SOFTWARE VULNERABILITIES: AN EMPIRICAL CLASSIFICATION BASED ON PROGRAMMING LANGUAGE

Simone Scalabrino[1], Gabriele Bavota[2], Massimiliano Di Penta[3], Rocco Oliveto[1]
[1]University of Molise, Italy; [2]Free University of Bolzano, Italy; [3]University of Sannio, Benevento.

The identification of software vulnerabilities, *i.e.,* source code bugs that could have security implications, is a crucial aspect to consider when developing any kind of application. A lot of effort has been devoted in the last years to the definition of approaches for the automatic detection of vulnerabilities (1). For instance, Avancini and Ceccato (2), Antoniol (3) and Thomé *et al.* (4) defined detection approaches based on dynamic analysis, using search-based techniques. Scandariato *et al.* (5) and Walden *et al.* (6) defined approaches for vulnerability prediction. Besides approaches for vulnerability detection, several studies have been conducted to classify software vulnerabilities. Tsipenyuk *et al.* (7) defined a first taxonomy of software vulnerabilities, while the MITRE organization provided a ranked list of the most dangerous vulnerabilities that plague software systems (8). Such a classification has inspired the present work. Specifically, the analysis performed to identify such ranked lists, while considering many aspects, like potential impact and exploitability, does not take into account the differences among programming languages. This limits the usefulness that such a list could provide to software developers. Indeed, different programming languages are used for different purposes, which means that they are threatened by different vulnerabilities. In this work, we plan to bridge this gap by performing a large empirical study to define new ranked lists of vulnerabilities, based on specific programming languages and based on how frequently each vulnerability is fixed. The study findings can be useful for both a researcher who wants to study software vulnerabilities and a manager, who wants to know what are the main security threats of a project developed in a specific programming language.

The *context* of the study is represented by: *(i)* the change log history of all the projects stored in GitHub, one of the most important hosting services for software repositories; *(ii)* the vulnerabilities reported in the CWE (Common Weakness Enumeration) identified by MITRE. Such a list has been defined in 2011 and contains over 1000 different kinds of software vulnerabilities. In order to make the study feasible we decided to use only a subset of the identified vulnerabilities focusing the attention on those classified as most dangerous, i.e. the top 25 vulnerabilities. It is worth noting that a ranked list of the most dangerous vulnerabilities is also provided by OWASP (9). However, we decided to use the ranked list by MITRE because the considered vulnerabilities cover several kinds of applications, while the vulnerabilities considered by OWASP are related to web applications only (8). To the date of the study was performed the number of hosted projects is about 31M.

The study was guided by the following research questions:

- **RQ$_1$:** *To what extent the list of top 25 vulnerabilities identified by MITRE corresponds to the list of top 25 fixed vulnerabilities in the analysed projects?*
- **RQ$_2$:** *Are different programming languages affected by different software vulnerabilities?*

In order to answer both the above research questions, as a first step, we analysed all the commit events occurred from February 2011 to December 2015 on all the projects stored in GitHub. Such an analysis was carried out by using the data provided by the GitHub Archive (10), *i.e.* «a project to record the public GitHub timeline, archive it, and make it easily accessible for further analysis» (10). As a second step, we identified the commits related to the fixing of vulnerability, by automatically analysing the commit message. Specifically, we classified a commit as a vulnerability fixing if in the commit message the name of a vulnerability (belonging to the top 25 vulnerabilities) co-occurs with the keyword "fix". Such an approach classified 120,209 commits as vulnerability fixing. Such commits were then grouped by the specific vulnerabilities fixed. Thus, at the end of this process we obtained a list of commits grouped by the fixed vulnerability.

In order to answer **RQ$_1$**, we sorted the vulnerabilities according to the number of commits that fix them and compared the achieved ranked list with the top 25 vulnerabilities list identified by MITRE. As for **RQ$_2$**, we analysed the files modified in each commit and tagged each commit with the programming languages of such files. After that, we grouped the commits by the programming language of the related files. Then, for each group we built the ranked list of fixed vulnerabilities. Finally, the list of top fixed vulnerability obtained for each programming language are compared each other.

Figure 1 shows the comparison between the top 25 vulnerabilities identified by MITRE and the most fixed vulnerabilities identified in GitHub. The analysis of the results indicates a lack of agreement between the two ranked lists. For example, the most fixed vulnerability (*Cross-Site request forgery*) is the 12th most

Name	Top 25 fixed	MITRE's top 25
Cross-site request forgery	1	12
Cross-site scripting	2	4
Buffer overflow	3	3
Integer overflow/Wraparound	4	24
SQL-Injection	5	1
Missing authorization	6	6
Path traversal	7	13
Use of potentially dangerous function	8	18
Incorrect authorization	9	15
Open redirect	10	22
Hash without salt	11	25
Hard-coded credentials	12	7
OS-Command Injection	13	2
Uncontrolled format string	14	23
Unnecessary privileges	15	11
Incorrect calculation of buffer size	16	20
Incorrect permission assigned for critical resource	17	17
Broken or risky cryptographic algorithm	18	19
Imporper restriction of Excessive authentication attempts	19	21
Missing authentication for critical function	20	5
Functionality from untrusted control sphere	21	16
Download code without integrity check	22	14
Reliance on untrusted inputs in a security decision	23	10
Unrestricted upload of files with dangerous type	24	9
Missing encryption of sensitive data	25	8

Fig. 1 - The number in the each column indicates the position in the specific ranked list. The vulnerabilities are sort by their position in the top 25 fixed vulnerabilities.

dangerous vulnerability according to MITRE classification; at the same time, the 5th most critical vulnerability according to MITRE (*Missing authentication for critical function*) is just the 20th most fixed vulnerability: actually, no commit reports a fix for such a vulnerability. This result could mean two things: on one hand, the developer's perception of the criticality of software vulnerabilities (and, so, their effort in detection and fixing) does not always match with the actual danger. On the other hand, something might be changed about vulnerabilities and how they tend to affect software in the last years, i.e. from 2011, when the list has been defined by MITRE, to 2015.
Figure 2 shows a comparison between the ranked lists on top fixed vulnerabilities of two programming languages, C and PHP. The difference is clear. For example, *Cross-Site Request Forgery* is the most fixed vulnerabilities in PHP (and among the most fixed in other languages like Ruby, Javascript and Python), but it is less important in C (or C++): in such languages, *Buffer Overflow* and *Integer Wraparound* are the most fixed vulnerabilities. In this case, the explanation is clear: PHP is widely used for implementing web applications, while C is usually preferred for desktop applications or embedded systems. Thus, the presence of certain vulnerabilities is strongly related to the programming language used and, consequently, to the kind of the application.
This preliminary study successfully shows that different programming languages can actually be affected by different vulnerabilities and that the vulnerability that developers fix more often are not always the most spread and dangerous. The results achieved open new research directions aiming at defining approaches for vulnerability identification that are programming language sensitive.

Rank	PHP	C
1	Cross-site request forgery	Buffer overflow
2	Cross-site scripting	Integer overflow/Wraparound
3	SQL-Injection	Cross-site scripting
4	Missing authorization	Use of potentially dangerous function
5	Integer overflow/Wraparound	Cross-site request forgery
6	Buffer overflow	SQL-Injection
7	Path traversal	Open redirect
8	Open redirect	Missing authorization
9	Incorrect authorization	Path traversal
10	Hash without salt	OS-Command injection
11	Use of potentially dangerous function	Incorrect authorization
12	Hard-coded credentials	Hash without salt
13	OS-Command injection	Uncontrolled format string
14	Broken or risky cryptographic algorithm	Incorrect calculation of buffer size
15		Hard-coded credentials
16		Unnecessary privileges

Fig. 2 - Each column shows the ranked list for a specific language. Some vulnerabilities are missing: this means that we did not find any commit that talk about such vulnerabilities.

1) M. Pistoia, S. Chandra, S.J. Fink, E. Yahav (2007) IBM Systems Journal, 46(2): 265-288
2) A. Avancini, M. Ceccato (2010) Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems, pagg. 65-71
3) G. Antoniol (2009) 2nd International Workshop on Search-Based Software Testing, Denver, Colorado
4) J. Thomé, A. Gorla, A. Zeller (2014) SBST 2014 Proceedings of the 7th International Workshop on Search-Based Software Testing, pages 5-14
5) R. Scandariato, J. Walden, A. Hovsepyan, W. Joosen (2014) IEEE Transactions on Software Engineering, 40(10): 993-1006
6) J. Walden, J. Stuckman, R. Scandariato (2014) IEEE 25th International Symposium on Software Reliability Engineering (ISSRE) DOI: 10.1109/ISSRE.2014.32
7) K. Tsipenyuk, B. Chess, G. McGraw (2005) IEEE Security and Privacy, 3(6): 81-84
8) http://cwe.mitre.org/top25/ (visited in December 2015)
9) OWASP (2013) The ten most critical web application security risks
10) http://www.githubarchive.org/ (visited in December 2015)