Software Vulnerability Management System

Marija Kovačević*, Goran Sladić*, Nikola Luburić*, Miroslav Zarić*
* Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
{marija.kovacevic, sladicg, nikola.luburic, miroslavzaric}@uns.ac.rs

Abstract— **This paper presents an implementation of the Software Vulnerability Management System. The main goal of this system is to analyze data flow diagrams in order to discover potential security exploits.**

The security experts put great effort to discover vulnerabilities in different systems and to successfully prevent all possible attacks. They spend many working hours analyzing diagrams and targeting possible vulnerabilities because targeting them on time reduces the overall cost and time spent on the later phases of the software design.

This system uses a knowledge base to find exploits on the different data flow diagrams and searches the National Vulnerability Database (NVD) to discover publicly known vulnerabilities that are associated with the diagram elements. This system brings a new type of element, that could be used on the diagram with the processes, external entities, and data stores. That element is called a complex process and it contains a new data flow diagram.

This proposed system automates the analyzing process, simplifies threat modeling, and provides a readable report with the list of all found potential security exploits. It was designed as a website with a modest user interface that is easy to use, even by non-experts.

I. INTRODUCTION

An attacker can compromise the entire system by attacking only one element of the system or one data flow. Software companies are constantly working hard to prevent these attacks. One way to solve this problem is to add the security aspect to the product design phase and to designate it as one of the most important aspects. The benefits of pointing out the importance of security and adding its aspect to the product design phase are multiple. It is simpler and more financially sustainable to eliminate vulnerabilities in the earlier phases of product design [1].

Over the years, many techniques have been developed to address various security issues. One of these techniques is threat modeling.

Threat modeling is a process that involves identifying potential threats and requires understanding the complexity of the entire system [1]. Usually, the team of designers share ideas and identify the vulnerabilities of the system. The modeling process should not be limited to complex systems only because even simple systems can contain vulnerabilities and benefit from threat modeling [1].

Threat modeling should be done systematically and on time, because only in this way it can be guaranteed that potential threats and vulnerabilities will be detected by the developer and not the attacker. If modeling is done after the product is already in use, removing attacks and eliminating vulnerabilities will be much more demanding and expensive, than performing it at the product design phase. Moreover, sometimes attempts to fix vulnerabilities increase the risk [2] which can cause much greater problems if the product is already in use.

Although threat modeling can be useful, it is also subject to human errors. Modeling performance depends on the designer's expertise, concentration, and other external factors. In that case, it would be very useful to have a system to help prevent all potential attacks and avoid unplanned costs in the later stages of development or in production itself.

To tackle this problem, we developed a new system, Software Vulnerability Management System. This system analyzes data flow diagrams in order to discover potential threats. The data flow diagram (DFD) represents a graphical view of the model of the system and includes the processes of the model's specification and construction [3].

The overall goal of the Software Vulnerability Management System is to detect all vulnerabilities on the given DFD, which will lead to improved security of the software products.

The rest of the paper is structured as follows. In Section II we present two tools that were the motivation to create this system. In Section III we define our objectives and present how we achieved them. The objectives we defined led to enhancements of the tools we discuss in section II. Section IV presents our system, describing each step of its analyzing process. Finally, in Section V, we conclude our work.

II. RELATED WORKS

In this section, we present two tools that were the motivation to create a Software Vulnerability Management System. For this study, we investigated the Threat Modeling Tool (TMT) and the Exploits Detection System (EDS).

TMT is a tool that enables software architects to detect and remove potential software security issues early. The tool is easy to use because it is designed in a way that non-security experts can create and analyze data flow diagrams [4].

Exploits Detection System is another tool that was used as a motivation for this study. It is a desktop application that is created to correct the essential shortcomings that exist in the TMT, such as the absence of the assets in the diagrams and limitations of the diagram analyzing process [5].

EDS successfully overcomes the drawbacks of the TMT, but on the other hand, has its own. One of the problems is the tight coupling with the assets. Only assets that are defined in the XML file can be used in this application. The user can modify default assets, but if the asset definitions file is not in accordance with its XML Schema, the system won't function properly. Furthermore, this application does not support diagrams with complex elements and the final report is provided to the user in the XML format, which is not very readable.

III. Methods

In this section, we define the goals we set and describe how we achieved them. We emphasize our main focus.

The system we propose is implemented as a website with a modest user interface. Anyone can easily use it and analyze data flow diagrams. Usually, system architects, security experts, or managers create input files and review the final report. It may make sense to perform threat modeling only with engineers, without experts, because hiring experts can be substantially expensive [6]. But, for the best results, it is recommended that the system architect creates a data flow diagram, the security expert should create rules and the system architect or manager should receive a final report.

We set multiple goals to create a system that is accurate, easy to use, and sufficiently secure. Therefore, we define the following goals:

1. Creation of the flexible knowledge base;
2. Introduction of the component for detecting publicly known vulnerabilities
3. Defining a new type of element, a complex process.

A. *Creation of the flexible knowledge base*

We decided to use the knowledge base for detecting the vulnerabilities. The main reason for this decision is because the knowledge base is logically separated from the application code. It has a set of rules that are easy to modify, remove, or to add a new one, which makes it flexible and easy to maintain.

The knowledge base of the system was implemented with the help of Drools technologies. Rules are written using the DRL (Drools Rule Language) language [7]. Each rule contains a name (rule), a condition (when), and consequences (then). When the condition is met, then the consequences are executed.

The database consists of a single document containing all the rules. The rules are divided into subgroups depending on the type of element. There are 3 subgroups, each for the corresponding type of element:

- Rules for processes;
- Rules for data stores;
- Rules for external elements.

Therefore, rules are not executed all at once, but the corresponding subgroup of rules is activated, depending on the type of element that is analyzed. If an element is a process, a subgroup with rules for the processes will be activated. If the element is a data store, a subgroup with rules for the data stores will be activated, and so on.

Considering only the types of elements is not enough to get a sufficiently accurate result. For that reason, the rules do not only consider the types of elements but also review the element's attributes and the defined assets within them. The creator of DFD may associate additional attributes and assets to the elements on the diagram. Each element may contain a list of assets. An asset has an ID, a category to which it belongs, and a list of security objectives. Security objectives are:

- Confidentiality;
- Integrity;
- Availability.

All security objectives are passed from the assets to the elements. When the element contains an asset whose e.g. confidentiality must be protected, then that security objective is added to the element. For example, if the element is a database that has a password as an asset, which has the security objective of confidentiality, then the confidentiality of the database should also be protected.

It is important to highlight that the knowledge base is not tightly coupled to the assets that are defined on the diagram elements, but it's rather software-centric.

One of the rules from the database is the rule for detecting sniffing attacks[1]. The rule goes through all assets, which belong to an element, and checks if either of the security objectives is confidentiality. If the confidentiality of the asset is protected, then the confidentiality of the element should be protected as well. Another condition that must be met is that the link between the observed element and some other element uses HTTP protocol. In that case, the confidentiality of the asset can be easily compromised by carrying out a sniffing attack.

B. *Introduction of the component for detecting publicly known vulnerabilities*

Within the last three years, more vulnerabilities have been detected than by 2010 [8]. This tells us that a significant number of vulnerabilities are discovered every year. The higher number of reported vulnerabilities is a consequence of the increased focus on research in this field [9]. It is important to note that this does not happen for no reason, because in more than 80% of security attacks known vulnerabilities were exploited [10]. Therefore, generating the list of publicly known vulnerabilities was another important task.

The National Vulnerability Database (NVD) is searched to discover publicly known vulnerabilities. The NVD is a United States Government database of standards-based vulnerability management reference data [11]. It updates its database with a new vulnerability whenever it is added to the Common Vulnerabilities and Exposures (CVE) dictionary [12]. Before updating NVD, security experts annotate the newly disclosed vulnerability with the metadata such as CPE, CVSS, etc. [13].

To be able to search any vulnerabilities database we need to identify components on the input DFD. We can reliably descriptively identify any component in a few sentences. However, such identified components would not be machine-readable, or it would require a very complex system implementation. We decided to use a standardized way to identify components, Common Platform Enumeration (CPE), which NVD also uses.

[1] Sniffing attack occurs in situations where not encrypted data packets are transmitted across networks. These packets can easily be read without authorization

Common Platform Enumeration (CPE) is a standardized way to describe and identify operating systems, applications, and devices [14]. The CPE assigns names to the products by specifying a set of attributes called Well-Formed CPE Name (WFN) [15]. WFN is an abstract logical construction that can be transformed into machine-readable encodings [14]. There are two formats in which it can be transformed:

1. Uniform Resource Identifier (URI);
2. Formatted string.

These two formats are generated from the WFN in a process called URI or formatted string binding [15].

We have expanded elements of our system so that we can search NVD. DFD elements, in addition to a list of assets, also contain lists of CPEs. The creator of DFD can assign the most suitable CPEs to the element on the diagram. Each CPE item in the list has 3 fields that define it: a path, title, and description. The most important attribute of a CPE item is the path. This attribute is an identifier of a CPE item and can be specified in URI or formatted string format. This attribute is used to search the NVD.

The component for detecting known vulnerabilities, for all defined CPEs, searches NVD's vulnerability list to find a match. If a suitable match is found, all its properties are added into the final report.

When comparing our system with those mentioned in Section II, it must be pointed out that with this component we have significantly improved security and weakened the attackers. The attacker must create a new way of attacking and make more efforts to initiate an attack because all known vulnerabilities are detected [9].

C. Defining a new type of element, a complex process

The last milestone was to create support for the new type of element that could be found on the diagram. That element has its data flow diagram, so we call it a complex process element. To simplify, we will call it a complex process from now on.

In EDS the data flow diagram has support for only 3 types of elements: process, external entity, and data store. With the introduction of a new element, the DFD and analyzing process become more complex, but the user of the system is no longer limited when creating DFD.

Complex processes are observed in the first phase of the process of diagram analysis. They are recursively decomposed into smaller parts, which are then passed to the knowledge base for analysis. If any threats are found, they are forwarded to the user of the system. The user of the system reviews the potential attacks on the complex processes and decides whether they could affect the entire diagram. If there can be a significant impact, the final report will contain those attacks.

IV. Proposed Solution

The following section describes our proposed system. We describe each step of the analysis process, from the decomposition of the input data flow diagram to the creation of the final report.

The analysis process is divided into 2 main phases:

1. Decomposing of the diagram and generating vulnerabilities for the complex process.
2. Generating vulnerabilities for the whole diagram and creating the final report.

A. Decomposing of the diagram and generating vulnerabilities for the complex process

The main goal of the first phase is to decompose an entire input diagram and to review only complex processes if they exist. The result of the analysis is a generated list of vulnerabilities, that could be found on the complex processes. The list of vulnerabilities for the rest of the diagram is generated in the next phase.

This phase has 4 steps:

1. Validating input documents;
2. Decomposing data flow diagram;
3. Analyzing risk patterns found on the complex processes;
4. Merging risk patterns with definitions of attacks.

a) Validating input documents

In the first step, at the beginning of the analyzing process, the system validates input documents that are specified by the user.

The user should provide the data flow diagram for analyzing, together with the diagram's XML schema. Also, the user can specify a custom XML file with attack definitions. When specifying this file, users should provide attacks description and countermeasures.

When all required documents are specified, the system checks if there are any syntax or semantic errors. A semantic check is performed against the provided XML schema. If there are no errors, the system will go to the next step.

b) Decomposing data flow diagram

The DFD is usually very complex and analyzing such a system will be substantially more demanding. As stated in [6] a typical DFD has between 10 and 150 elements. That is why we decided to decompose the diagram into smaller parts. Smaller parts of the diagram we named risk patterns.

The risk pattern consists of two elements and data flow between them or may consist of only one element. Also, the pattern has a list of assets on both connected elements, the start element, and the end element.

The decomposition algorithm is the same as the EDS proposed. It goes through every element that is in the complex process and finds all other elements that are directly or indirectly leading to this one.

When the system starts decomposing DFD, it first decomposes it into elements, data flows, and complex processes. This step is crucial for the further progress of this analyzing process phase. If there are no complex processes on the DFD, the system will move to the second phase, otherwise, it will continue this phase with further decomposing of the complex processes.

If there is at least one complex process on the diagram, it will be decomposed into risk patterns. All found risk patterns are analyzed in the next step.

c) Analyzing risk patterns found on the complex processes

The third step of this phase is analyzing risk patterns found on the complex processes. Each risk pattern is forwarded to a rule-based engine, which consists of a set of rules, to analyze and identify potential threats. The analysis of the pattern results with a list of attacks that can be performed on it.

To obtain a list of potential attacks, the knowledge base rules must be applied to each pattern. Once a potential attack is detected, the rule-based engine adds its identification number in the risk pattern.

d) Merging risk patterns with definitions of attacks

The last step of this phase is merging definitions of the attacks with the risk patterns. In the previous step, the identifiers of the potential attacks are saved within the risk pattern. Identifiers of the attacks are not useful for the end-user of the system. It is more useful to receive the attack's name, description and recommended countermeasures that can be taken. Therefore, the system goes through all attack's identifiers, saved in risk patterns, and replaces them with the corresponding attack properties (description and countermeasures).

Since the last step was successfully finished, the list of all found attacks, within the complex processes, is forwarded to the user of the system. The user should review all attacks and decide which attacks are relevant to the final report. After selecting the relevant attacks, the first phase of the analysis is completed.

B. Generating vulnerabilities for the whole diagram and creating the final report

In the first phase, we focused only on the specific elements, complex processes, within the diagram, and took no account of the data flow of the whole diagram. Now we analyze the whole diagram and search for the publicly known vulnerabilities.

At the end of this phase, we create a final report.

This phase has 4 steps:

1. Analyzing risk patterns;
2. Merging risk patterns with attacks definitions;
3. Connecting with the National Vulnerability Database (NVD) and generating publicly knows vulnerabilities;
4. Creating a final report.

a) Analyzing risk patterns

The first two steps do not differ much from the so-called steps in the previous phase. The only difference is that the system in this phase analyzes the patterns found on the rest of the diagram and not on the complex processes.

The system decomposes the rest of the diagram into risk patterns. Afterward, the analysis of the risk pattern starts. The same knowledge base is used to apply rules to each pattern as in the previous phase.

Once a potential attack is detected, the identifier of that attack is saved within the pattern. This step is finished when the set of rules are applied to all found patterns.

b) Merging risk patterns with attacks definitions

Next, definitions of the attacks are merged with the patterns. The merger will result in patterns that contain attack properties, including descriptions and recommended countermeasures.

c) Connecting with NVD and generating publicly knows vulnerabilities

In the third step, the component for detecting publicly known vulnerabilities generates a list of vulnerabilities.

The system can contact NVD at any time and request a list of all vulnerabilities with the latest, most recent changes. NVD data could be easily downloaded and extracted when needed.

For each defined CPE, the component searches NVD to find a suitable match. If there are any publicly known vulnerabilities corresponding to a CPE, the component adds their properties to the final report. Some of the properties are the access vector, the complexity of attacks, confidentiality, integrity, availability, etc.

d) Creating a final report

Finally, when the analyzing process is complete, all vulnerabilities and attacks are added into the final report. The final report is in the form of a PDF document and contains the following data:

- Date and time;
- Name of the file (DFD) that is being analyzed;
- List of patterns with all discovered possible attacks;
- List of all possible attacks that are discovered on the complex process elements;
- List of publicly known vulnerabilities for the CPEs that are specified on the diagram elements.

The final report is concise and provides plenty of information.

A drawback of the proposed system is that it does not have a graphical user interface for creating DFD. The creator must specify the DFD in the text file in the proper format before starting the analysis process, which can be time-consuming.

V. Conclusion

In this paper, we present a Software Vulnerability Management System. This system analyzes data flow diagrams in order to discover potential vulnerabilities.

The main objectives and advantages of our system are the implementation of the flexible knowledge base, a new component for detecting publicly known vulnerabilities, and the creation of the new type of element, a complex process.

The created knowledge base has a set of rules which consider types of elements, their attributes, and the defined assets within them. Next, to discover known vulnerabilities we search NVD with the CPE items specified on DFD elements. Finally, the creator of the DFD can add a complex process element, which is a subdiagram of the DFD.

Software Vulnerability Management System automates the analyzing process and simplifies threat modeling. It creates an easily readable report and offers countermeasures that can be taken. This proposed system provides support for building cost-effective and safe products.

Further work includes the integration of a graphical user interface component that would drastically improve the use of the system. Furthermore, analyzing the performance of NVD should be done. In [16] authors evaluated vulnerability disclosure delays from NVD to check how efficient it is. They compared NVD with other sources and their results prove that NVD is ineffective in detecting vulnerabilities, because it depends on CVE. The goal of this system improvement would be to review relevant papers and to try to integrate other vulnerabilities databases.

References

[1] Suvda Myagmar, Adam J. Lee, William Yurcik, “Threat Modeling as a Basic for Security Requirements”, National Center for Supercomputing Applications (NCSA), University of Illinois at Urbana-Champaign, 2005.

[2] Matt Bishop, “Vulnerabilities Analysis”, Department of Computer Science University of California, 1999.

[3] Rosziati Ibrahim, Siow Yen Yen, “Formalization of the data flow diagram rules for consistency check”, Department of Software Engineering, Faculty of Computer Science and Information, Technology, Universiti Tun Hussein Onn Malaysia (UTHM), *International Journal of Software Engineering & Applications* (IJSEA), October 2010.

[4] Microsoft, Secure Development Documentation, Microsoft Threat Modeling Tool,

https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool, Accessed: 26/5/2020.

[5] Exploits Detection System, https://github.com/cyber-security-novi-sad/Pretnja/wiki, Accessed: 26/5/2020.

[6] Adam Shostack, “Experiences Threat Modeling in Microsoft”, 2008.

[7] Drools, https://www.drools.org/, Accessed: 26/5/2020.

[8] Listed vulnerabilities by date, https://www.cvedetails.com/browse-by-date.php, Accessed: 26/5/2020.

[9] Kelley Dempsey, Paul Eavy, George Moore, Eduardo Takamura, “Automation Support for Security Control Assessments: Software Vulnerability Management” NISTIR National Institute of Standards and Technology, 2019.

[10] P. John, “Cyber Security Trends: Aiming Ahead of the Target to Increase Security in 2017”, SANS Institute, March 2017.

[11] The National Vulnerability Database, https://nvd.nist.gov/general, Accessed: 26/5/2020.

[12] Common Vulnerabilities and Exposures, https://cve.mitre.org/, Accessed: 26/5/2020.

[13] Clément Elbaz, Louis Rilling, Christine Morin, “Towards Automated Risk Analysis of "One-day" Vulnerabilities“, RESSI 2019 - Rendez-vous de la Recherche et de l’Enseignement de la Sécurité des Systèmes d’Information, 2019.

[14] Brant A. Cheikes, David Waltermire, Karen Scarfone, “Common Platform Enumeration: Naming Specification Version 2.3” Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2011.

[15] Luis Alberto, Benthin Sanguino, Rafael Uetz, “Software Vulnerability Analysis Using CPE and CVE”, 2017.

[16] Luis Gustavo, Araujo Rodriguez, Julia Selvatici Trazzi, Victor Fossaluza, Rodrigo Campiolo, Daniel Macedo Batista, “Analysis of Vulnerability Disclosure Delays from the National Vulnerability Database”, 2018.