OpenSSH 7.7. In practise, the security expert has entered into AMADEUS-Exploit the terms OpenSSH 7.7 and returned the FM for CVE-2018-15473 vulnerability. If we were to explore the entire configuration space to test this vulnerability, we have found 5008 different attack vectors (depending on the products) representing 256 CPEs and 3 exploits. An example of an attack vector obtained from the FM analysed is as follows: {CVE-2018-15473, type: {application}, source: {nvd}, exploits {exploit_45939}, dropbear_ssh_project, dropbear_ssh, dropbear_ssh_version, dropbear_ssh_version_0_35, dropbear_ssh_version, dropbear_ssh_version_0_35_update, dropbear_ssh_version, dropbear_ssh_version_0_35_update_test3}. In particular, AMADEUS-Exploit helps experts by pointing out a specific product Dropbear SSH in version (0.35) and updating (update_test3) that can be exploited for the exploit 45,939.

5.2. Reasoning about exploits

As stated above, vulnerability repositories, e.g., NVD or VulDB, do not link the CVE to exploits stored in other repositories (e.g. Exploit-DB). AMADEUS-Exploit allows us to obtain CVEs directly by identifying a set of exploits without using an FMs reasoner. In this way, AMADEUS-Exploit provides experts with a pointer to the exploit(s) to be used for each vulnerability (CVE). From a vulnerability management point of view, this operation gives the expert a hint on which vulnerabilities have direct resources to test them. Experts can use this information to define the assessment of vulnerabilities with exploits, including prioritisation, or to check some attack vectors against certain exploits. Therefore, if we identify the exploits for some environments, we could point out the configuration vulnerability. For example, exploit 7000 affects PHP platforms due to insecure cookie handling. The exploit provides a snippet of code to check the cookie settings in the admin panel. The exploit points out the vulnerabilities CVE-2008-6232 and CVE-2008-6231 that are related to the Pre Shopping Mall web application. However, we can extract more information that affects exploits, in particular by applying filters to the FM. For example, thanks to the FM of CVE-2008-6231, we found that 7000 exploits affect the product *pre_classified_listings* of the vendor Pre Shopping Mall. This information is not provided by the exploit but is contained in the CPEs within the CVE. This extraction of information from exploits requires the use of the reasoner, in particular, through filtering.

Similarly, for a set of vulnerabilities and exploits, AMADEUS-Exploit can determine the lack of exploits to be analysed. This operation does not require a specific reasoning operation, and because of it, vulnerabilities that cannot be directly exploited are identified. This is useful from a security point of view, as possible attack vectors can be generated from CVEs that we do not know how to assess. Therefore, experts must decide how to assess it, bearing in mind that there are attack vectors raised by CVEs that have no resources to use. For instance, the vulnerability CVE-2019-16905 affecting OpenSSH in different versions due to an integer overflow does not have an exploit. This lack of exploits does not prevent the problem; the problem is identified by AMADEUS-Exploit and must be managed and evaluated in case of having any of the affected versions.

5.3. Reasoning about vulnerabilities and exploits

For a given set of exploits, vulnerabilities related to them can be extracted. This operation is the opposite of the one in Section 5.2, but with similar goals. For example, if we try to find exploits manually via Exploit-DB in terms of the software OpenSSH 7.7, 28 possible exploits arise. Using AMADEUS-Exploit, we can first detect possible vulnerabilities in the software, i.e., CVE-2018-15473 and CVE-2019-16905. Since AMADEUS-ExploitS keeps the CVE-related exploits indexed, we can directly retrieve that, for the vulnerability CVE-2018-15473, three different exploits can be used (i.e., exploit 45939, exploit 45233, exploit 45210), but there are no exploits available for CVE-2019-16905. Experts can use this operation in the definition of the assessment to quickly identify vulnerabilities related to a set of exploits to prioritise the test to be performed. AMADEUS-Exploit can provide this information without the need to perform reasoning operations.

5.4. Reasoning about configurations

Given details about a specific configuration, i.e., a partial selection of features in the FM such as product, version, operating system, etc., AMADEUS-Exploit can determine whether a specific attack vector affects it by diagnosing the configuration against the FM (detecting errors). This operation takes a configuration and checks whether it is correct or not (valid configuration) and checks its validity on the selected model. In addition, you can point out which products would be a valid configuration (explaining the errors). For example, let us assume the configuration {*debian*, *debian_linux_version*, *debian_linux_version*_8_0}, for the vulnerability CVE-2018-15473, is validated with 16 different valid attack vectors. Both obtaining and checking the FM configurations require the use of the FaMaPy reasoner.

The types of operations mentioned above are ground-breaking proposals in the combination of cybersecurity and software product lines. Moreover, they are only a few examples of the potential use of FMs in cybersecurity, leaving the inclusion of many more functionalities for further work.

6. Evaluation

To conduct the evaluation, we propose the following research questions:

- **RQ1.** Can we analyse the ecosystem of tools and compare them in terms of the use of multiple repositories, scanning, and reasoning capabilities?
- **RQ2.** Can we automatically infer FMs in an acceptable runtime under different scenarios and conditions? Can we compare our vulnerability identification capabilities with other tools?
- **RQ3.** Can reasoning help us in the vulnerability management process in real scenarios?

For guiding the answers to the research questions, we propose to evaluate AMADEUS-Exploit in three different ways: (RQ1.) by analysing vulnerability management tools and their capabilities to position AMADEUS-Exploit; (RQ2.) by evaluating AMADEUS-Exploit in a synthetic scenario and comparing it with other tools, and; (RQ3.) by evaluating AMADEUS-Exploit in a real case by applying reasoning operators to guide the vulnerability management process, from discovery to choice of a set of vulnerabilities to evaluate.

6.1. Analysis of vulnerability management tools

Taking advantage of the importance of vulnerability management, several tools are available. AMADEUS-Exploit has appeared as a solution for providing more functionalities, as analysed in this section.

We have chosen a set of representative tools, both commercial and open source, to carry out a qualitative comparison. We intend to analyse the scanning and reasoning characteristics of these tools in comparison with AMADEUS-Exploit. For this purpose, we have evaluated the following characteristics:

- **Open Source**: the tool is open to the community and can be used free or under a licence.
- **Type of scanning**: the tool allows automatic, manual, or both scanning mechanisms to discover targets.
- **Terms**: the tool allows us to introduce terms for the scanning manual option. These terms could be, e.g., identifiers of CVEs or parts of CPEs, or only identifiers for the targets to be analysed.
- **Databases**: the tool uses vulnerability databases, exploits databases, or both.
- **Reasoning**: the tool can perform any operation or reasoning analysis based on the results of vulnerabilities and exploits.

There are other general but interesting characteristics that may help in vulnerability management task, such as:

- Reporting: The solution provides any kind of dashboard to summarise the information on targets, vulnerabilities, and exploits.
- Prioritisation (Prio.): The solution enables the prioritisation of vulnerabilities and exploits.
- Type of Service: The solution is based on standalone service (S), cloud service (C), or both (B) services.

The results obtained for each tool, including AMADEUS-Exploit, can be seen in Table 4. We can see that the tools most similar to ours are Vuls and Vulscan, with the difference that Vuls and Vulscan do not accept search terms. In fact, no tool allows us to establish the scope of the analysis by establishing the target based on a list of terms. In general, these tools need to point out network targets (IP or domain name) to establish the scope of the analysis. All the tools provide mechanisms for automatically discovering targets, but some tools enable manually defining the target (cf., column Manual) that avoids the discovery. It is important to highlight that the vulnerability and exploits scanning is limited to the services that can be consumed by the exposed ports (e.g., HTTP server exposed in 80 port), but other components like software add-ons, plug-ins, even stand-alone apps, etc. are out of the context of scanning. For example, a web browser application such as Firefox cannot be scanned with tools such as OpenVAS, Vulscan, etc.

Regarding databases, all tools integrate some vulnerability databases, but just a few tools integrate exploit databases. Furthermore, these tools are very limited to retrieving information on vulnerabilities and exploits; for example, they only provide a ranking of vulnerabilities by impact (sorting operations) and do not provide capabilities such as inference of known affected software components provided for vulnerabilities to refine vulnerability assessment and management. In fact, we highlight that the only tool that uses modelling techniques is AMADEUS-Exploit. The use of modelling enables the application of reasoning operations to the results.

6.2. Comparison in vulnerability and exploit identification

In the first evaluation experiment, we propose a synthetic threat scenario that represents real applications and services used in the day-to-day life of organisations. The purpose of this scenario is to include the most representative applications and services that allow web browsing, external connectivity (through ssh tunnels and VPN), and service exposure (application server and content managers). For this purpose, we have used the following applications and services for this scenario: (1) *Mozilla Firefox* (any version) as one of the most used Internet browsers; (2) *Adobe Flash* 32 bits as a plugin for those browsers, which is affected by multiple vulnerabilities; (3) *OpenSSH* 7.7 or higher as a typical solution to enable external connections; (4) *Apache HTTP server* (any version) as a web application server with an *OpenSSL* as SSL/TLS provider to support secure connections; (5) *Nginx* 1.7 as an alternative Web server for web applications; (6) *OpenVPN* 2.3 as a client/server that enables secure external connections; (7) *WordPress* (any version and plugin) as the most widely used content management system on the Internet for the development of web applications. To make the scenario more interesting, we have included some extensions or plugins such as Adobe Flash for a web browser or OpenSSL on the web server, and some versions for applications and services but not for all.

Table 4
Comparison for vulnerability management tools .

Tool	Open source	Type of scanning		Terms	Databases		Reasoning	Reporting	Prio.	Type of service[a]
		Automatic	Manual		Vulnerabilities	Exploits				
InsightVM (Nexpose)		×			×	×		×	×	B
Qualys Cloud Platform		×			×			×	×	C
Qualys VM		×			×			×	×	C
Acunetix by Invicti		×			×			×	×	C
Nessus		×			×	×		×	×	S
Tenable.io		×			×			×	×	C
AlienVault USM		×			×			×	×	C
OpenVAS	×	×	×		×			×	×	B
OpenSCAP	×	×	×		×					S
Vulscan	×	×	×		×	×		×	×	S
Vuls	×	×	×		×	×		×	×	S
AMADEUS-Exploit	×	×	×	×	×	×	×		×	S

[a]S: Standalone, C: Cloud, B: Cloud and Standalone.

Table 5
Number of vulnerabilities and exploits per application and service.

Name	Vulscan		AMADEUS-Exploit			
	CVE	Exploits	CVE	Exploits	Avg. features	Avg. constrains
Mozilla Firefox	–	–	1501	120	261	24
Adobe Flash	–	–	2	–	130	28
OpenSSH	3	13	2	2	48	15
Apache HTTP server	10	36	4	2	141	24
Nginx	6	3	3	–	119	12
OpenVPN	–	–	4	–	74	48
Wordpress	11	36	2416	450	63	13

This set of applications and services represents the target elements to analyse. We have deployed each application and service in a separate container and scanned them using the Vulscan tool. When we did not have the exact version specified in the scenario available, we decided to use the closest version available. The results in terms of the identified vulnerabilities and exploits are given in Table 5. We can observe that there are some applications and services for which Vulscan cannot find vulnerabilities and exploits, such as in the case of Mozilla Firefox or OpenVPN. This table shows the total set of vulnerabilities and exploits that the user must take into account to develop a correct vulnerability management process. This is just the first step; now, it will be up to the stakeholders involved in the vulnerability management process (for example, security testers) to analyse, select and prioritise the set of vulnerabilities and exploits to test from the set provided.

To compare the capabilities of AMADEUS-Exploit in identifying vulnerabilities and exploits regarding Vulscan, we can explore the search by terms. By manually including these applications and services as terms in AMADEUS-Exploit, it automatically extracts 3932 different items corresponding to vulnerabilities and exploits. They correspond to the NVD and Exploit-DB results from 2002 to 2021. In particular, vulnerabilities and exploits are distributed as shown in Table 5. Moreover, the chosen CVEs cover many known affected configurations (CPEs). For example, a single Mozilla Firefox vulnerability, e.g., CVE-2020-6801, gathers approximately 450 CPEs. This gives an idea in terms of possible attack vectors affected by a single vulnerability.

As explained in Section 4, AMADES-Exploit retrieved an FM for each vulnerability. The FMs inferred for the evaluation and the source code for the AMADEUS-Exploit implementation are available,[16] free of charge.

To analyse the key characteristics (features and constraints) of FM, Figs. 7(a) and 7(b) show the number of features and constraints. Additionally, the average number of features and constraints for each application and service is included in Table 5 as complementary details.

To evaluate the extraction capabilities of AMADEUS-Exploit, Fig. 8 shows an analysis of the time required to scrap CVEs and exploits, extract CPEs and exploit information, and generate FMs. In Fig. 8, the dots represent the time consumed in the creation of an FM for each CVE, i.e., the Y-axis is the time spent in the creation, specified in seconds, while on the X-axis each entry represents a CVE. The testing process developed to obtain the performance time runs each phase several times and calculates the average time taken for each phase (in seconds). The generation of FMs requires an appropriate time (sublinear time) in the general case. However, we can observe that there are certain cases where the time is longer, since web scrapping is affected by the Internet response time of vulnerability and exploit repositories, i.e., NVD, VulDB and Exploit-DB.

As mentioned in Section 4, model validation can be seen as a partial metric of correctness (Lopez-Herrejon et al., 2015). That is, a valid FM allows us to create at least one valid attack vector. Therefore, we have validated all FMs (i.e., valid model) to demonstrate that they succeed in obtaining at least one valid product (i.e., a complete selection of features in the FM). In this sense, the 3932 models have been successfully validated.[17] In this context, an attack vector is the selection of features in the FM related to products, vendors, OS, version, running configurations, exploits, etc. that describe a configuration with a vulnerability. Therefore, this configuration can be considered as an attack vector to be evaluated in vulnerability assessment.

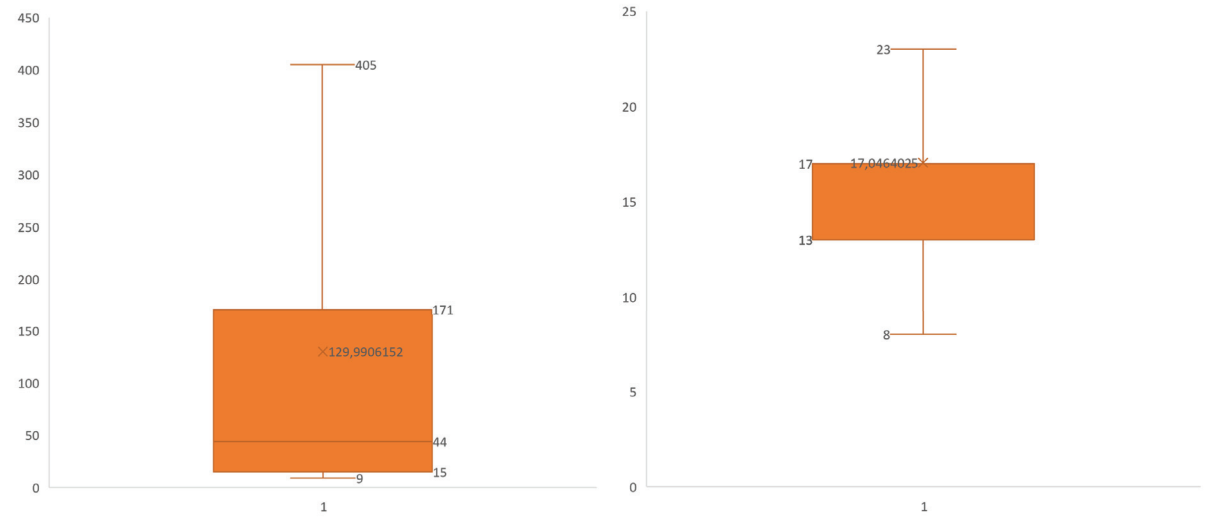
6.3. Evaluation in a real case

The case study included in this section was part of the Security Observatory, a project of the University of Seville.[18] The project aims to analyse and test the security of several systems. The

16 https://doi.org/10.5281/zenodo.7072369

17 FaMaPy valid operation was used.

18 https://sic.us.es/seguridad-tic



(a) Features per CVE.	(b) Constraints per CVE.

Fig. 7. Analysis of features and constraints per CVE.

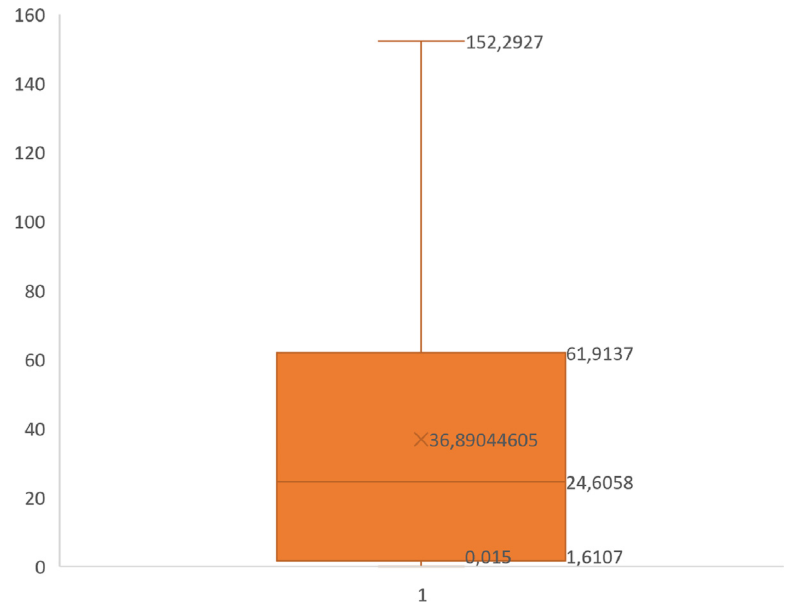


Fig. 8. Time consumed (seconds) in the whole process (Scrapping, Building FM and Including Cross-tree Constrains).

security commission involved in the project provides us with a list of target systems. To evaluate AMADEUS-Exploit, we have chosen one of these real systems accessible through a public domain name.[19] The system behind the domain is a dedicated server that offers certain university services for the community. The system needed to be analysed, and information related to vulnerabilities and exploits was used to find the surface of the exposure and to define a risk treatment plan or mitigation plan for the system under analysis. Although the system is known, the project required a black-box analysis to know of an entry point. For that reason, the analysis aims to demonstrate how to perform a vulnerability management process with AMADEUS-Exploit in a black-box scenario, where the characteristics of the underlying systems and software are completely unknown. Therefore, the purpose is to discover the target elements, identify potential vulnerabilities and exploits, and use reasoning capabilities to choose the appropriate vulnerabilities to cover the entire scenario.

First, we launched AMADEUS-Exploit with automatic capabilities (cf. Section 3.1) to analyse the domain (only by giving the domain as input). Then, these terms have been extracted (automatically): *f5, BIG-IP, load balancer, http*, and *proxy*. Then, AMADEUS-Exploit automatically created FMs for those terms in relation to 167 CVEs and 13 exploits. The full list of CVEs and exploits can be found in Appendix B. Additionally, we have determined the number of attack vectors for all FMs (cf., Appendix B) to evaluate their correctness.

To help the expert in the choice of vulnerabilities, by applying 'Reasoning about exploits' operation (cf., Section 5), we can obtain which vulnerabilities contain at least one exploit and which do not. This information is not available in the NVD database nor in the Exploit-DB. Using this operation as a classification criterion, we can sort the vulnerabilities as shown in Appendix B. Therefore,

[19] For confidentiality restrictions, the specific domain name cannot be provided.

experts can use this information provided by the AMADEUS-Exploit operation to prioritise these 11 vulnerabilities first instead of others, as they provide some resources in the form of exploits. We can see how only 11 out of 167 vulnerabilities have exploits: {CVE-2012-1493, CVE-2008-0265, CVE-2008-0539, CVE-2008-7032, CVE-2014-2927, CVE-2014-2928, CVE-2014-8727, CVE-2012-2997, CVE-2015-4040, CVE-2015-3628, CVE-2018-5511}. Specially, the first one has 3 exploits to be used.

In the lack of more information on systems and software, experts may be interested in knowing those vulnerabilities that affect more known configurations (i.e., represent more attack vectors). Using the operation 'Reasoning about attack vectors' (cf., Section 5), we can obtain all attack vectors for each vulnerability and use that number of vectors as an importance criterion to rank the vulnerabilities to be assessed. In that case, the top 10 vulnerabilities are {CVE-2019-6609, CVE-2018-5507, CVE-2017-6153, CVE-2019-6649, CVE-2018-5535, CVE-2018-5531, CVE-2018-15311, CVE-2018-5534, CVE-2018-5519, CVE-2018-5520}. Recall that the specific characteristics of the real system and software (product, vendor, OS, versions, etc.) are unknown, but we have identified certain aspects related to them. These vulnerabilities recovered the largest number of attack vectors and cover a wide spectrum of configurations, so they may be closer to the system and software in question.

In a detailed analysis of FMs, we can use information related to the type, vendors, and products to choose vulnerabilities with the best attack vectors for our scenario. Applying 'Reasoning about attack vectors' (cf. Section 5) by using certain filters, we have obtained the type, vendor, and products for each vulnerability shown in Appendix B. The number of identified vendors is a maximum of two for each vulnerability. The identified vendors are: $f5$ for 165 CVEs, Jenkins for CVE-2017-6153, CISCO for CVE-2018-5500, $f5$ and Vmware for CVE-2018-5511, $f5$ and Redhat for CVE-2019-6648. For our scenario, we identify '$f5$' as a term. This information can be used by the expert to discard those CVEs that are not directly related to the '$f5$' vendor, hence CVE-2017-6153 and CVE-2018-5500. Analysing the products, we can use a similar operation to obtain information about the products and use this as criteria to order the vulnerabilities that affect more products. In that case, the top 10 vulnerabilities with the most products are {CVE-2016-5022, CVE-2015-8099, CVE-2017-6128, CVE-2014-2927, CVE-2015-5516, CVE-2015-7394, CVE-2016-2084, CVE-2015-3628, CVE-2018-5516, CVE-2016-5021}. Likewise, we can use the terms associated with the products to tune up the search for potential vulnerabilities for further assessment. As mentioned above, we identified the term BIG-IP, and the vulnerabilities containing this information are CVE-2008-7032 and CVE-2014-9342. However, we found no results within the CVEs filtering for the other two terms. Although experts may prioritise the evaluation for these two vulnerabilities, we cannot discard the other CVEs because we do not know if the products referring to CVEs are involved in our scenario.

Finally, we can use the aforementioned criteria to define an adequate vulnerability assessment. Thus, we propose to analyse first the CVEs with exploits and associated with the identifier vendor ($f5$) and product (BIG-IP), and then the CVEs that cover more attack vectors. The result is collected in Table 6. Of course, we cannot ignore the rest of the vulnerabilities, but those listed are the ones that best fit the scenario requirements identified by AMADEUS-Exploit and the different operations used. However, other reasoning queries can be used to further refine this proposed list.

The results were reported to the security commission and rapidly transferred to those administrators responsible for the system. The actions taken by the security commission were to prioritise the analysis of the systems according to the vulnerabilities and exploits discovered to determine possible actions to take (e.g., patches or updates) for the system. Now, the administrators responsible for the system must analyse the results in Table 6 to evaluate the vulnerabilities and exploits discovered.

7. Related work

System vulnerability scanning is a well-known problem to manage system risks (Sterlini et al., 2020; Jimenez et al., 2019). To reduce risks, vulnerabilities must be collected and analysed to identify potential attacks and define adequate assessments (security testing). There are several works on these topics in the literature. Traditionally, vulnerability location and extraction focus on the analysis of source code or repositories in different directions, e.g., Perl et al. (2015), Jimenez et al. (2018) and Cho et al. (2011). There are approaches based on static analysis of the code (Perl et al., 2015), and others based on symbolic and dynamic analysis (Cho et al., 2011). In terms of repository analysis, Neuhaus et al. (2007) analyse the Mozilla vulnerability repository to provide a solution to predict the most prominent components that may be vulnerable. Jimenez et al. (2018) present VulData7, a framework for automatically obtaining a dataset of NVD and Git vulnerabilities for specific systems. VulData7 allows to align vulnerabilities with possible fixes (patches), if any. From another perspective, Sanguino and Uetz (2017) provide a tool called IVA that automates the search process for potential vulnerabilities in software products installed in organisations. This approach relies on an asset inventory, but our approach is decoupled from the infrastructure, as AMADEUS-Exploit supports scanning tools such as Nmap, which allows us to discover assets and services automatically without the need for an inventory.

Regarding vulnerability assessment, Dass and Namin (2020) and Murthy and Shilpa (2018) present solutions to obtain a set of vulnerabilities to be used in security testing. Dass and Namin (2020) propose a genetic algorithm approach to generate Common Vulnerability Scoring System (CVSS) vectors to find the best set of vulnerabilities for adequate security testing. However, the main drawback is that, after generating the CVSS vector, they have to search the CVE repository to find the vulnerabilities to use. On the contrary, Murthy and Shilpa (2018) focus on the coverage of the security test, applying the concepts of pairwise testing. Thus, they assume that security testing is defined and only propose a coverage criterion to determine when the security testing process should stop by reducing the number of tests to be performed.

In the literature, formal (Mulwad et al., 2011) and pseudo-formal (Emeka and Liu, 2018) structures have been used to identify vulnerabilities. In the contribution by Mulwad et al. (2011), a term ontology is created to identify future vulnerability terms to query the NVD. Jia et al. (2018) use ML techniques on a cybersecurity knowledge base to extract entities and build an ontology to obtain a cybersecurity knowledge base. Then, the calculation of formulas and the use of the path-ranking algorithm allow for the derivation of new rules. The use of the knowledge base implies keeping this structure updated in case new terminology appears and not analysing in-depth the set of vulnerabilities of a system, providing less customised solutions. However, our approach focusses on analysing the vulnerability of a system from its components, configurations, or terms introduced by experts.

Other approaches, such as the contribution by Weerawardhana et al. (2014), perform information extraction from vulnerability databases, such as NVD (Anon., 2020a), using Natural Language and ML techniques. This can be further used by applications, such as vulnerability scanners and security monitoring tools. In the contribution by Mulwad et al. (2011), a framework is presented to detect and extract information about vulnerabilities and attacks from Web text. The use of exploits to analyse potential vulnerabilities has been an important area of research (Jacobs

Table 6
List of 20-top vulnerabilities considered for evaluation.

#	Vulnerability	N° of attack vectors	Exploits (EDBID)	Vendors	Products	Versions
1	CVE-2012–1493	848	{exploit_19099, exploit_19091, exploit_19064}	1	5	14
2	CVE-2008–0265	2	exploit_31024	1	1	1
3	CVE-2008–0539	2	exploit_31065	1	1	1
4	CVE-2008–7032	2	exploit_31133	1	1	1
5	CVE-2014–2927	390	exploit_34465	1	19	2
6	CVE-2014–2928	172	exploit_34927	1	9	1
7	CVE-2014–8727	58	exploit_35222	1	1	14
8	CVE-2012–2997	8	exploit_38233	1	1	18
9	CVE-2015–4040	796	exploit_38448	1	14	1
10	CVE-2015–3628	194	exploit_38764	1	18	9
11	CVE-2014–9342	2	–	1	1	14
12	CVE-2019–6609	26,450	–	1	14	14
13	CVE-2018–5507	1,998	–	1	13	13
14	CVE-2019–6649	1,314	–	1	14	14
15	CVE-2018–5535	1,176	–	1	13	13
16	CVE-2018–5531	1,168	–	1	13	13
17	CVE-2018–15311	924	–	1	13	1
18	CVE-2018–5534	870	–	1	13	13
19	CVE-2018–5519	868	–	1	13	13
20	CVE-2018–5520	868	–	1	13	13

et al., 2020). Previous work has analysed how exploits can be selected to reduce the risk produced by vulnerabilities (Suciu et al., 2021; Bozorgi et al., 2010). However, extracting and integrating them into a single model in which both vulnerabilities and exploits can be combined has been a challenge. Kenner et al. (2020) pointed out this combination as necessary and it has been achieved innovatively in our AMADEUS-Exploit contribution.

This introduces the use of FM to manage the variability of vulnerability and configuration of systems in a rational way. There are previous works in the literature that use FM to represent system vulnerabilities (ter Beek et al., 2020; Kenner et al., 2020). In the contribution of ter ter Beek et al. (2020), the authors use variability techniques to define the attack-defence scenario. However, Kenner et al. (2020) built synthetic attack scenarios based on vulnerability analysis. Nevertheless, the most widely used methodology to obtain these models is still manual. In contrast to this, this paper provides a novel automated method capable of outperforming existing human-oriented ones. We use FMs to define a consistent and homogeneous structure representing configurations with vulnerabilities, and AMADEUS provides a solution that covers all phases of the process, from vulnerability extraction through reasoning to the creation of FMs.

In the SPL area, the extraction of FMs from existing systems has already been addressed by reverse engineering techniques. These techniques are applicable in many tasks, but are mainly used to determine features, feature restrictions, and to generate complete feature models. There are several techniques applied to reverse engineering in SPL: search-based techniques (Lopez-Herrejon et al., 2015); using propositional logic (Czarnecki and Wasowski, 2007); natural language requirements (Weston et al., 2009); ad-hoc algorithms (Haslinger et al., 2013; Acher et al., 2012; Haslinger et al., 2011); and, configuration scripts (She et al., 2011). Most reverse engineering approaches focus on the application of different topics of software engineering. However, they are far from the particular characteristics of cybersecurity and vulnerability issues, so in this paper we have considered the extraction of FMs from vulnerabilities.

8. Threats to validity

Even though the experiments presented in this paper provide pieces of evidence for validation, we discuss the different threats to validity that affect our approach:

1. *Internal validity*. Although the evaluation performed on thousands of CVEs demonstrates that there are no errors, the use of external databases may contain uncontrolled errors to us. The analysis done in the evaluation reveals different properties of FMs, vulnerabilities, known affected configurations, and exploits. However, there might be characteristics that are not revealed, e.g., the most prominent vulnerable feature. For instance, we can infer (indirect) relations between features and exploits that are not directly extracted from the exploit databases. The main benefit of using FMs is the opportunity to use automatic analysis techniques over plain information that is scattered through different and heterogeneous repositories. However, the use of FMs introduces a disadvantage to experts in learning the logic under the FMs and automatic analysis. Hence, our approach tries to fulfil this gap by providing shortcuts.
2. *External validity*. Although the evaluation covers many CVEs and realistic scenarios, we cannot generalise the conclusions for any scenario. AMADEUS-Exploit is useful for different security stakeholders to reveal reasoning capabilities on vulnerability information that currently are not exploited, but it would be necessary to carry out an external validation with experts.
3. *Conclusion validity*. Anyone can replicate the experiments, since we provide a repository with AMADEUS-Exploit source code and the models extracted for the evaluation.

9. Concluding remarks & future directions

Vulnerability management is essential to avoid security risks. However, the large amount of information in the different databases, the high complexity and variability of system configurations, and the need for reasoning to assist the vulnerability management process make it very difficult to obtain an efficient solution. Therefore, it is essential to provide models that collect information from vulnerability and exploit databases, and provide automatic analysis mechanisms to support the vulnerability management process.

Previous solutions have faced this challenge, but the AMADEUS-Exploit framework proposes a holistic solution to bridge the gap between vulnerability identification and assessment through the application of feature models. It is an extension of the AMADEUS framework presented in a previous work (Varela-Vaca et al., 2020) that proposes a methodology to automatically generate FMs and use them in automatic reasoning to support vulnerability management, integrating some vulnerability and exploit repositories. In addition to this functionality, the new AMADEUS-Exploit framework addresses aspects

Table B.7
List of vulnerabilities, exploits, and known vulnerable configurations.

Vulnerability	N° of attack vectors	Exploits (EDB-ID)	Type	Vendors	Products	Versions
CVE-2012–1493	848	{exploit_19099 exploit_19091 exploit_19064}	App - OS - Hw	1	5	14
CVE-2008–0265	2	exploit_31024	OS	1	1	1
CVE-2008–0539	2	exploit_31065	OS	1	1	1
CVE-2008–7032	2	exploit_31133	Hw	1	1	1
CVE-2014–2927	390	exploit_34465	App	1	19	2
CVE-2014–2928	172	exploit_34927	App	1	9	1
CVE-2014–8727	58	exploit_35222	App	1	1	14
CVE-2012–2997	8	exploit_38233	App	1	1	18
CVE-2015–4040	796	exploit_38448	App	1	14	1
CVE-2015–3628	194	exploit_38764	App	1	18	9
CVE-2018–5511	114	exploit_46600	App	2	16	13
CVE-1999–1550	2	–	OS	1	1	5
CVE-2005–2245	10	–	OS	1	1	1
CVE-2008–1503	2	–	OS	1	1	19
CVE-2008–6474	2	–	OS	1	1	9
CVE-2009–4420	22	–	App - Hw	1	3	1
CVE-2012–3000	98	–	App - Hw	1	10	16
CVE-2013–0150	22	–	App - Hw	1	2	1
CVE-2013–5975	12	–	App - Hw	1	1	1
CVE-2013–5976	20	–	App - Hw	1	1	1
CVE-2013–6016	172	–	App	1	9	1
CVE-2013–6024	42	–	App - Hw	1	3	3
CVE-2013–7408	10	–	App	1	1	10
CVE-2014–3959	56	–	App	1	14	1
CVE-2014–4023	300	–	App - Hw	1	14	9
CVE-2014–4024	312	–	App	1	13	3
CVE-2014–6031	352	–	App	1	14	1
CVE-2014–8730	272	–	App	1	14	14
CVE-2014–9326	78	–	App	1	9	13
CVE-2014–9342	2	–	App	1	1	14
CVE-2015–1050	72	–	App	1	1	14
CVE-2015–4637	14	–	App	1	4	1
CVE-2015–4638	114	–	App	1	10	4
CVE-2015–5058	62	–	App	1	12	10
CVE-2015–5516	386	–	App	1	18	12
CVE-2015–6546	202	–	App	1	13	18
CVE-2015–7394	252	–	App	1	18	13
CVE-2015–8021	142	–	App	1	13	18
CVE-2015–8022	236	–	App	1	14	13
CVE-2015–8098	14	–	App	1	1	14
CVE-2015–8099	218	–	App	1	21	1
CVE-2015–8240	56	–	App	1	10	21
CVE-2016–1497	272	–	App	1	14	10
CVE-2016–2084	212	–	App	1	18	14
CVE-2016–3686	36	–	App	1	2	18
CVE-2016–3687	20	–	App	1	2	2
CVE-2016–4545	18	–	App	1	9	2
CVE-2016–5020	278	–	App	1	14	9
CVE-2016–5021	146	–	App	1	16	14
CVE-2016–5022	286	–	App	1	22	16
CVE-2016–5023	100	–	App	1	13	22
CVE-2016–5024	54	–	App	1	10	13
CVE-2016–5700	134	–	App	1	8	10
CVE-2016–5736	188	–	App	1	15	8
CVE-2016–5745	32	–	App	1	1	15
CVE-2016–6249	160	–	App	1	11	1
CVE-2016–6876	256	–	App	1	14	11
CVE-2016–7467	12	–	App	1	1	14
CVE-2016–7468	130	–	App	1	10	1

(*continued on next page*)

such as the subsequent extraction of exploits to enable vulnerability assessment, the incorporation of new vulnerability and exploit repositories, and the improvement of reasoning models. AMADEUS-Exploit integrates all functionality into a single FM model, including multiple reasoning operations to facilitate the task of vulnerability management from identification to choosing the most relevant vulnerabilities to assess.

The new framework has been evaluated in three different ways: (1) being compared with other vulnerability management tools concerning certain capabilities for the identification and reasoning of vulnerabilities and exploits; (2) in a synthetic case in which almost 4000 FMs have been extracted from typical applications and services under high threat; and (3) being applied in real case scenario obtaining a set of vulnerabilities and analysing their characteristics to choose the most relevant ones to be considered for assessment according to the system and software identified.

As future work directions, AMADEUS-Exploit has many potential extensions: it can be extended (1) with decision-making techniques for attack vector generation; (2) using FMs to detect inconsistencies in vulnerability repositories; (3) integrating

Table B.7 (*continued*).

Vulnerability	N° of attack vectors	Exploits (EDB-ID)	Type	Vendors	Products	Versions
CVE-2016–7472	4	–	App	1	1	10
CVE-2016–7474	246	–	App	1	14	1
CVE-2016–7476	136	–	App	1	10	14
CVE-2016–9245	60	–	App	1	10	10
CVE-2016–9250	268	–	App	1	14	10
CVE-2016–9251	80	–	App	1	10	14
CVE-2016–9252	312	–	App	1	14	10
CVE-2016–9253	60	–	App	1	10	14
CVE-2016–9256	80	–	App	1	10	10
CVE-2016–9257	8	–	App	1	1	10
CVE-2017–0301	22	–	App	1	1	1
CVE-2017–0302	10	–	App	1	1	1
CVE-2017–0303	184	–	App	1	8	1
CVE-2017–0305	4	–	App	1	1	8
CVE-2017–6128	206	–	App	1	21	1
CVE-2017–6129	4	–	App	1	1	21
CVE-2017–6131	90	–	App	1	9	1
CVE-2017–6132	300	–	App	1	11	9
CVE-2017–6133	112	–	App	1	10	11
CVE-2017–6134	528	–	App	1	11	10
CVE-2017–6135	22	–	App	1	11	11
CVE-2017–6136	124	–	App	1	11	11
CVE-2017–6137	100	–	App	1	11	11
CVE-2017–6138	124	–	App	1	11	11
CVE-2017–6139	4	–	App	1	1	11
CVE-2017–6141	48	–	App	1	8	1
CVE-2017–6142	18	–	App	1	1	8
CVE-2017–6143	82	–	App	1	2	1
CVE-2017–6144	6	–	App	1	1	2
CVE-2017–6145	80	–	App	1	10	1
CVE-2017–6147	40	–	App	1	10	10
CVE-2017–6148	368	–	App	1	8	10
CVE-2017–6150	146	–	App	1	10	8
CVE-2017–6151	26	–	App	1	13	10
CVE-2017–6152	4	–	App	1	1	13
CVE-2017–6153	1718	–	App	1	1	1
CVE-2017–6154	20	–	App	1	1	1
CVE-2017–6155	452	–	App	1	11	1
CVE-2017–6156	438	–	App	1	13	11
CVE-2017–6157	206	–	App	1	8	13
CVE-2017–6158	534	–	App	1	13	8
CVE-2017–6159	86	–	App	1	8	13
CVE-2017–6160	52	–	App	1	2	8
CVE-2017–6161	344	–	App	1	11	2
CVE-2017–6162	212	–	App	1	8	11
CVE-2017–6163	234	–	App	1	8	8
CVE-2017–6164	352	–	App	1	13	8
CVE-2017–6165	220	–	App	1	11	13
CVE-2017–6167	112	–	App	1	10	11
CVE-2017–6169	18	–	App	1	1	10
CVE-2018–15311	924	–	App	1	13	1
CVE-2018–15312	778	–	App	1	13	13
CVE-2018–15313	62	–	App	1	1	13
CVE-2018–15314	62	–	App	1	1	1
CVE-2018–15315	778	–	App	1	13	1
CVE-2018–15316	42	–	App	1	2	13
CVE-2018–15332	276	–	App	1	2	2
CVE-2018–5500	4	–	OS	1	1	2
CVE-2018–5501	416	–	App	1	13	1
CVE-2018–5502	278	–	App	1	13	13
CVE-2018–5503	40	–	App	1	1	13
CVE-2018–5504	436	–	App	1	13	1
CVE-2018–5505	20	–	App	1	2	13
CVE-2018–5506	556	–	App	1	13	2
CVE-2018–5507	1998	–	App	1	13	13
CVE-2018–5508	52	–	App	1	1	13
CVE-2018–5509	288	–	App	1	8	1
CVE-2018–5510	156	–	App	1	13	8
CVE-2018–5512	168	–	App	1	13	13
CVE-2018–5513	824	–	App	1	13	13
CVE-2018–5514	168	–	App	1	13	13
CVE-2018–5515	168	–	App	1	13	13
CVE-2018–5516	858	–	App	1	16	16

(*continued on next page*)

Table B.7 (*continued*).

Vulnerability	N° of attack vectors	Exploits (EDB-ID)	Type	Vendors	Products	Versions
CVE-2018-5517	168	–	App	1	13	13
CVE-2018-5518	352	–	App	1	13	13
CVE-2018-5519	868	–	App	1	13	13
CVE-2018-5520	868	–	App	1	13	13
CVE-2018-5521	558	–	App	1	13	13
CVE-2018-5522	638	–	App	1	13	13
CVE-2018-5523	614	–	App	1	14	14
CVE-2018-5524	328	–	App	1	11	11
CVE-2018-5525	702	–	App	1	13	13
CVE-2018-5526	14	–	App	1	1	1
CVE-2018-5529	244	–	App	1	2	2
CVE-2018-5530	484	–	App	1	9	9
CVE-2018-5531	1168	–	App	1	13	13
CVE-2018-5532	760	–	App	1	13	13
CVE-2018-5533	638	–	App	1	13	13
CVE-2018-5534	870	–	App	1	13	13
CVE-2018-5535	1176	–	App	1	13	13
CVE-2018-5536	56	–	App	1	1	1
CVE-2018-5537	806	–	App	1	10	10
CVE-2018-5538	118	–	App	1	4	4
CVE-2018-5539	52	–	App	1	1	1
CVE-2018-5540	118	–	App	1	5	5
CVE-2018-5541	48	–	App	1	1	1
CVE-2018-5542	866	–	App	1	13	13
CVE-2018-5543	20	–	App	1	1	1
CVE-2018-5544	48	–	App	1	1	1
CVE-2018-5546	48	–	App	1	2	2
CVE-2018-5547	6	–	App	1	1	1
CVE-2019-6595	46	–	App	1	1	1
CVE-2019-6609	26450	–	App	1	14	14
CVE-2019-6648	3	–	App	2	1	1
CVE-2019-6649	1314	–	App	1	14	14
CVE-2019-6650	106	–	App	1	1	1
CVE-2019-6665	104	–	App	1	4	4
CVE-2020-5944	2	–	App	1	1	1

other analysis tools (e.g., Lynis); (4) integrating other vulnerability databases (e.g., CNVD, IBM X-Force, or US-Cert), etc. And last but not least, the AMADEUS-Exploit process needs to be validated by external experts to make a strong and practical validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

https://zenodo.org/record/7072369.

Acknowledgements

This work has been funded by the projects COPERNICA (P20_01224), METAMORFOSIS (US-1381375), and AETHER-US PID2020-112540RB-C44 funded by MCIN/AEI/10.13039/501100-011033.

Appendix A. Feature model construction algorithms

For a better understanding of each part of the algorithm, some concepts are introduced. Let L be a list of n configurations (CPEs) and L_{EXP} a list of exploits for a given CVE. L could be considered as a composition of two smaller lists, $L_{VUL} = \{cpe_1, cpe_2, \dots, cpe_j\}$ and $L_{RC} = \{cpe_{j+1}, cpe_{j+2}, \dots, cpe_n\}$, containing vulnerable configurations and running configurations (i.e., execution environments), respectively. In the same way, L_{EXP} is the list of exploits $\{exp_1, exp_2, exp_3, \dots, exp_m\}$, if any. The content of both lists with respect to the running example in Table 3 would be: $L_{VUL} = \{cpe_1, cpe_2, cpe_3, cpe_4, cpe_5\}$, $L_{RC} = \{cpe_6\}$, and $L_{EXP} = \{exp_1, exp_2, exp_3\}$.

Algorithm 1: Build unrestricted FM from a CVE.

```
  Input: CVE-ID, $L : \{cpe_1, cpe_2, \dots, cpe_n\}$,
         $L_{EXP} : \{exp_1, exp_2, \dots, exp_m\}$
  Result: fm: Feature Model
1 $listOfFM_{VUL-EXP} \leftarrow \{\}$; $listOfFM_{RC} \leftarrow \{\}$; $fm \leftarrow \{\}$;
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 $listOfFM_{VUL-EXP} \leftarrow createSubFMs(L_{VUL}, L_{EXP})$;
7 /* Merge the FMs from Exploits and CPEs*/
8 for $fm_{vul_i} \in listOfFM_{VUL-EXP}$ do
9     $merge(fm,$ CVE-ID$, fm_{vul_i})$;
10 end
11 /* Create a branch for part in the FM */
12 $children(fm, 'type', getTypes(L))$;
13 /* Include the Running Configurations in FM*/
14 if $|L_{RC}| > 0$ then
15     /* Create a node that will contain all RCs */
16     $opt(fm,$ CVE-ID, "rc"$)$;
17     /* Create an FM for each RC */
18     for $rc_i \in L_{RC}$ do
19         /* Retrieve list of sub-models by Algorithm 2 */
           $listOfFM_{VUL-EXP} \leftarrow createSubFMs(getRC(L_{RC}, rc_i))$;
20         /* Merge FMs together */
21         for $fm_{rc_i} \in listOfFM_{RC}$ do
22             $merge(fm, rc_i, fm_{rc_i})$;
23         end
24     end
25 end
```

Derived from the special characteristics mentioned of the CPE attributes (*product, vendor and part*), some functions are defined below:

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

Other operators have been defined when developing the algorithms. These operators are grouped into two categories:

1. Operators to get information from a list L of CPEs:
    - $vul(L)$ takes a list L of CPEs as input and returns the list of vulnerable configurations, L_{VUL}. For the example, *vul(L) = {cpe*1*, cpe*2*, cpe*3*, cpe*4*, cpe*5*}*.
    - $rc(L)$ takes a list L of CPEs as input and returns a map (list of pairs key$\rightarrow$ value) indexing the running environment configurations in L_{RC}. For the example, *rc(L) = ['rc*$_1$*'* $\rightarrow$ {*cpe*6}].
    - $getRC(L, rc_i)$ returns a list of CPEs associated to the rc_i in the running configurations L_{RC}. 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 already exist, and a mandatory relationship between them.
    - $opt(FM, f_1, f_2)$ creates two new features if they do not already exist, and an optional relation between them.
    - $xor(FM, f, A)$ creates a new feature f in *FM* if it does not already 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 already 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 already 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., $children(FM, f, R)$.

A set of operators is introduced to facilitate understanding of Algorithm 3:

Algorithm 2: Create sub-FMs.

```
Input: L : {$cpe_1, cpe_2, cpe_3, \ldots, cpe_n$},
       $L_{EXP}$ : {$exp_1, exp_2, \ldots, exp_m$}
Result: listOfFM: List of FMs
1  listOfFM ← {};
2  /* Create new FM representing each vendor */
3  for $v_i \in getVendors(L)$ do
4      fm ← {};
5      /* Include all vendors as root feature */
6      $createRootF(fm, v_i)$;
7      for $p_j \in getProducts(L, v_i)$ do
8          for $a_k \in getAttributes(L, p_j)$ do
9              /* Create features and relations 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         /* Create features and relations between the product $p_k$
              and their attributes */
13         $children(fm, p_j, getAttributes(L, p_j))$;
14     end
15     /* Create features and relations representing the vendor $v_i$
          and the products */
16     $children(fm, v_i, getProducts(L, v_i))$;
17     listOfFM ← fm;
18 end
19 /* Create new FM representing the exploits */
20 $fm_{exp}$ ← {};
21 if $|L_{EXP}| > 0$ then
22     /* Create root feature for exploits */
23     $createRootF(fm_{exp}, 'exploits')$;
24     for $exp_i \in L_{EXP}$ do
25         /* Create features and relations between 'exploits' and
              concrete exploit $exp_i$ */
26         $children(fm_{exp}, 'exploits', exp_i)$;
27     end
28 end
29 listOfFM ← $fm_{exp}$;
```

- $getLeaves(FM)$ takes a feature model *FM*, and returns the set of leaves of *FM*, which 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 L of CPEs and a value f which represents a feature, returning *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 L of CPEs and a value f which represents a feature, returning a list of values that are siblings of it and belong 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 L of CPEs and a value f which represents a feature, returning 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 $getRelatedRC(L, 'es')$ = {'$rc1$'}.
- $getProducts(L)$ takes a list L of CPEs, and returns a list of values that represent the products in L. For the example, $getProducts(L)$ = {'*ocelot*', '*civet*', '*lynx*' }.
- $getRelatedType(L, p)$ takes a list L of CPEs, a value f representing a product feature, and returns a list of values that represent the related type in L. For the example, $getRelatedType(L, 'civet')$ = {*application*} or $getRelatedType(L, 'ocelot')$ = {*application*}.
- $const(FM, f, C)$ 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: Create Cross-tree Constraints for unrestricted FM.

```
Input: $L : \{cpe_1, cpe_2, cpe_3, \ldots, cpe_n\}$, $FM$: Feature Model
Result: $FM$ : Feature Model with Constraints
/* For each product in L */
$products \leftarrow getProducts(L)$;
for $p_i \in products$ do
    /* Include a new cross-tree for each relative Type */
    $types \leftarrow getRelatedType(FM, p_i)$;
    $const(FM,\ types,\ p_i)$;
end
/* Obtain the FM leaves */
$leaves \leftarrow getLeaves(FM)$;
/* For each leaf */
for $leaf \in leaves$ do
    if $\neg isRC(L, leaf)$ then
        /* Get other leaves related to the same CPE */
        $listSiblings \leftarrow getSiblings(L,\ leaf)$;
        /* Include a new cross-tree for each relative leaf */
        for $s_i \in listSiblingsAttr$ do
            $const(FM,\ leaf,\ s_i)$;
        end
        /* Get RC related to the leaf */
        $listRelatedRC \leftarrow getRelatedRC(L,\ leaf)$;
        /* Include a new cross-tree for each relative RC */
        for $rc_i \in listRelatedRC$ do
            $const(FM,\ leaf,\ rc_i)$;
        end
    end
end
```

Appendix B. Information about vulnerabilities, exploits, and known vulnerable configurations

See Table B.7.

References

Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P., 2012. On extracting feature models from product descriptions. In: VAMOS. pp. 45–54.

Acher, M., Collet, P., Lahire, P., France, R.B., 2013. FAMILIAR: A domain-specific language for large scale management of feature models. Sci. Comput. Program. 78 (6), 657–681.

Anon., 2020a. National vulnerability database. Available from NIST. URL https://nvd.nist.gov/.

Anon., 2020b. Common vulnerability scoring system SIG. Available from FIRST. URL https://www.first.org/cvss/.

Anon., 2020c. Common vulnerability exposure. Available from MITRE. URL http://cve.mitre.org/.

Anon., 2020d. The community-driven vulnerability database. Available from VULDB. URL https://vuldb.com/.

Anon., 2021. Exploit database. URL https://www.exploit-db.com/. Offensive Security.

Backes, M., Hoffmann, J., Künnemann, R., Speicher, P., Steinmetz, M., 2017. Simulated penetration testing and mitigation analysis. CoRR arXiv:1705.05088.

ter Beek, M.H., Legay, A., Lafuente, A.L., Vandin, A., 2020. Variability meets security: Qantitative security modeling and analysis of highly customizable attack scenarios. VAMOS '20, Association for Computing Machinery, New York, NY, USA.

Benavides, D., Segura, S., Ruiz-Cortés, A., 2010. Automated analysis of feature models 20 years later. Inf. Syst. 35 (6), 615–636.

Benavides, D., Trinidad, P., Cortés, A.R., Segura, S., 2013. Fama. In: Capilla, R., Bosch, J., Kang, K.-C. (Eds.), Systems and Software Variability Management. Springer, Berlin, Heidelberg, pp. 163–171.

Bozorgi, M., Saul, L., Savage, S., Voelker, G.M., 2010. Beyond heuristics: Learning to classify vulnerabilities and predict exploits. In: Proceedings of the Sixteenth ACM Conference on Knowledge Discovery and Data Mining. KDD-2010, pp. 105–113.

Cho, C.Y., Babić, D., Poosankam, P., Chen, K.Z., Wu, E.X., Song, D., 2011. MACE: Model-inference-assisted concolic exploration for protocol and vulnerability discovery. In: Proceedings of the 20th USENIX Conference on Security. SEC '11, USENIX Association, USA, p. 10.

Clements, P., Northrop, L., 2002. Software Product Lines. Addison-Wesley Boston.

Czarnecki, K., Wasowski, A., 2007. Feature diagrams and logics: There and back again. In: 11th International Software Product Line Conference. SPLC 2007, IEEE, pp. 23–34.

Dass, S., Namin, A.S., 2020. Vulnerability coverage for adequacy security testing. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. SAC '20, Association for Computing Machinery, New York, NY, USA, pp. 540–543.

Emeka, B.O., Liu, S., 2018. Assessing and extracting software security vulnerabilities in SOFL formal specifications. In: 2018 International Conference on Electronics, Information, and Communication. ICEIC, pp. 1–4.

Engebretson, P., 2013. The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy. Elsevier.

Foreman, P., 2009. Vulnerability Management. Auerbach Publications.

Foreman, P., 2019. Vulnerability Management, second ed Auerbach Publications, Milton.

Galindo, J.A., Benavides, D., 2020. A python framework for the automated analysis of feature models: A first step to integrate community efforts. SPLC '20, Association for Computing Machinery, New York, NY, USA, pp. 52–55.

Gawron, M., Cheng, F., Meinel, C., 2015. Automatic detection of vulnerabilities for advanced security analytics. In: 2015 17th Asia-Pacific Network Operations and Management Symposium. APNOMS, pp. 471–474.

Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A., 2011. Reverse engineering feature models from programs' feature sets. In: 2011 18th Working Conference on Reverse Engineering. IEEE, pp. 308–312.

Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A., 2013. On extracting feature models from sets of valid feature combinations. In: FASE. Springer, pp. 53–67.

Jacobs, J., Romanosky, S., Adjerid, I., Baker, W., 2020. Improving vulnerability remediation through better exploit prediction. J. Cybersecur. 6 (1).

Jia, Y., Qi, Y., Shang, H., Jiang, R., Li, A., 2018. A practical approach to constructing a knowledge graph for cybersecurity. Engineering 4 (1), 53–60, Cybersecurity.

Jimenez, M., Le Traon, Y., Papadakis, M., 2018. [Engineering Paper] enabling the continuous analysis of security vulnerabilities with VulData7. In: 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation. SCAM, pp. 56–61.

Jimenez, M., Rwemalika, R., Papadakis, M., Sarro, F., Traon, Y.L., Harman, M., 2019. The Importance of Accounting for Real-World Labelling When Predicting Software Vulnerabilities. ACM, pp. 695–705.

Karataş, A.S., Oğuztüzün, H., 2013. From extended feature models to constraint logic programming. Sci. Comput. Program. 78 (12), 2295–2312, Special Section on SPLC 2010 and FSEN 2011.

Kenner, A., Dassow, S., Lausberger, C., Krüger, J., Leich, T., 2020. Using variability modeling to support security evaluations: Virtualizing the right attack scenarios. In: VaMoS '20. pp. 10:1–10:9.

Kuehn, P., Bayer, M., Wendelborn, M., Reuter, C., 2021. OVANA: An approach to analyze and improve the information quality of vulnerability databases. In: The 16th International Conference on Availability, Reliability and Security. ARES 2021, Association for Computing Machinery, New York, NY, USA.

Lopez-Herrejon, R.E., Linsbauer, L., Galindo, J.A., Parejo, J.A., Benavides, D., Segura, S., Egyed, A., 2015. An assessment of search-based techniques for reverse engineering feature models. J. Simple Syst. 103, 353–369.

Mazo, R., Muñoz-Fernández, J.C., Rincón, L., Salinesi, C., Tamura, G., 2015. VariaMos: An extensible tool for engineering (dynamic) product lines. SPLC '15, Association for Computing Machinery, New York, NY, USA, pp. 374–379.

Mendonca, M., Branco, M., Cowan, D., 2009. S.P.L.O.T.: Software product lines online tools. OOPSLA '09, Association for Computing Machinery, New York, NY, USA, pp. 761–762.

Morrison, P.J., Pandita, R., Xiao, X., Chillarege, R., Williams, L., 2018. Are vulnerabilities discovered and resolved like other defects? Empir. Softw. Eng. 23 (3), 1383–1421.

Mulwad, V., Li, W., Joshi, A., Finin, T., Viswanathan, K., 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, pp. 257–260.

Murthy, P.V.R., Shilpa, R.G., 2018. Vulnerability coverage criteria for security testing of web applications. In: 2018 International Conference on Advances in Computing, Communications and Informatics. ICACCI, pp.489–494.

Neuhaus, S., Zimmermann, T., Holler, C., Zeller, A., 2007. Predicting vulnerable software components. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. CCS '07, Association for Computing Machinery, New York, NY, USA, pp. 529–540.

Oyler, A., Saiedian, H., 2016. Security in automotive telematics: A survey of threats and risk mitigation strategies to counter the existing and emerging attack vectors. Secur. Commun. Netw. 9 (17), 4330–4340.

Palmaers, T., 2021. Implementing a Vulnerability Management Process. Tech. Rep., SANS Institute.

Parmelee, M., Booth, H., Waltermire, D., Scarfone, K., 2011. Common Platform Enumeration: Name Matching Specification Version 2.3. NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD.

Perez, S.M., Cosentino, V., Cabot, J., 2017. Model-based analysis of java EE web security misconfigurations. Comput. Lang. Syst. Struct. 49, 36–61.

Perl, H., Dechand, S., Smith, M., Arp, D., Yamaguchi, F., Rieck, K., Fahl, S., Acar, Y., 2015. VCCFinder: Finding potential vulnerabilities in open-source projects to assist code audits. CCS '15, Association for Computing Machinery, New York, NY, USA, pp. 426–437.

Roos Frantz, F., Benavides Cuevas, D.F., Ruiz Cortés, A., 2009. Feature model to orthogonal variability model transformation towards interoperability between tools. In: Kiss Workshop@ ASE2009, Auckland, New Zealand.

Sanguino, L.A.B., Uetz, R., 2017. Software vulnerability analysis using CPE and CVE. arXiv:1705.05347.

Schmitt, A., Rock, G., Bettinger, C., 2018. Glencoe – A tool for specification, visualization and formal analysis of product lines.

Seidl, C., Winkelmann, T., Schaefer, I., 2016. A software product line of feature modeling notations and cross-tree constraint languages. In: Oberweis, A., Reussner, R. (Eds.), Modellierung 2016. Gesellschaft für Informatik e.V., Bonn, pp. 157–172.

Shah, S., Mehtre, B.M., 2015. An overview of vulnerability assessment and penetration testing techniques. J. Comput. Virol. Hacking Tech. 11 (1), 27–49.

She, S., Lotufo, R., Berger, T., Waşowski, A., Czarnecki, K., 2011. Reverse engineering feature models. In: ICSE. pp. 461–470.

Skopik, F., Fiedler, R., Lendl, O., 2014. Cyber attack information sharing. Datenschutz Datensicherheit 38 (4), 251–256.

Sterlini, P., Massacci, F., Kadenko, N., Fiebig, T., van Eeten, M., 2020. Governance challenges for European cybersecurity policies: Stakeholder views. IEEE Secur. Priv. 18 (1), 46–54.

Suciu, O., Nelson, C., Lyu, Z., Bao, T., Dumitras, T., 2021. Expected exploitability: Predicting the development of functional vulnerability exploits. arXiv:2102.07869.

Syed, R., 2020. Cybersecurity vulnerability management: A conceptual ontology and cyber intelligence alert system. Inf. Manage. 57 (6), 103334. http://dx.doi.org/10.1016/j.im.2020.103334, URL https://www.sciencedirect.com/science/article/pii/S0378720620302718.

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T., 2014. FeatureIDE: An extensible framework for feature-oriented software development. Sci. Comput. Program. 79, 70–85.

Tommy, R., Sundeep, G., Jose, H., 2017. Automatic detection and correction of vulnerabilities using machine learning. In: 2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication. CTCEEC, pp. 1062–1065.

Varela-Vaca, Á.J., Gasca, R.M., 2013. Towards the automatic and optimal selection of risk treatments for business processes using a constraint programming approach. Inf. Softw. Technol. 55 (11), 1948–1973.

Varela-Vaca, Á.J., Gasca, R.M., Carmona-Fombella, J.A., López, M.T.G., 2020. AMADEUS: towards the AutoMAteD security testing. In: SPLC '20. ACM, pp. 11:1–11:12.

Varela-Vaca, A.J., Gasca, R.M., Ceballos, R., Gómez-López, M.T., Torres, P.B., 2019. CyberSPL: A framework for the verification of cybersecurity policy compliance of system configurations using software product lines. Appl. Sci. 9 (24).

Wang, J.A., Guo, M., 2009a. OVM: An ontology for vulnerability management. In: Sheldon, F.T., Peterson, G., Krings, A.W., Abercrombie, R.K., Mili, A. (Eds.), CSIIRW. ACM, p. 34, URL http://dblp.uni-trier.de/db/conf/csiirw/csiirw2009.html#WangG09.

Wang, J.A., Guo, M., 2009b. Security data mining in an ontology for vulnerability management. In: 2009 International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing. IEEE, pp. 597–603.

Weerawardhana, S.S., Mukherjee, S., Ray, I., Howe, A.E., 2014. Automated extraction of vulnerability information for home computer security. In: FPS.

Weston, N., Chitchyan, R., Rashid, A., 2009. A framework for constructing semantically composable feature models from natural language requirements. In: Proceedings of the 13th International Software Product Line Conference. pp. 211–220.

Xiong, W., Lagerström, R., 2019. Threat modeling – A systematic literature review. Comput. Secur. 84, 53–69. http://dx.doi.org/10.1016/j.cose.2019.03.010, URL https://www.sciencedirect.com/science/article/pii/S0167404818307478.

Yadav, T., Rao, A.M., 2015. Technical aspects of cyber kill chain. In: Abawajy, J.H., Mukherjea, S., Thampi, S.M., Ruiz-Martínez, A. (Eds.), Security in Computing and Communications. Springer International Publishing, Cham, pp. 438–452.

Zhang, S., Ou, X., Caragea, D., 2015. Predicting cyber risks through national vulnerability database. Inf. Secur. J. 24 (4–6), 194–206.