

SOFTWARE VULNERABILITIES: AN EMPIRICAL CLASSIFICATION BASED ON PROGRAMMING LANGUAGE


Simone Scalabrino Gabriele Bavota Massimiliano Di Penta Rocco Oliveto

CONTEXT

A software vulnerability is a weakness (i.e., a bug) which entails a reduction of the security of a program and, sometimes, of an entire system. A vulnerability can be exploited by an attacker in two ways:

- 1) she can acquire private information (loss of **privacy** constraint)
- 2) she can perform actions not allowed to her (loss of **integrity** constraint)

MITRE's CWE (Common Weakness Enumeration) defined a categorization of software vulnerabilities. Such an organization identified more than **1,000** vulnerability types. Besides, in 2011 MITRE defined a list of the **25 most dangerous vulnerabilities**. Nevertheless, our conjecture is that a universal ranking of vulnerabilities is not sufficient: vulnerabilities can strongly depend on the programming language used to develop a system. In this study we plan to check if our conjecture is correct, i.e., if different programming languages are plagued by different kinds of vulnerabilities.




In 2014 the **heartbleed** vulnerability was disclosed. It is estimated that 17% of secure web servers were affected, about

500,000 units.

From 2012 to 2014 the privacy of all internet users was unconsciously in danger.


CASE STUDY DESIGN

RQ₁



Is there an agreement between the most fixed vulnerabilities and MITRE's top 25 vulnerabilities?

RQ₂



Is there a difference among programming languages as for the most fixed vulnerabilities?

RESULTS

Top 5 fixed vulnerabilities

| | |
|---|----------------------------|
| 1 | Cross Site Request Forgery |
| 2 | Cross-Site Scripting |
| 3 | Buffer Overflow |
| 4 | Integer Overflow |
| 5 | SQL Injection |


MITRE top 5 vulnerabilities

| | |
|------------------------|---|
| SQL Injection | 1 |
| OS Command Injection | 2 |
| Buffer Overflow | 3 |
| Cross-Site Scripting | 4 |
| Missing Authentication | 5 |

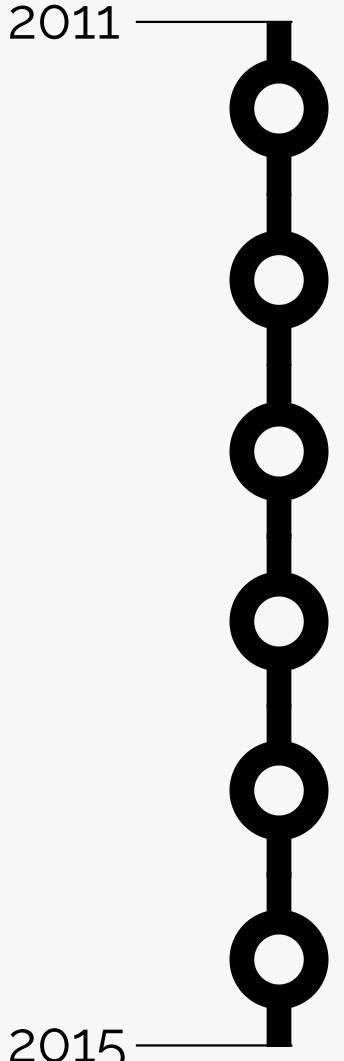
There is a lack of agreement between the two ranked lists. This result could mean two things: on one hand, the developer's perception of the criticality of software vulnerabilities (and, so, their effort in detection and fixing) does not always match with the actual danger. On the other hand, something 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.

31,000,000+ GitHub projects

25 most dangerous software vulnerabilities




2011




2015

Extracted **120,209** vulnerability fixing commits

Tagged with **vulnerability** and **language**



| | |
|---|----------------------------|
| 1 | Buffer Overflow |
| 2 | Integer Overflow |
| 3 | Cross-Site Scripting |
| 4 | Use of dangerous functions |
| 5 | Cross-Site Request Forgery |



| | |
|---|----------------------------|
| 1 | Cross Site Request Forgery |
| 2 | Cross-Site Scripting |
| 3 | SQL Injection |
| 4 | Missing authorization |
| 5 | Integer Overflow |

The difference between programming languages is clear in some cases. For example, PHP programs are plagued above all by *Cross-Site Request Forgery* and other web-related vulnerabilities, while programs developed in C are affected mostly by vulnerabilities, such as *Buffer Overflow*, related to a wrong handling of memory. The explanation is clear: PHP is widely used for implementing web applications, while C is usually preferred for desktop applications or embedded systems. Therefore, the presence of certain vulnerabilities is strongly related to the programming language used.

CONCLUSION AND FUTURE WORK

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 next step will be to check if vulnerability detection tools were used in order to find the vulnerabilities analyzed in this study, in order to understand their actual benefits and problems. Finally, the results achieved open new research directions aiming at defining approaches for vulnerability identification that are programming language sensitive.





Simone Scalabrino
University of Molise
s.scalabrino@studenti.unimol.it



Gabriele Bavota
University of Bolzano
gabriele.bavota@unibz.it



Massimiliano Di Penta
University of Sannio
dipenta@unisannio.it



Rocco Oliveto
University of Molise
rocco.oliveto@unimol.it