Enabling the Continuous Analysis of Security Vulnerabilities with VulData7

Matthieu Jimenez
Snt, University of Luxembourg
matthieu.jimenez@uni.lu

Yves Le Traon
Snt, University of Luxembourg
yves.letraon@uni.lu

Mike Papadakis
Snt, University of Luxembourg
michail.papadakis@uni.lu

***Abstract*—Studies on security vulnerabilities require the analysis, investigation and comprehension of real vulnerable code instances. However, collecting and experimenting with a sufficient number of such instances is challenging. To cope with this issue, we developed VulData7, an extensible framework and dataset of real vulnerabilities, automatically collected from software archives. The current version of the dataset contains all reported vulnerabilities (in the NVD database) of 4 security critical open source systems, i.e., Linux Kernel, WireShark, OpenSSL, SystemD. For each vulnerability, VulData7 provides the vulnerability report data (description, CVE number, CWE number, CVSS severity score and others), the vulnerable code instance (list of versions), and when available its corresponding patches (list of fixing commits) and the files (before and after fix). VulData7 is automated, flexible and easily extensible. Once configured, it extracts and links information from the related software archives (through Git and NVD reports) to create a dataset that is continuously updated with the latest information available. Currently, VulData7 retrieves fixes for 1,600 out of the 2,800 reported vulnerabilities of the 4 systems. The framework also supports the collection of additional software defects and aims at easing empirical studies and analyses. We believe that our framework is a valuable resource for both developers and researchers interested in secure software development. VulData7 can also serve educational purposes and trigger research on source code analysis. VulData7 is publicly available at : https://github.com/electricalwind/data7**

***Index Terms*—Dataset, Tool, Software Archives, Security Vulnerabilities**

I. Introduction

Secure code development is an important requirement for the development of security-critical software. A simple omission can lead to disastrous issues, which have the potential, such as the case of the OpenSSL heartbleed bug, of jeopardising the safety of millions of computers [1]. To deal with this issue, security-critical software industry usually involves dedicated teams, training processes and procedures for identifying security holes and potential issues [2].

In practice, secure development requires extensive experience, special skills and an attacker's mindset [3]. Vulnerabilities, usually remain unnoticed for a long time, as they do not disrupt the typical system use cases. The difficulty is the weakness identification and unawareness of developers. To deal with these issues, researchers and organizations develop tools, techniques and models aiming at informing and helping developers.

However, ensuring the realism and assessment of empirical studies is challenging due to the shortage of real and established corpora of vulnerable code instances [4], [5]. Such ground-truth instances are used to develop and assess automated tools and techniques. However, the lack of sufficient representative vulnerable code instances prevents the measurable assessment and performance interpretation of the security related research results. Overall, the use of few subjects threatens the external validity of empirical studies [6].

To deal with this issue, researchers need to collect and study a relatively large number of vulnerabilities. Such a corpus is composed by collecting information from previously reported vulnerabilities. However, extracting related information and liking it from the vulnerability reported systems, such as the National Vulnerability Database (NVD) [7], with software archives, such as Git, requires considerable efforts. As a result, studies with a large number of real vulnerabilities are scarce.

Even if such a corpus is constructed, it needs to be continuously updated in order to account for the most recent issues and being generally up-to-date. Additionally, having a code base that has been reported as vulnerable, does not really help comprehension and education. Such tasks can be eased by considering the way such vulnerabilities have been fixed (patched). Studying the way developers fixed vulnerable code instances is insightful and a good starting point for understanding its root causes. In other words, investigating vulnerability fixes, the origin of the vulnerabilities, their types and characteristics can be determined.

We present VulData7, a framework and a dataset of 2,800 real vulnerable code instances of 4 security critical open source systems, i.e., Linux Kernel, WireShark, OpenSSL, SystemD. The framework includes and links all reported vulnerabilities in the NVD database with their related code instances and program versions, and aims at supporting source code analysis and empirical research. The name VulData7 stands for the (Vul)nerabilities (Data)set(7[1]) and its purpose is to enable the automated, continuous collection (collection over time) and linking of reported vulnerabilities (in NVD and Git repository). In summary, this paper makes the following contributions:

- VulData7 provides a set of real vulnerabilities for 4 security critical systems. The current version includes 2,800 vulnerabilities and 1,600 patches.

[1]In French 7 is pronounced as set so VulData7 stands for VulDataSet

- VulData7 brings together related code, its commits and all related information, i.e., CVE number, vulnerability description, CWE number (if applicable), time of creation, time of last modification, CVSS severity score, bug ids (if existing), list of impacted versions.
- VulData7 is automated. Once configured, it extracts and links information from the related software archives (through Git and NVD reports) to create a dataset that is continuously updated with the latest information available.
- VulData7 includes all reported and mentioned (in the software archives) vulnerabilities. Special care was taken in order to make the dataset as complete as possible (with respect to what can be mined) by searching both sources of information (links on NVD and Git messages). As a result we managed to mine vulnerabilities with assigned CVE that have not yet been recorded in NVD.
- VulData7 is flexible and easily extensible. Our framework links NVD with Git and thus, it involves little effort in configuring and importing data from additional projects. It includes all available processed and "raw" information (commit hashes, commit timestamps, commit messages and fixes - files in their states before and after fix), in order to be useful and extensible for research purposes. It also retrieves the complete related code bases so that it eases complete analyses.
- VulData7 provides a friendly interface for retrieving the related information. It includes utilities (such as XML exports, Git utilities, CWE Importer and others) for common analysis tasks which eases the access and analysis of the set.

II. Vulnerabilities

Secure development is key for almost every software organization. Industry usually adopts dedicated procedures to minimize and prevent security vulnerabilities. Here, we focus on code-based vulnerabilities as these are responsible for the majority of the exploits [8]. But, what exactly is a vulnerability?

Vulnerabilities are usually missing or insufficient checks, unhanded exceptions and bugs. Since vulnerabilities are of many forms, there are many definitions ("security bug" [9] or "software weakness" [10]). We give the CVE definition [11]:

"*An information security "vulnerability" is a mistake in software that can be directly used by a hacker to gain access to a system or network.*"

We now detail the acronyms we use through the paper (CVE, NVD, CVSS and CWE):

- The Common Vulnerability Exposures (CVE) is actually an index (reference system) for indexing vulnerabilities that have been publicly disclosed. It has been established and maintained by the National Institute of Standard and Technology (NIST). CVE references with a unique key every disclosed vulnerability. NVD references approximately 7,900 vulnerabilities only for the year 2017.
- The National Vulnerability Database (NVD) is a database that records the vulnerabilities referenced by CVE. It has been established by the U.S. government and provides vulnerability metadata such as the CVSS and CWE.
- The Common Vulnerability Scoring System (CVSS) is a number representing the severity of a vulnerability. In a sense CVSS captures the main characteristics and potential consequences of a vulnerability.
- The Common Weakness Enumeration (CWE) [12] is a community initiative that lists reported types of software weaknesses. CWE is used by NVD in order to categorize vulnerabilities.

III. VulData7 Framework

VulData7 is a tool that brings together vulnerability reports, vulnerable files and their patches for a given project. The tool is automated, it retrieves, stores and updates the sought data with the latest available information. In short the main information that can be retrieved is the following:

- CVE number
- Vulnerability description
- CWE number (if applicable)
- time of creation
- time of last modification
- CVSS severity score
- Bug ids (if existing)
- list of impacted versions
- list of commits that fixed the vulnerability. These contain the *commit hash, timestamp and message*, and the *commit fixes* (files in their states before and after fix).

A high level view of the VulData7 architecture and process can be described as follows: For a given project P, VulData7 clones in a local folder the git repository, connects to the NVD database and downloads all the available XML feeds for vulnerabilities for a given period of time (normally this should be in the range from 2002 to the current year). Then, VulData7 parses the XML feeds and retrieves all vulnerabilities reported for the specified period of time. For each vulnerability, it retrieves and saves all declared links (links mentioning bug reports or direct links to fixing commits). Then, VulData7 follows these links and retrieves the related commit information (for each vulnerability that had a link to a fixing commit). To account for missing links, the framework searches the version history of the project to identify (in the related commit messages) for a CVE Identifier or a bug id that was mentioned in the vulnerability report. Based on this information VulData7 retries vulnerable and fixed project versions.

To support continuous analysis, the framework can be automaticaly updated. Thus, it checks NVD for the latest information and updates its data. In case new data are there, VulData7 will pull vulnerabilities reported for P and create a new entry if there is a new vulnerability or update as necessary. In a nutshell, for each vulnerability entry, the framework checks for new links or commit fixes and/or bug id. It then pulls the repository, retrieves the new (vulnerable) commits, checks for bug ids and CVE Identifiers and updates the dataset.

TABLE I
DATASET STATISTICS

Systems	No Vulnerabilities	No Fixed Vulnerabilities	Average CVSS	Unique Vulnerable Files
Linux Kernel	2,082	1,202	5.41	1,508
Wireshark	531	265	4.99	221
OpenSSL	187	126	5.34	164
SystemD	9	5	5.76	5
Total	**2,809**	**1,598**	**5.38**	**1,898**

TABLE II
TOP-10 MOST FREQUENT VULNERABILITIES. EACH ENTRY REPRESENTS A PAIR OF THE FORM CWE ID (FREQUENCY).

Rank	Linux	Wireshark	OpenSSL	SystemD
1	264 (318)	20 (136)	310 (32)	20 (3)
2	200 (219)	399 (108)	399 (28)	264 (2)
3	399 (212)	119 (98)	116 (17)	362 (2)
4	119 (204)	189 (51)	200 (15)	787 (1)
5	20 (161)	400 (14)	20 (12)	119 (1)
6	189 (106)	74 (9)	189 (11)	-
7	362 (89)	476 (8)	362 (5)	-
8	476 (45)	134 (5)		-
9	284 (45)	200 (4)	-	-
10	416 (28)	-	-	-

TABLE III
MEANING OF MOST FREQUENT CWE IDS

CWE id	Description
264	Permissions, Privileges, and Access Controls
200	Information Exposure
399	Resource Management Errors
119	Improper Restriction of Operations within the Bounds of a Memory Buffer
20	Improper Input Validation
189	Numeric Errors
400	Uncontrolled Resource Consumption ('Resource Exhaustion')
310	Cryptographic Issues
116	Improper Encoding or Escaping of Output
362	Concurrent Execution with Shared Resource and Improper Synchronization

IV. DATA AVAILABLE

At the time of writing VulData7 contains data for 4 major security critical projects. While the framework is completely automated, we restrict our analysis on these projects because we are interested in collecting a large number of instances (in a per project basis) and our tool is (currently) working only with Git. Thus, we mined C programs, which tend to have more reported vulnerabilities. Nevertheless, the purpose of VulData7 is to provide a framework to support the collection and mining of vulnerabilities.

We collected data for the following four projects:

- **Linux kernel**: this is a major project that is now shipped in billions of systems as it is embedded in all Android devices. It should be one of the biggest, if not the biggest, open source system involving more than 19.5 million lines of code. Linux kernel involves many security aspects and it is ranked in the second place in CVE. The project is also mature (it started in 1991) and was the first project to switch to git in 2005 as git was created for it. According to our data, the project has 2082 reported vulnerabilities.
- **Wireshark**: is packet analyser enabling the analysis of network traffic, protocols and interface controllers. It is mainly used for troubleshooting of related network issues and to support development. It is available on all operating systems and is open source. The project firstly named ethereal was renamed following a fork as Wireshark in 2006. Since then 531 vulnerabilities were reported for the project.
- **OpenSSL**: is a widely used library that provides implementations of the SSL and TLS protocols (used extensively in communications). The project code is not that big (it contains approximately 650k lines of codes) but due to its criticality [13] it is often subject to attacks. The project started in 1998 and migrated to Git in 2013. The migration was successful and no significant loss of information occurred making possible to access all the versioning information directly from the git history. Currently, the project involves 187 vulnerabilities.
- **SystemD**: is the service manager for the Linux operating system. As such, its main goal is to unify the services and configurations of the Linux systems. It is used in the VulData7 tool as an example project. So far, 5 vulnerabilities have been reported for this system.

Overall, the descriptive statistics of our data are shown in Table I. The table records details about the number of vulnerabilities (column "No Vulnerabilities"), number of vulnerabilities with available patches (column "No Fixed Vulnerabilities'), the average severity score of the collected vulnerabilities (column "Average CVSS'), the average score of the collected vulnerabilities with patches (column "Average CVSS Fixed") and the number of unique vulnerable files involved (column "Unique Vulnerable Files").

In total our data contain 2,809 vulnerabilities and for 1,598 of them we retrieved a patch. These account for a collection of 1,898 vulnerable files with an average severity of 5.34.

Table II records the 10 most frequent types of vulnerabilities (according to CWE categorisation) per project. Every entry on this table represents a pair of the form vulnerability type (CWE id) and the frequency it appeared in the project. The list (description) of the vulnerability types (CWE id) is given in Table III.

V. DATASET STRUCTURE

The dataset can be accessed in two different ways, either through a JAVA API, either through a generated XML file.

A. Java API

Upon the creation (or the update) of the dataset, the user will receive a Data7 Java object. This object contains information about the project, some required data that are used by the tool to optimise its update action and the dataset in the form of a VulnerabilitySet Object.

Among the data used by the tool to optimise its update, two can be used for other purposes (i) a mapping of bug ids to commit hashes and (ii) a mapping of all CVE identifiers that were found in commit messages and are not present in

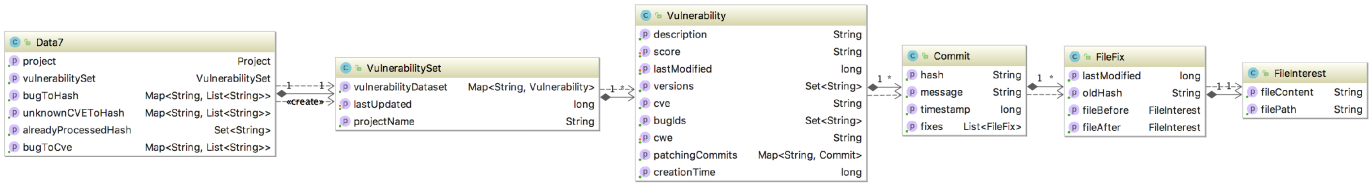

Fig. 1. VulData7 API

the CVE database. While the first one (i) can be used to create a Bug Dataset (see section VIII for details), the second one (ii) offer the possibility to observe vulnerable fixes before the release of the vulnerability report. As an example, in its latest run, our tool found that the commit [14] was made to address the CVE-2018-10840 which is not yet public. A glimpse at the commit message informed us that there was an issue with ext4 extended attribute. This list can thus be seen as a way to retrieve data related to all new vulnerabilties.

The VulnerabilitySet object is the dataset itself and is composed of the following elements: a list of every vulnerability ever reported for the chosen project (list of Vulnerability object) and the time of the last update. A Vulnerability object contains all of the information mentioned in Section III, which in essence includes the information that was found in the vulnerability reports. The Vulnerability object also includes a list of all the commits (Commit object) that were reported as fixing commits. A Commit object includes the hash, the message, the timestamp of the commit and a list of files that were modified by it (FileFix Object). A FileFix object records information on the time of last modification before the given commit and its corresponding previous hash as well as the file in its state before and after commit (FileInterest object). A FileInterest object contains the text of the file and its fullPath in the project.

All information related to the fields of each object that is accessible through the API is presented (as a UML diagram) in Figure 1.

B. XML exporter

Once the dataset is created the user is offered the possibility to export the data to an XML file. The generated XML file contains only vulnerabilities for which fixes were found. The file also includes all relevant information from the NVD database.

The schema of the generated file is presented in Listing 1.

VI. Using the tool

A. Installation, generation and export

The tool can be downloaded from GitHub. The latest version can be found at https://github.com/electricalwind/data7/releases. VulData7 uses maven and Java (version 8 or higher). To proceed with the installation, the user should type mvn install in a terminal at the location of the project (where it was downloaded). The project can then be used from any maven project by adding the dependency presented in Listing 2.

Succeding with the compilation, the user needs to define a path (to the place where the binary will be saved) and create an instance of an importer. These steps are quite simple as demonstrated on the following Java code (Listing 3).

As show in Listing 4, to export to XML format, an instance of an exporter must be created before calling the XML export function (Listing 4).

B. Integrating other tool or database

The VulData7 tool offers the possibility to export the data to another tool through listeners (which should be provided by the user). Indeed, when creating or updating a dataset, the user has the possibility to declare Listeners for the update. These listeners should use the DatasetUpdateListener interface. Thus, a potential listener could use the notifications to populate a SQL database.

Listing 1. XML schema

```
<?xml version="1.0" encoding="UTF-8"
    standalone="no"?>
    <data7 last_updated="YYYY-MM-DD HH:mm:ss
        CEST" project="project name">
      <cve id="CVE-YYYY-XXXXXX"
          last_modified="timestamp">
        <cwe></cwe>
        <score></score>
        <description></description>
        <affectedVersions>
          <version></version>
        </affectedVersions>
        <bugs/>
        <patches>
          <commit hash="aaaaaaa"
              timestamp="xxxxxxx">
            <message></message>
            <files>
              <file>
                <before hash="aaaaaaaa"
                    path="src/file.c">
                Content of the file
                </before>
                <after path="src/file.c">
                Content of the file
                </after>
              </file>
            </files>
          </commit>
        </patches>
      </cve>
    </data7>
```

Listing 2. Dependency to add in pom.xml

```
<dependency>
    <groupId>lu.jimenez.research</groupId>
    <artifactId>data7</artifactId>
    <version>1.1</version>
</dependency>
```

Listing 3. Generating a dataset

```
ResourcesPath path = new ResourcesPath("Path
    To Save Tour Data into");
Importer importer = new Importer(path);
Data7 data7 =
    importer.updateOrCreateDatasetFor
                    (CProjects.LINUX_KERNEL);
```

Listing 4. Exporting to xml

```
Exporter exporter = new Exporter(path);
exporter.exportDatasetToXML(data7);
```

VII. Additional Utilities

To ease analysis, VulData7 includes three custom-made libraries. These include:

- **Git Utils**: This library is coded in Kotlin and provides useful functions related to the mining of Git repositories. In short, the library provides methods related to the retrieval of files from specific commits, the retrieval of commits touching a file, gitBlame, use developer history class, delta history etc.
- **Misc Utils**: This library has been coded in Kotlin and contains useful functions for common tasks such as downloading a file, unzipping a file, normalizing a (directory) path and getting the recursive list of directories.
- **CWE Importer**: This library collects and reports data related to the CWE (types of vulnerabilities). The library can be invoked by calling *Importer.getListOfCWE();*. This call downloads (from NVD) the descriptions of the CWE, which are parsed and stored. Data related to the hierarchy of CWE types are also collected.

Further details about the utilites supported by the VulData7 framework can be found in the website of the tool.

VIII. Bug Collector

The framework supports the collection of software defects (other than vulnerabilities). The purpose of this tool is to collect data related to potential defects and their patches. Indeed (as we already explained) to retrieve the highest possible number of vulnerabilities in a project, VulData7 links the references of bug ids (mentioned in vulnerability reports) with commit messages. Using the same process, VulData7 can also collect defects mentioning (in the commit messages) bug ids and bug fixes.

Bug Collector stores all the related information in a separate dataset. In particular, the Bug Collector includes a list of commits that fixed a bug which contains their hash, timestamp, message and patches (files in their states before and after fix). Table IV records descriptive statistics about the collected bugs. Further details about the bug collector can be found in its GitHub page.

TABLE IV
Defect Statistics (defects not declared as vulnerabilities)

Systems	No Defects	No Fixed Defects	Unique Defective Files
Linux Kernel	3,160	5,193	2,428
Wireshark	3,871	8,019	1,907
OpenSSL	2,442	7,741	1,733
SystemD	1,868	3,538	925
Total	**11,341**	**24,491**	**6,993**

IX. Discussion

VulData7 is an ongoing project aiming at the automatic mining, analysis and evaluation of software defects and security vulnerabilities. As such, the current version has some potential limitations and use cases, which we discuss in this section.

A. Limitations

The dataset is automatically constructed by mining software archives. This results in some noise in our data. We have no guarantee that what we retrieve is in fact vulnerability patches or that the vulnerabilities have been really fixed. Additionally, we made no attempt to prune or minimize the commits of the retrieved versions to the most likely causes. Despite these issues, due to the number of the collected data our data provide a good starting point.

Another limitation regards the retrieved data, which may include some duplicated instances (due to commits residing in different branches). This fact depending on the performed analysis may be or may not be an important issue. Our dataset also does not include any timeline navigation of commits or other repository mining analysis tasks. These can be easily performed programmatically, but the current version does not suppport them. We plan to resolve these issues in the near future.

B. Potential Use Cases

The VulData7 project aims at the automatic collection of real vulnerable code instances with the intention to support empirical research. The tool leverages the experience from our research projects and we believe that it will be a valuable resource to the community. Previous versions of VulData7 have been used in our work on prediction modelling [15] and analysis of vulnerabilities [16]. Extensions of the tool (Bug Collector) have been used on the analysis of defects with natural language models [17]. Overall, we envision that VulData7 can have the following Use Cases:

- **Prediction modelling:** We believe that VulData7 can be used in prediction modelling research [15], [18], [19]. We have designed the tool to provide a collection of vulnerable, fixed (potentially non-vulnerable), buggy (potentially non-vulnerable) and clear (files never declared as vulnerable) files. Our utilities provide the ability to automatically retrieve, balance and analyse components (for both training and evaluation), tasks that are frequent in such research works.

- **Defect analysis:** VulData7 categorises the retried vulnerabilities according to their CWE ID. It also collects and distinguishes vulnerable from other buggy files. Such a categorization can help the analysis, categorization and study of different types of bugs and vulnerabilities. It can also serve as the starting point for taxonomizing and classifying defects for particular projects.
- **Assessment mechanism:** VulData7 is intended to provide the means to support empirical research. Therefore, its natural application is to assess the effectiveness of related techniques. For instance, static analysis tools, fuzzing tools and other bug finding techniques can be evaluated against the most recent reported issues. Another important benefit of our tool is that it contributes and to some extent anables the reproducibility and replicability of empirical research [6], [20].
- **Education:** An interesting use case of this work regards education. Developers, students and generally practitioners can use the collected instances in order to be informed, learn and comprehend the most important and recent security issues related to the projects they study.

X. Conclusion

We presented VulData7, a framework and dataset supporting the continuous collection and analysis of security vulnerabilities. Currently, the dataset involves 2,800 reported vulnerabilities, with 1,600 fixes, for 4 large security critical systems. We made a considerable effort in making this toolset automated, extensible and easy to use. Our goal is to provide the community with the means to support research and analysis at a large-scale.

Special care was taken in order to make VulData7 usable for research. We provide the data and tools to fetch the requested "raw" and processed data, i.e., metadata, source codes (complete or partial), configuration files etc., in order to make easy the inclusion and application of source code analysis tools. As the framework was built on top of Git and NVD it is simple to include additional open source projects. The addition of new projects requires a simple configuration (seting the appropriate links and paths), if the new projects are on Git (with reported vulnerabilities on NVD) and a couple of extentions for other software archives.

Collecting vulnerability patches can be considered a good starting point for automated solutions in the area. For instance, vulnerabilities can be categorized, comprehended and empirically analysed. Yet, VulData7 provides all needed tools to isolate related changes and ease their analysis.

The inclusion of additional (source code) management systems such as Subversion [21], Mercurial [22] is part of future work. Such an extension will increase the size of the dataset and the number of analysed software systems.

VulData7 is provided under the Apache License (Version 2.0) and is publicly available on GitHub:

https://github.com/electricalwind/data7

References

[1] Bug in openssl opens two-thirds of the web to eavesdropping. [Online]. Available: http://arstechnica.com/security/2014/04/critical-crypto-bug-in-openssl-opens-two-thirds-of-the-web-to-eavesdropping/

[2] M. Howard and S. Lipner, *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press, 2006.

[3] G. McGraw and B. Potter, "Software security testing," *IEEE Security & Privacy*, vol. 2, no. 5, pp. 81–85, 2004.

[4] M. Zitser, R. Lippmann, and T. Leek, "Testing static analysis tools using exploitable buffer overflows from open source code," in *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2004, Newport Beach, CA, USA, October 31 - November 6, 2004*, 2004, pp. 97–106. [Online]. Available: http://doi.acm.org/10.1145/1029894.1029911

[5] B. Dolan-Gavitt, P. Hulin, E. Kirda, T. Leek, A. Mambretti, W. K. Robertson, F. Ulrich, and R. Whelan, "LAVA: large-scale automated vulnerability addition," in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, 2016, pp. 110–121. [Online]. Available: https://doi.org/10.1109/SP.2016.15

[6] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell, *Experimentation in Software Engineering*. Springer, 2012. [Online]. Available: https://doi.org/10.1007/978-3-642-29044-2

[7] National vulnerability database:. [Online]. Available: https://nvd.nist.gov

[8] G. McGraw, "Automated code review tools for security," *IEEE Computer*, vol. 41, no. 12, pp. 108–111, 2008.

[9] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai, "Bug characteristics in open source software," *Empirical Software Engineering*, vol. 19, no. 6, 2014.

[10] Ics/scada top 10 most dangerous software weaknesses. [Online]. Available: http://www.toolswatch.org/wp-content/uploads/2015/11/ICSSCADA-Top-10-Most-Dangerous-Software-Weaknesses.pdf

[11] Definition of vulnerability. [Online]. Available: https://cve.mitre.org/about/terminology.html

[12] Cwe home page. [Online]. Available: https://cwe.mitre.org/data/

[13] Openssl usage statistics. [Online]. Available: https://trends.builtwith.com/Server/OpenSSL

[14] Cve-2018-10840 fixing commit:. [Online]. Available: https://github.com/torvalds/linux/commit/8a2b307c21d4b290e3cbe33f768f194286d07c23

[15] M. Jimenez, M. Papadakis, and Y. L. Traon, "Vulnerability prediction models: A case study on the linux kernel," in *16th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2016, Raleigh, NC, USA, October 2-3, 2016*, 2016, pp. 1–10. [Online]. Available: https://doi.org/10.1109/SCAM.2016.15

[16] ——, "An empirical analysis of vulnerabilities in openssl and the linux kernel," in *23rd Asia-Pacific Software Engineering Conference, APSEC 2016, Hamilton, New Zealand, December 6-9, 2016*, 2016, pp. 105–112. [Online]. Available: https://doi.org/10.1109/APSEC.2016.025

[17] M. Jimenez, M. Cordy, Y. L. Traon, and M. Papadakis, "On the impact of tokenizer and parameters on n-gram based code analysis," in *34th IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2010*, 2018.

[18] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Trans. Software Eng.*, vol. 38, no. 6, pp. 1276–1304, 2012. [Online]. Available: https://doi.org/10.1109/TSE.2011.103

[19] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Trans. Software Eng.*, vol. 37, no. 6, pp. 772–787, 2011. [Online]. Available: https://doi.org/10.1109/TSE.2010.81

[20] Z. Mahmood, D. Bowes, T. Hall, P. C. R. Lane, and J. Petric, "Reproducibility and replicability of software defect prediction studies," *Information & Software Technology*, vol. 99, pp. 148–163, 2018. [Online]. Available: https://doi.org/10.1016/j.infsof.2018.02.003

[21] Enterprise-class centralized version control for the masses. [Online]. Available: https://subversion.apache.org/

[22] Mercurial. [Online]. Available: https://www.mercurial-scm.org/