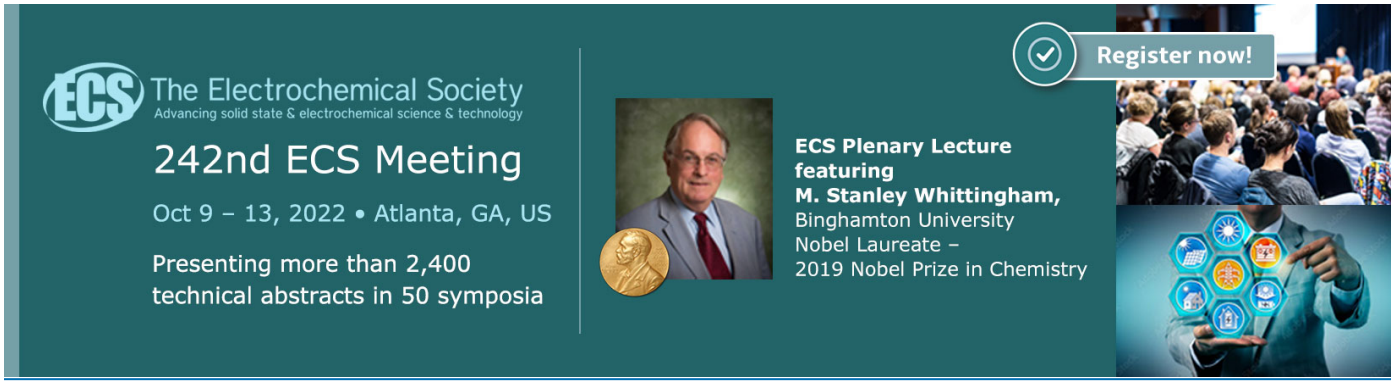
A Survey of the Key Technology of Software Vulnerability Mining



IOP Conf. Series: Journal of Physics: Conf. Series **1187** (2019) 052031 doi:10.1088/1742-6596/1187/5/052031

A Survey of the Key Technology of Software Vulnerability Mining

Dansheng Lin[1], Xiaoquan Wu[1], Qinqin Wu[1*], Ye Liu[1], Jijun Zeng[1], Jinjun Tan[2]

[1]Information Center, Guangdong Power Grid Company Limited, Guangzhou, Guangdong,China

[2]Guangdong Information Technology Security Evaluation Center, Guangzhou, Guangdong, China

[*]Corresponding author's e-mail: 149679600@qq.com

Abstract. In the current cyberspace, vulnerability has attracted the widespread attention. This paper briefly introduces the typical software vulnerability mining techniques. Based on the existing research work, this paper puts forward the whole idea of the research on the vulnerability mining technology of software in the future, and then makes some research on some key points and key technologies respectively. In this paper, a brief study of the software vulnerability mining technology is given, which is helpful to carry out the follow-up research work.

1. Introduction

Software vulnerabilities manifest in many forms, including race conditions, buffer overflows, integer overflows, dangling pointers, poor input validation (e.g., SQL injection, cross-site scripting), information leakage, violation of least privilege and other access control errors, use of weak random numbers in cryptography, protocol errors, and insufficient authentication[1-2]. Discovering vulnerabilities has become an extremely widespread activity and new threats are constantly being faced. Vulnerabilities in software represent a serious risk for information systems. According to the National Vulnerability Database (NVD), which maintains records of all acknowledged vulnerabilities of software products on the market, more than 90,000 vulnerabilities have been discovered since 1997. The number of vulnerabilities is increasing rapidly due to the development of new hacking techniques. Between 2010 and 2015, around 80,000 vulnerabilities were newly registered in the major database known as the CVE (Common Vulnerability Enumeration)[3].

In recent years, the scope of security threats has also been expanded. As cataloged in the National Vulnerability Database, prominent recent examples of serious software vulnerabilities include Heartbleed (CVE-2014-016), Shellshock (CVE-2014-6271), the glibc buffer overflow (CVE-2015-7547), VENOM (CVE-2015-3456), and Microsoft Malware Protection Engine vulnerability (CVE-2017-0290). According to the website CVE Details, there have been 937 vulnerabilities between January and August 2017 to date. Ten percent of those vulnerabilities were in the "critical" (highest severity) category with assigned Common Vulnerability Scoring System (CVSS) scores of 9 or 10.

There are many kinds of vulnerability mining techniques. It is very difficult to complete the analysis work by using only one vulnerability mining technique. Generally, several vulnerability mining techniques are optimized to find a balance between efficiency and quality

2. Vulnerability Mining Techniques

2.1. Manual Analysis

Manual analysis is a grey box analysis technique. Aiming at the target program being analyzed, special input conditions are manually constructed, output and target state changes are observed, and vulnerability analysis techniques are obtained. Input includes valid and invalid inputs, including normal output and abnormal output. Abnormal output is the premise of a loophole or a loophole in the target program. The change of abnormal target state is also a harbinger of vulnerabilities and the direction of deep excavation. Manual analysis is highly dependent on the experience and skills of the analyst. Manual analysis is mostly used for target programs with human-computer interaction interface. Manual analysis is mostly used in Web vulnerability mining [4-5].

2.2. Fuzzing

Fuzzing technology[6-7] is an automatic software testing technology based on defect injection. It uses black box analysis technology, uses a large number of semi-effective data as the input of the application program, and uses the abnormality of the program as a sign to discover the possible security vulnerabilities in the application program. Semi-valid data means that the necessary identification part and most of the data of the target program under test are valid, and the intentionally constructed data part is invalid. When the application processes the data, it may have errors, which may lead to the collapse of the application or trigger the corresponding security vulnerabilities.

According to the characteristics of the target, Fuzzing can be divided into three categories: 1) Dynamic Web page Fuzzing, for ASP, PHP, Java, Perl and other web page programs, also includes B/S architecture applications built using this technology, the typical application software is HTTP Fuzz; 2) File format Fuzzing, for various document formats, typical application software is PDF Fuzz.

The construction method of Fuzzer software input is similar to that of black-box testing software. Boundary values, strings, additional strings of file head and end can be used as basic construction conditions. Fuzzer software can be used to detect a variety of security vulnerabilities, including buffer overflow vulnerabilities, integer overflow vulnerabilities, formatted string and special character vulnerabilities, competition conditions and deadlock vulnerabilities, SQL injection, cross-site scripting, RPC vulnerability attacks, file system attacks, information leakage, etc.

Compared with other technologies, Fuzzing technology has the advantages of simple thought, easy to understand, easy to find vulnerabilities, easy to reproduce vulnerabilities, and no false alarm. At the same time, it also has all the shortcomings of black box analysis, and has some problems such as non-universal, long construction test cycle and so on. Common Fuzzer software includes SPIKE Proxy, Peach Fuzzer Framework, HTTP Fuzzer of Acunetix Web Vulnerability Scanner, OWASP JBroFuzz, WebScarab, etc.

2.3. Patch Comparison

Patch matching techniques[8-9] is mainly used by hackers or competitors to find bugs that have been corrected but not yet disclosed by software publishers. It is a common technical means used by hackers before exploiting bugs. Security announcements or patch release instructions generally do not specify the exact location and cause of vulnerabilities, and it is difficult for hackers to exploit vulnerabilities only according to the announcement. Hackers can determine the location of vulnerabilities by comparing binary files before and after patching, and then combine with other vulnerability mining technologies to understand the details of vulnerabilities. Finally, they can get the attack code of vulnerability exploitation.

Simple comparison methods include binary byte and string comparison, and the comparison after reverse engineering of the target program. The first method is suitable for the comparison of minor changes before and after patching, and is commonly used for the analysis of vulnerabilities caused by string changes and boundary value changes. The second method is applicable to the analysis of vulnerabilities caused by the change of function parameters according to decompilation. Neither of these

two methods is suitable for document modification. Complicated comparison methods include Tobb Sabin's graphical comparison based on instruction similarity and Halvar Flake's structured binary comparison. Some unstructured changes in files, such as changes in buffer size, can be found and displayed graphically.

The commonly used patch comparison tools are Beyond Compare, IDACompare, Binary Differing Suite (EBDS), BinDiff, NIPC Binary Differ (NBD). In addition, a large number of advanced text editing tools have similar functions, such as Ultra Edit, HexEdit and so on. These patch comparison tools are based on string comparison or binary comparison techniques.

2.4. Static Analysis

Static analysis technology[10-11] is a typical white-box analysis technology, which analyses and detects the source program of the target being analyzed and finds the security vulnerabilities or hidden dangers in the program. Its methods mainly include static string search and context search. Static analysis process is mainly to find incorrect function calls and return states, especially function calls that may not have boundary checks or incorrect boundary checks, which may cause buffer overflow functions, external call functions, shared memory functions and function pointers.

For open source programs, security flaws in programs can be found by detecting file structures, naming rules, functions and stack pointers that do not conform to security rules. When the target is not accompanied by the source program, it is necessary to reverse-engineer the program, obtain reverse-engineering code similar to the source code, and then search. Using a method similar to the source code, vulnerabilities can also be found in programs. This kind of static analysis method is called disassembly scanning. Because of the use of the underlying assembly language for vulnerability analysis, in theory, all computer vulnerabilities can be found. For programs that do not open source code, it is often the most effective way to find security vulnerabilities. However, this method also has great limitations. The expanding feature library or dictionary will result in large result set and high false alarm rate. At the same time, this method focuses on the analysis of the features of the code, but does not care about the function of the program, and will not have the analysis and inspection of the function and program structure.

2.5. Dynamic Analysis

Dynamic analysis technology[12-13] originates from software debugging technology. It uses debugger as a dynamic analysis tool. But unlike software debugging technology, it often deals with the analyzed program without source code or the analyzed program which has been reverse-engineered.

Dynamic analysis needs to run the target program in the debugger, and discover the vulnerabilities by observing the running state of the program, memory usage and register values during execution. The general analysis process is divided into code flow analysis and data flow analysis. Code flow analysis mainly traces the target program code flow dynamically by setting breakpoints to detect defective function calls and their parameters. Data flow analysis is to trigger potential errors by constructing special data.

In the process of dynamic analysis, dynamic code replacement technology can be used to destroy the process of program operation, replace function entrance and function parameters, which is equivalent to constructing semi-effective data, so as to find the hidden defects in the system. Common dynamic analysis tools include SoftIce, OllyDbg, WinDbg and so on.

3. Typical Technique Applications

3.1. Acunetix Web Vulnerability Scanner

We can use Acunetix Web Vulnerability Scanner to mine vulnerabilities, this software provides some predefined parameters Library of Fuzz operation, which can be easily used by beginners and analysts. The process is as follows:

1) Define the HTTP request (Request), that is, define the web page URL needed to access.

2) Define operation parameters (Add generator), that is, define string expressions that may cause vulnerabilities, such as: $password, $passwd, $token;
3) Insert in request, which binds defined parameters to a search strategy;
4) Define Fuzzer Filters, which binds parameters to HTTP requests;
5) Scan (Start);
6) Waiting for software to return matching items, which are possible vulnerabilities.

Through above steps, a possible loophole in a web page is discovered.

3.2. Patch Comparison Technology Application

In October 23, 2008, Microsoft released the patch of MS08-067, which was listed as serious. The security update solves a secret report loophole in the server service. If a user receives a special RPC request on the affected system, the vulnerability may allow remote code execution. The process of vulnerability mining using patch comparison technology is as following: First, keep an original file, then install a new patch, extract the same and new files, you can use the software to compare. After comparison, 3 functions that have been modified are found. The comparison software lists three function names, 0.25, 0.67 and 0.94 respectively, and lists the similarities before and after patching. By comparing the results, we can construct parameters and observe the behavior before and after patching. Finally, we find that two of the three functions are directly related to vulnerabilities.

4. Conclusion

Vulnerability mining technology is derived from software testing theory and software development debugging technology, which can greatly improve the security of software. Third-party organizations and technology enthusiasts in the field of network security also use this technology to find various software vulnerabilities and release them to the public in time, contributing to improving the overall level of information security, but vulnerability mining is also a double-edged sword, which has become the mainstream technology for hackers to crack software. The development of vulnerability mining technology has broad prospects. With more and more attention to information security, software development technology is more and more advanced, and new analysis methods will emerge.

Acknowledgments

This work was financially supported by GDKJXM20162130(037800KK52160003).

References

[1] Khalili A, Sami A, Azimi M, et al. Employing secure coding practices into industrial applications: a case study[J]. Empirical Software Engineering, 2014, 21(1):1-13.
[2] Pham N H, Nguyen T T, Nguyen H A, et al. Detection of recurring software vulnerabilities[C]// Ieee/acm International Conference on Automated Software Engineering. ACM, 2010:447-456.
[3] NVD, https://nvd.nist.gov/.
[4] Zhao M, Grossklags J, Liu P. An Empirical Study of Web Vulnerability Discovery Ecosystems[C]// ACM Sigsac Conference on Computer and Communications Security. ACM, 2015:1105-1117.
[5] Holm H, Ekstedt M, Sommestad T. Effort Estimates on Web Application Vulnerability Discovery[C]// Hawaii International Conference on System Sciences. IEEE, 2013:5029-5038.
[6] Takanen A, Demott J, Miller C. Fuzzing for software security testing and quality assurance[M]. Artech House, 2008.
[7] Zhi-Yong W U, Wang H C, Sun L C, et al. Survey on Fuzzing[J]. Application Research of Computers, 2010, 27(3):829-832.

[8] Guo H, Wang Y Y, Pan Z L, et al. Research on Detecting Windows Vulnerabilities Based on Security Patch Comparison[C]// International Conference on Instrumentation & Measurement. IEEE, 2016:366-369.
[9] Hua X L, Long S Y, Feng G, et al. Security patch comparison techniques based on graph isomorphism theory[J]. Journal of Computer Applications, 2006, 26(7):1623-1622.
[10] Xia Y M. Security Vulnerability Detection Study Based on Static Analysis[J]. Computer Science, 2006, 33(10):279-282.
[11] Li P, Cui B. A comparative study on software vulnerability static analysis techniques and tools[C]// IEEE International Conference on Information Theory and Information Security. IEEE, 2011:521-524.
[12] Tang H, Huang S, Li Y, et al. Dynamic taint analysis for vulnerability exploits detection[C]// International Conference on Computer Engineering and Technology. IEEE, 2010:V2-215 - V2-218.
[13] Kim S, Kim R Y C, Park Y B. Software Vulnerability Detection Methodology Combined with Static and Dynamic Analysis[J]. Wireless Personal Communications, 2016, 89(3):777-793.