Extended Abstract: Source Code Patterns of Software Vulnerabilities

Felix Schuckert

April 18, 2017

1 Introduction

Looking into the CWE [2] top 25 show *SQL Injection* (CWE-89) in first place, *Buffer Overflow* (CWE-120) in third place and *Cross Site Scripting* (CWE-79) in the fourth place. Vulnerabilities of these categories are not new. They already exist for a long time. To discover the reason why the same vulnerabilities are occurring, we investigated the source code from open source projects. Similar methods and operations are grouped and are called source code patterns. Our work shows which source code patterns occur in real life projects, to provide a data set that can be compared to existing vulnerability data sets like SAMATE SARD [5]. The first question to be answered is: How do such source code patterns look like? The next question is, are there any special cases that are not typical for these vulnerabilities? Real source code samples are investigated to get answers to these questions.

We use a crawler to get source code from open source projects. CVEs in the *Cross Site Scripting* category using PHP as programming language are reviewed. We choose PHP because it is a popular programming language. Additionally, in PHP a lot of vulnerability from different categories are possible. For the *Buffer Overflow* category, we choose vulnerabilities from Firefox because the source code is accessible and they provide a good bug reports with bugzilla. We categorize the manual review results in source code patterns.

2 Data Sources

A manual review requires a selection of data. We focus on vulnerabilities that are tracked in the *CVE* [1] database. For Buffer Overflows Firefox provides enough vulnerabilities for doing a manual review process. CVEs between 2010 and 2015 (six years) are used as data samples. Firefox provides enough corresponding CVEs to cover the Buffer Overflow category. 50 random CVEs are chosen in the given time frame. Cross Site Scripting does not have an open source project that has enough corresponding CVEs. For this category a crawler looks up *CVE Details* [3] for entries that have Cross Site Scripting and contain any GitHub [4] *commit* links. It found in 4,279 Cross Site Scripting CVEs 177 *GitHub* links. For each link the patch that fixes the vulnerabilities is looked up. The most used programming language is PHP with 101 patches. From these 101 patches 50 are chosen randomly and are used as our data set for the review process.

For SQL injection 37 CVEs with corresponding *GitHub* confirm links are found. These are used to look up for source code patterns.

3 Results

For each vulnerability category different parts were reviewed. For SQL Injection **sources**, **string concatenation**, **sinks**, **failed sanitization** and **fixes** were reviewed. Different methods that are used for these parts are illustrated. Very similar are the source code patterns for Cross Site Scripting. They were reviewed for **sources**, **string concatenation**, **sinks** and **fixes**. Instead for Buffer Overflows the patterns for **type of errors**, **sinks** and **fixes** were reviewed. The source are missing because of the complexity of the source code. Additionally, there is no reliable tool to do a reverse data flow analysis that would be required to get all possible sources.

Some special cases are explained that show unexpected patterns. The results show that some developers still miss basic knowledge about software security. Buffer Overflow instead usually have some kind input check (sanitization). But these checks failed because of different kind of errors. A common mistake were Integer overflows and resulting values were used in the checks or for allocating memory. In future the results will be used for insecurity refactoring to create different code patterns that occurred in real projects.

References

[1] Common Vulnerabilities and Exposures. https://cve.mitre.org/.

[2] Common Weakness Enumeration. https://cwe.mitre.org/.

[3] CVE Details. https://www.cvedetails.com/.

[4] GitHub. https://github.com/.

[5] SAMATE - SARD. https://samate.nist.gov/SARD.

All links were last followed on March 24, 2017.