Mining Security Requirements from Common Vulnerabilities and Exposures for Agile Projects

Wentao Wang, Arushi Gupta, and Nan Niu
Department of Electrical Engineering and Computer Science, University of Cincinnati, USA
Email: {wang2wt, gupta2ai}@mail.uc.edu, nan.niu@uc.edu

***Abstract*—Agile software development (ASD) is becoming increasingly popular in the software industry. Several researchers point out that characterized with short iterations and the quick delivery of working software, ASD often does not give consideration to security requirements as well as other non-functional requirements. This means important security requirements might be neglected in ASD. However, implementing all necessary security requirements is determinant for the success of software projects. Many approaches are proposed to elicit security requirements, but most of them rely on analysts' knowledge and experience about security requirements management. In this paper, we propose a new approach in which security requirements are mined from the vulnerability repository of common vulnerabilities and exposures (CVE). We describe our approach with illustrative examples, discuss operational insights, and raise research questions for future work.**

***Index Terms*—Security requirements, common vulnerabilities and exposures, data mining, security testing**

I. Introduction

Security refers to a class of non-functional requirements (NFRs) related to software system's confidentiality, integrity, and availability [1]. Security experts point out that failing to fulfill security requirements leads to vulnerabilities in software systems. In September 2017, Equifax, a customer credit reporting agency, announced a security breach where cybercriminals accessed approximately 145.5 million United States Equifax consumers' personal data, including their full names, social security numbers, and credit card information [2]. Equifax reported that this data breach was caused by a vulnerability in the Apache Struts 2[1], a web application framework used by Equifax to build its website.

Compared to commercial software, security vulnerabilities in open-source software (OSS) projects have a greater impact on other software systems, since they are available to public and many other software projects are dependent on them. For instance, in 2016, common vulnerabilities and exposures (CVE)[2] published a vulnerability[3] in Libxml2[4], an open-source XML file parser. More than 10 companies, including Apple and IBM, reported that over 30 of their products were affected by this vulnerability since developers use Libxml2 in those products.

The situation becomes worse and worse these days. CVE published over 14 thousand vulnerabilities in 2017, 1.27 times more than the total number of 2016. Over 20% of them are reported in OSS projects. Part of reason comes from agile software development (ASD), a software development approach which is widely used in OSS projects. According to Behutiye *et al.* [3], in ASD, developers either do not document security requirements and other NFRs, or ignore lower-level details such as security designs in requirements documentation. Documenting security requirements improperly introduces difficulties for following the life of security requirements. Therefore, having a suitable approach to help development teams elicit important security requirements and document them properly is determinant for the success of ASD projects.

A couple of security requirements elicitation approaches have been proposed [4], [5]. However, most of them rely on analysts' expertise and experience. Unfortunately, security experts are not always available in ASD teams. Meanwhile, CVE becomes the industry standard for security vulnerability and exposure identifiers. Security vulnerabilities and related implementation-level details, such as attacks to vulnerabilities and mitigations for security vulnerabilities, are well-documented in CVE. This information can be used to not only guide security requirements management in ASD projects, but also help design test cases for verifying and validating security requirements.

In this research, we propose a new approach to mining related security requirements from CVE for the given functional requirements (FRs) in ASD projects. For example, a related security requirement for the sub-requirement "uploading patient's information via CVS file" in use case 1 in iTrust[5] is "If the file is malformed, then no data is added, and an error message explaining the correct file structure is presented". In our approach, an automated tool is used to retrieve FRs' related vulnerabilities from CVE. In order to further reduce manual effort, a mechanism is developed to filter out redundant vulnerabilities. Then security requirements are derived from retrieved vulnerabilities. In the remainder of the paper we describe our process for deriving security requirements and test cases from CVE within an ASD project environment. Section II describes the subject project, iTrust, which provides the context for our work and from which all of our examples are drawn. Section III provides a detailed description of the process with illustrative examples. We discuss related work

[1]https://nvd.nist.gov/vuln/detail/CVE-2017-5638
[2]https://cve.mitre.org
[3]https://nvd.nist.gov/vuln/detail/CVE-2016-4449
[4]http://www.xmlsoft.org
[5]http://agile.csc.ncsu.edu/iTrust

978-1-5386-8412-2/18/$31.00 ©2018 IEEE
DOI 10.1109/QuaRAP.2018.00007

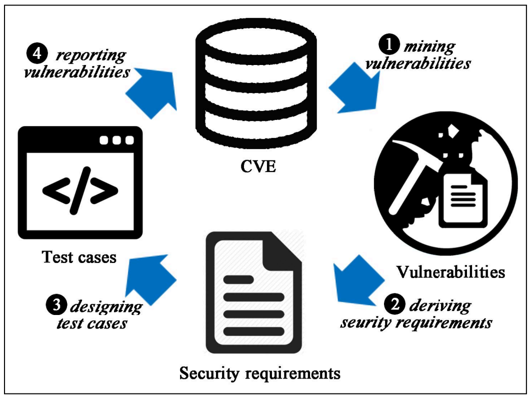
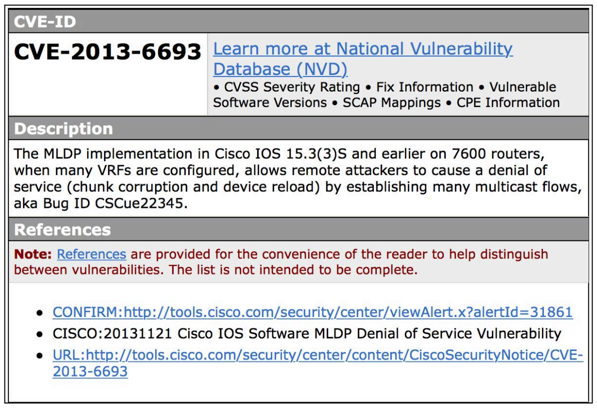


Fig. 1. Overview of mining security requirements from common vulnerabilities and exposures (CVE)



CVE-ID

CVE-2013-6693 [Learn more at National Vulnerability Database (NVD)](#)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

The MLDP implementation in Cisco IOS 15.3(3)S and earlier on 7600 routers, when many VRFs are configured, allows remote attackers to cause a denial of service (chunk corruption and device reload) by establishing many multicast flows, aka Bug ID CSCue22345.

References

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

- CONFIRM:http://tools.cisco.com/security/center/viewAlert.x?alertId=31861
- CISCO:20131121 Cisco IOS Software MLDP Denial of Service Vulnerability
- URL:http://tools.cisco.com/security/center/content/CiscoSecurityNotice/CVE-2013-6693

Fig. 2. CVE ID example: CVE-2013-6693

and draw conclusions in Section IV.

II. The Subject Project

The process described in this paper emerged from our subjective lessons learned in detecting vulnerabilities in the iTrust project. iTrust is a medical web application that provides patients with a means to keep up with their medical histories. All healthcare related products in the USA must comply with the Health Insurance Portability and Accountability Act (HIPAA) [6], which is the legislation that provides data privacy and security provisions for safeguarding medical information. Therefore, security and dependability is one of the main concerns of the iTrust project [7].

iTrust was created by Dr. Laurie Williams in the Fall of 2005. Starting from version 6, the project follows a predefined release cycle (i.e., six months). In each cycle, developers keep changing the requirements [8]. Since iterative release cycle and adaptability to late requirements changes are core principles of ASD method [9], we believe that developers of the iTrust project follow the ASD.

The project employs Java Server Pages (JSPs) to handle user interfaces and HTTP requests. The business logic and data accesses are coded in Java production classes. Security requirements are recorded in its project wiki. However, only system-wide, high-level security requirements are documented, such as “implementation must not violate HIPAA guidelines”. According to Behutiye *et al.* [3], missing lower-level details of security requirements introduces challenges to ASD developers, especially when requirements are constantly changing [10]. Therefore, a new approach is needed to improve security requirements management in ASD projects.

III. Mining Security Requirements from CVE

The overview of our approach is summarized in Fig. 1. In this section, we describe each of these steps with illustrative examples from iTrust.

A. Mining Vulnerabilities

CVE is an open-source repository of cybersecurity vulnerabilities. For each vulnerability, CVE documents it as a CVE entry or CVE ID. Fig. 2 shows an example of CVE ID (CVE-2013-6693). In this example, detailed information about the vulnerability, including the name of the affected product, the vulnerability type, the access that an attacker requires to exploit the vulnerability, and the important code components that are involved, is documented in the “Description” portion. References attached to the vulnerability are used for identifying source of the vulnerability, recording notes of the vulnerability, or describing attacks associated to the vulnerability.

Although vulnerabilities published by CVE are mitigated, the recorded high-quality information can be used to guide building projects more securely. However, manually analyzing all related vulnerabilities from similar projects for ASD projects is an impossible mission since there are over 100,000 vulnerabilities in CVE database. An automated tool which can retrieve highly related vulnerabilities for FRs in ASD projects can help improve developers’ working efficiencies.

Most information retrieval methods used in requirements engineering like vector space model [11] are based on shared terms between documents. However, according to our experiments, performances of those methods in mining related vulnerabilities for FRs are poor. The main reason is that CVE and ASD developers use different words to describe same objects. This is referred to as the term mismatch problem [12]. For example, iTrust developers use “database” in FRs (e.g., UC12[6]), and CVE uses database management system names like “MySQL” in description of vulnerabilities. However, both of them are associated to the same topic “database management system”. This type of latent semantic relationship between words can be used to improve the performances of information retrieval methods.

Latent semantic indexing (LSI) is a prominent method for capturing latent semantic relations [13]. We applied LSI to improve our automated vulnerability mining tool. Step ❶ in Fig. 1 represents this process. The output of information retrieval methods is a ranked list of vulnerabilities according

[6]https://152.46.18.254/doku.php?id=requirements:uc12

to their similarities to FRs. Currently, only top 70% highest-ranked vulnerabilities are selected as candidate vulnerabilities. However, there are still too many candidates. Therefore, a future research topic could be how to further reduce the number of candidates. We summarize this topic as research question *RQ1*:

- *RQ1*: How to improve information retrieval method to achieve the goal of removing irrelevant candidates without filtering our relevant ones?

B. Derive Security Requirements

For a FR, after retrieving candidate vulnerabilities, we first group them with their common weakness enumerations (CWE)[7] types which are predefined by CVE. For each FR in iTrust, average 100 types of vulnerabilities are retrieved from step ❶. We only consider top 10 types that contain the most vulnerabilities. There are two main reasons: 1) statistic analysis indicates that the distribution of vulnerabilities in different types fits to power law distribution [14] (i.e., 90% vulnerabilities come from 10% vulnerability types), and 2) most of remaining 90% vulnerability types are irrelevant to FRs. For instance, there are three vulnerabilities (i.e., CVE-2017-13871, CVE-2017-13903, and CVE-2017-13828) belonging to CWE-371 (state issues) and retrieved as candidates for UC1 (Create and disable patients)[8] in step ❶. According to CWE, state issues vulnerabilities are related to improper management of operating system state. Obviously, it is irrelevant to UC1 which describes the process of creating new patients and disabling existing patients.

For each CWE type, we then manually investigate vulnerabilities belonging to this type. We found that if vulnerabilities satisfy one of following conditions, that means they are irrelevant and shall be ignored. First, if a vulnerability is reported in the implementation of a component/method in a specific programming language platform and we shall ignore this vulnerability unless we use this component/method in our ASD project. For instance, CVE-2012-5373[9] which was caused by a malfunctioned method in Oracle Java Development Kit is retrieved as candidate vulnerability of UC1. However, according to the original developers' documentation, iTrust does not use this method, therefore we ignore this vulnerability. Second, if a vulnerability is reported in the operating environment we shall ignore it. For instance, another retrieved candidate for UC1 (CVE-2012-1941[10]) is a buffer overflow vulnerability reported in Mozilla Firefox which happened while Firefox tries to calculate the size of HTML components like hypothetical box. However, we can do nothing except for asking iTrust users to run it on other web browsers instead of Firefox. In fact, those conditions can be used to help solve research question *RQ1*.

[7]CWE is a list of software weaknesses types http://cwe.mitre.org/data/index.html.

[8]https://152.46.18.254/doku.php?id=requirements:uc1

[9]https://nvd.nist.gov/vuln/detail/CVE-2012-5373

[10]https://nvd.nist.gov/vuln/detail/CVE-2012-1941

For the remaining candidates, we derive security requirements from their description and document security requirements as acceptance criteria in FRs. We considered other representations as well, such as ASD-specific artifacts like backlog. However, the complexity of backlogs' structure makes it hard to maintain traceability between security requirements and FRs, especially when requirements changes are anticipated [3]. The "Given-When-Then" format is used to document the acceptance criteria. For example, in UC1's related vulnerability CVE-2017-15974[11], a vulnerability is reported as "tPanel 2009 allows SQL injection for Authentication Bypass via 'or 1=1' to login.php". Therefore, we derived an acceptance criteria for UC1 as following:

> *AC1*: Given an eligible user, when create patient or upload patients, then all input values shall be properly sanitized to prevent tautology (e.g., 1=1).

For the similar vulnerabilities, instead of putting them into separate acceptance criteria, we merge them together. For example, another related vulnerability (CVE-2012-5612[12]) for UC1 is described as "Heap-based buffer overflow in Oracle MySQL, allows remote authenticated users to cause a denial of service, as demonstrated using certain variations of the (1) use, (2) show tables, (3) describe, (4) show fields from, (5) show columns from, (6) show index from, (7) create table, (8) drop table, (9) alter table, (10) delete from, (11) update, and (12) set password commands". We merge it into *AC1*:

> *AC1*: Given an eligible user, when create patient or upload patients, then all input values shall be properly sanitized to prevent SQL injection, such as tautology (e.g., 1=1) or SQL commands (e.g., use, show tables, describe, show fields, show columns, show index, create table, drop table, alter table, delete from, update, and set password).

We represent the process described in this sub-section as step ❷ in Fig. 1.

Another interesting phenomenon we observed in step ❷ is that, it is very hard to derive new acceptance criteria after top 20 vulnerabilities for each CWE type. Since much vulnerability after top 20 is irrelevant or repeats previous ones. This phenomenon makes us speculate that classifying similar vulnerabilities according to their topics instead of types, and only investigating representative vulnerabilities in each topic can potentially reduce manual workload. We summarize it as a research question *RQ2*:

- *RQ2*: Which methods can better classify vulnerabilities to achieve the goal of easily selecting all representative vulnerabilities?

C. Design and Execute Test Cases

Like in acceptance criteria *AC1*, test cases can be easily defined since attacks associated with the security requirements are provided. For example, a trace link from UC1 to a Java method "AddPatientAction().addPatient(PatientBean p)" is defined by original developers, where new patient's first

[11]https://nvd.nist.gov/vuln/detail/CVE-2017-15974

[12]https://nvd.nist.gov/vuln/detail/CVE-2012-5612

name is saved in PatientBean.firstName. Therefore a test input derived from *AC1* could be: PatientBean.firstName="1=1", and the expected output of this test case is that "AddPatientAction().addPatient()" will return an error message "the input of patient's first name is illegal". In addition, since some references of vulnerabilities provide associated attacks, those attacks are also valuable for designing security test cases.

For different test cases, we apply different technologies to execute them. For example, test cases extracted from SQL injection vulnerabilities like *AC1* can be executed by applying JUnit. For test cases extracted from other vulnerability types such as cross-site scripting (XSS), vulnerability scanning tools like snuck [15] can be used to execute them.

Step ❸ in Fig. 1 represents test case generation process. An important research question for testing is the quality of those test cases. Testing coverage is one of important metrics that can be used to evaluate quality of test cases. More test cases increase testing coverage. Meanwhile, malicious attackers will learn new knowledage in order to develop new attacks [16]. White-box fuzzing, which generates new test cases by modifying test inputs, could be a useful approach to predict new attacks as well as increase testing coverage. Therefore, the third future research question *RQ3* can be summarized as:

- *RQ3*: How to modify attacks in CVE to generate more security test cases which can achieve the goal of increasing testing coverage?

New vulnerabilities can be observed from analyzing testing results. Those new vulnerabilities shall be included in CVE database in order to guide security requirements management in other projects. We model this process as step ❹ in Fig. 1.

IV. Conclusions

In this paper, we describe a new security requirements mining approach that provides following benefits: First, all security requirements come from well-documented vulnerabilities in CVE, therefore the qualities of them are guaranteed. Second, it is learnability effective. Since less security related experience is needed when using our approach, new developers to security can also apply it to manage security requirements in ASD. Third, adding security acceptance criteria into the user story will increase awareness regarding security requirements.

There are several existing security requirements management approaches. Cleland-Huang proposed a lightweight approach to integrate safety stories into ASD [4]. Elicitation of security requirements is based on brainstorming, checklists, and analyzing reports of previous failures. Since the success of analysis relies on analysts' experience, it introduces challenges to new developers joining the team who have limited visibility of security requirements. Our approach, benefited from well-documented vulnerabilities, can help solve this problem.

Alqahtani *et al.* [5] proposed another security vulnerability analysis framework called SV-AF. In this framework, bi-directional traceability links between project source code and vulnerabilities reported in projects which are used in this project are estimated. With those trace links, this approach supports finding affected projects of vulnerabilities reported in CVE. Having a way to combine our approach and SV-AF can make the results of step ❶ (retrieving vulnerability) more accurate.

This research is ongoing. There are still several research questions described in Section III that need to be addressed. Other than that, another limitation of current research is that some steps (e.g., test cases generation in step ❸) are done manually. Therefore further automation can be pursued. Finally, differences between vulnerability categories lead to different expressions of attacks in CVE (e.g, textual description vs. source code), questions regarding the generalizability of our approach require systematic analysis.

Acknowledgement

The work is funded by the U.S. National Science Foundation (Award CCF 1350487).

References

[1] A. Avizienis, J.-C. Laprie, B. Raandell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," in *IEEE Trans. on Dependable Secure Computing*, vol. 1, no. 1, 2004, pp. 11–33.

[2] Equifax, "2017 Cybersecurity Incident & Important Consumer Information," 2017. [Online]. Available: https://www.equifaxsecurity2017.com

[3] W. Behutiye, P. Karhapää, D. Costal, M. Oivo, and X. Franch, "Non-functional requirements documentation in agile software development: challenges and solution proposal," in *Proc. the 18th PROFES*, Innsbruck, Austria, Nov. 2017, pp. 515–522.

[4] J. Cleland-Huang, "Safety stories in agile development," in *IEEE Software*, vol. 34, no. 4, 2017, pp. 16–19.

[5] S. S. Alqahtani, E. E. Eghan, and J. Rilling, "SV-AF - a security vulnerability analysis framework," in *Proc. the 27th IEEE ISSRE*, Ottawa, ON, Canada, Oct. 2017, pp. 219–229.

[6] U.S. Department of Health and Human Services, "Health Insurance Portability and Accountability Act (HIPAA) of 1996," 1996. [Online]. Available: https://www.hhs.gov/hipaa/index.html

[7] W. Wang, A. Gupta, N. Niu, L. D. Xu, J.-R. C. Cheng, and Z. Niu, "Automatically tracing dependability requirements via term-based relevance feedback," in *IEEE Trans. on Industrial Informatics*, vol. 14, no. 1, 2018, pp. 342–349.

[8] W. Wang, A. Gupta, and Y. Wu, "Continuously delivered? periodically updated? never changed? studying an open source project's releases of code, requirements, and trace matrix." in *Proc. the 1st IEEE JIT RE*, Ottawa, ON, Canada, Aug. 2015, pp. 13–16.

[9] L. López, W. Behutiye, P. Karhapää, J. Ralyté, X. Franch, and M. Oivo, "Agile quality requirements management best practices portfolio: A situational method engineering approach," in *Proc. the 18th PROFES*, Innsbruck, Austria, Nov. 2017, pp. 548–555.

[10] N. Niu, W. Wang, and A. Gupta, "Gray links in the use of requirements traceability," in *Proc. the 24th FSE*, Seattle, WA, USA, Nov. 2016, pp. 384–395.

[11] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," in *IEEE Trans. on Software Engineering*, vol. 32, no. 1, 2006, pp. 4–19.

[12] N. Niu and S. M. Easterbrook, "Managing terminological interference in goal models with repertory grid," in *Proc. the 14th RE*, Minneapolis/St.Paul, MN, USA., Sep. 2006, pp. 296–299.

[13] A. Mahmoud and N. Niu, "On the role of semantics in automated requirements tracing," in *Requirements Engineering*, vol. 20, no. 3, 2015, pp. 281–300.

[14] P. Louridas, D. Spinellis, and V. Vlachos, "Power laws in software," in *ACM Trans. on Software Engineering and Methodology*, vol. 18, no. 1, 2008, pp. 2:1–2:26.

[15] F. d'Amore and M. Gentile, "Automatic and context-aware cross-site scripting filter evasion," Sapienza University of Roma, Roma, Italy, Tech. Rep. 1, Oct. 2012.

[16] N. Hussein, W. Wang, J. L. Nedelec, X. Wei, and N. Niu, "Unified profiling of attackers via domain modeling," in *Proc. the 1st iRENIC*, Beijing, China, Sep. 2016, pp. 98–101.