International Journal of Research Radicals in Multidisciplinary Fields (IJRRMF), ISSN: 2960-043X
Volume 1, Issue 1, January-June, 2022, Available online at: www.researchradicals.com

Application Vulnerability Detection Techniques

Amit Sengupta

Senior Delivery Manager, Software Engineering, Capgemini America

ABSTRACT

In this modern era of technology where data and its integrity are vital for organizations, software security has become a major area to focus on in the software life cycle. Organizations must preserve the program's security to ensure the computer program's availability, authenticity, and data integrity delivered to the clients. The major focus in software security processes is to find the vulnerabilities displayed in source code prior to the production phase of the software product. Recognizing the bugs present in the code in the early stages of the software lifecycle may help resolve the vulnerability findings in the computer program and help the software developers settle those bugs. This detection process is effective at runtime but can also be performed in the production phase where the computer program is under development and partially implemented. A static code analysis process is used to detect vulnerabilities. It can be done computerized or evaluated physically by development and testing teams. The use of source code scanning tools that are mostly automated for detecting vulnerabilities is utilized in this paper. These tools review the source code for its quality based on several code metrics and identify bugs present in the program. Unlike dynamic analysis methods, static code analysis helps find the security vulnerabilities in the initial stages of the software life cycle, where the software product is in the production phase and static analysis does not require code to be in the execution state.

Keywords: - Code Scan, Vulnerability, Security, Secure Coding, Threat Modeling, Sql injection, Cross site scripting, Bug Prediction, Risk Analysis, Risk Severity.

INTRODUCTION

Preserving the security of the software products have been the foremost basic tasks also deemed to be the critical ones for the organizations now-a-days. In the recent years the number of known software vulnerabilities has grown its numbers vastly. Computer security alludes to steps taken to ensure that the program is guaranteed from attacker's malicious intents that may influence the proper functioning of the computer program.

Different analysts and cyber security professionals have classified these security vulnerabilities, explained that software security vulnerability can be deemed as a flaw in the source code written by developers which in turn may provide unauthorized access to the attacker and obstruct the behavior of program. Subsequently, software developers need to find and settle these bugs found within the code before it is moved to production phase.The two main broadly classified ideas of detecting vulnerabilities are –

1. **Static code Analysis** - Static code analysis, also known as source code analysis or static code review, is the process of detecting bad coding style, potential vulnerabilities, and security flaws in a software's source code without actually running it, a form of white-box testing. It will enable your teams to detect code bugs or vulnerabilities that other testing methods and tools, such as manual code reviews and compilers, frequently miss. The fast feedback loop is a key tenet of the DevOps movement. Static code analysis also helps you achieve a quick automated feedback loop for detecting defects that, if left unchecked, could lead to more serious issues.This is not only useful for checking code styles; it can also be used for static application security testing (SAST).

2. **Dynamic code Analysis** - Dynamic code analysis, also called Dynamic Application Security Testing (DAST) is designed to test a running application for potentially exploitable vulnerabilities. DAST tools to identify both compile time and runtime vulnerabilities, such as configuration errors that only appear within a realistic execution environment.A DAST tool uses a dictionary of known vulnerabilities and malicious inputs to “fuzz” an application. Examples of these potentially malicious inputs include - SQL Injection, Long input strings, Negative and large positive numbers, unexpected input data. As the application runs, it is bombarded by these potentially malicious inputs, and the DAST tool analyzes the responses from the application. If the application has a negative response to an input, then the DAST tool records the identified vulnerability.Since DAST tools are executed on a

running application they can detect a wide range of potential vulnerabilities. This includes vulnerabilities that are difficult or impossible to detect in source code, such as memory allocation, buffer overflow etc.

Many software developers and analysts have claimed that the static code analysis process is more advantageous and effective than dynamic analysis for finding security flaws in source code.

It is less demanding for the software developer since the method of settling these flaws found can be done in early stages, additionally, decreases the amount of time spent and the cost put within the organization for settling these flaws. Static code analysis enjoys the utilization of computerized scanning tools, or it can be done physically if needed.

APPLICATION VULNERABILITY DETECTION AND RISK ANALYSIS

Code Scanning and Severity Ranking – Code scannersbreak down the code into smaller pieces called tokens. Through tokens which are similar to words in this context it understands the code. A token may contain a single character or a reserved keyword for that language (like class in Java). Trailing space and program semantics such as comments are often discarded by scanners. The tokens are then parsed into simple words in a language to get the grammar of a language. Parser identifies these tokens and validates them so that in a sequence they provide the grammar and organized in a tree like fashioned structure called Abstract Syntax Tree.

The tree itself focuses only on the logical structure of the program and the least insignificant details like indentation and parenthesis are abstracted.The syntax tree is parsed, and the static analysis tools look for a specific pattern in the code that results in vulnerabilities.

Various taint paths that lead to flaws are captured and using a bug severity model and bug filter they can be filtered based on the bug severity.Bug filter is capable of matching instances in context to certain criteria. A filter can select a set of bug instances for including or excluding them in a report. Usually, they are used to exclude instances of a bug.

The filters can also be conditioned to combine two or more filter types by following clauses Or, And Not. It specifies a particular pattern or patterns to match a bug instance. The Rank element matches bug with having at a specified bug rank. The values range in between 1 and 20 where 1 to 4 are critical, 5 to 9 is high, 10 to 14 is medium and rest is deemed to be low risk factored bugs.

This enables filtering bugs associated within a specified class. It takes the class name as an attribute to filter. The inputs are broken into tokens using various patterns by scanners.

CVE Identification and Analysis - Common Vulnerabilities and Exposures (CVEs) provide a reference for publicly known information security vulnerabilities and exposures.Leveraging CSAF (Common Security Advisory Framework) documents for structured, machine-readable vulnerability information and OEM (Original Equipment Manufacturer) advisories we can stay updated with advisories published by device manufacturers for the latest vulnerability disclosures and patches. Below are high level steps for CVE Identification –

- Inventory Management: Keep an updated inventory of applications, including OS versions and firmware.
- Vulnerability Feeds and Databases: Regularly check vulnerability databases, CSAF feeds, and OEM advisories for new CVEs affecting your systems.
- Assessment: Evaluate the relevance and impact of identified CVEs on your software environment.

Risk Assessment by Practicing Chaos - Understanding potential exploits available for identified CVEs helps in assessing risk levels and prioritizing remediation efforts. Below are some of the sources to generate chaos experiments –

- Exploit Databases: Platforms like Exploit-DB provide a searchable database of publicly known exploits.
- GitHub: An invaluable resource for finding exploits and proof-of-concept code shared by the security research community.
- Metasploit Framework: A comprehensive toolkit for developing and executing exploit code against remote target devices.
- Google Hacking Database: A compilation of Google search queries that can uncover hidden information and vulnerabilities.

Once the experiments are executed a comprehensive risk analysisis performed following the below techniques –

- Search for Exploits: Use the identified CVEs as keywords to search across various databases and platforms.
- Analyze Exploit Code: Study available exploit code to understand the exploitation techniques and affected system components.
- Test in a Controlled Environment: If possible, replicate the vulnerable environment and test the exploit in a controlled setting to gauge its impact.

Code Quality Prediction Technique: -
Below are various source code metrics with module, class and method level granularity were used to measure software quality and predict errors. For example, McClure proposed to improve McCabe [17] by considering the number of control variables. We use most of the popular method-level code indicators to see the predictability of error codes and Gravitation.

Metrics	Description
LC	Number of source code lines without comments and blank lines
MA	Number of independent paths (logical complexity)
ML	Total variables and total comparisons in a predicate
NBD	Counting the depth of the most nested block
PI	Counting indentation of source code lines
FO	Counting the total number of methods called a given method
R	Measuring the readability of the code in the range of 0-1
D	Difficulty in writing or understanding the code
E	Effort in developing the code

The Shift Left Strategy for Code Quality: -
Developers can review the detected errors and fix them proactively in the development phase itself, to move into the next stage. Most code analysis tools are capable of working alongside in an integrated development environment (IDE). Anyhow, these tools are often used post development process and strategies need to be applied for analyzing performance beyond what is a completed software product.

Reviewers then analyses code to find bugs and policy violations. Querying or emphasizing through the model is done looking for specific properties or patterns that indicate a bug. Sophisticated symbolic execution techniques examine paths through control flow graphs. A data structure that represents a path that can be traversed during program execution. When Path Scouting detects an anomaly, an alert is generated. To model and explore an astronomical number of situational combinations, these automated analysis tools use a variety of strategies to ensure scalability.

The most important part of detection is to avoid getting used to failing dependency checks. Modern day build pipelines are integrated with automated vulnerability checks and the build would fail if there isa vulnerable dependency detected. Since the Open Worldwide Application Security Project (OWASP) dependency check primarily uses the National Vulnerability Database (NIST NVD) database, it sometimes struggles with false positives.

CONCLUSION

Bug detection and correction remains the main maintenance activity in software product development life cycle. However, among many bugs that are likely to exist in high severity codes are of great interest to developers because their consequences are the most important. In this article, we have highlighted source code metrics and automated code analysis techniques, as popular methods to predict buggy codes, to find the possibility of estimating bug severity respectively in most cases. The quality of this code correlates with your application security, resiliency and reliablity.

To endure quality, many improvement teams embrace techniques like automated code review integrated with their build pipelines. According to the State of Cloud Native Application Security Report, misconfiguration, and known unpatched vulnerabilities were responsible for the greatest number of security incidents in cloud native environments.Source code analysis could prevent half of the problems that often slip through the cracks in production. Rather than putting out fires caused by bad code, a better approach would be to incorporate quality assurance and enforce coding standards early in the software development life cycle using static code analysis.

REFERENCES

[1]. Servant, F., Jones, J.A.: Fuzzy fine-grained code-history analysis. In: Proceedings of the International Conference on Software Engineering (ICSE), pp. 746–757 (2017)
[2]. Sravan Kumar Pala, "Detecting and Preventing Fraud in Banking with Data Analytics tools like SASAML, Shell Scripting and Data Integration Studio", *IJBMV*, vol. 2, no. 2, pp. 34–40, Aug. 2019. Available: https://ijbmv.com/index.php/home/article/view/61
[3]. Wahono, R.S.: A systematic literature review of software defect prediction. Journal of software engineering 1(1),1–16 (2015)
[4]. Goswami, MaloyJyoti. "Leveraging AI for Cost Efficiency and Optimized Cloud Resource Management." International Journal of New Media Studies: International Peer Reviewed Scholarly Indexed Journal 7.1 (2020): 21-27.
[5]. Habib, A., Pradel, M.: Neural bug finding: A study of opportunities and challenges. arXiv preprint arXiv:1906.00307 (2019)
[6]. Chintala, Sathishkumar. "Explore the impact of emerging technologies such as AI, machine learning, and blockchain on transforming retail marketing strategies." Webology (ISSN: 1735-188X) 18.1 (2021).
[7]. Matter, D., Kuhn, A., Nierstrasz, O.: Assigning bug reports using a vocabulary-based expertise model of developers. In: 2009 6th IEEE International Working Conference on Mining Software Repositories, pp. 131–140 (2009)
[8]. Sravan Kumar Pala, Improving Customer Experience in Banking using Big Data Insights, International Journal of Enhanced Research in Educational Development (IJERED), ISSN: 2319-7463, Vol. 8 Issue 5, September-October 2020.
[9]. Ayewah, N., Pugh, W., Morgenthaler, J.D., Penix, J., Zhou, Y.: Using findbugs on production software. In: Companion to the 22nd ACM SIGPLAN conference on Objectoriented programming systems and applications companion, pp. 805–806 (2007)
[10]. Goswami, MaloyJyoti. "Utilizing AI for Automated Vulnerability Assessment and Patch Management." EDUZONE,Volume 8, Issue 2, July-December 2019, Available online at: www.eduzonejournal.com
[11]. Tian, Y., Lo, D., Sun, C.: Information retrieval based nearest neighbor classification for finegrained bug severity prediction. In: 2012 19th Working Conference on Reverse Engineering, pp. 215–224. IEEE (2012)
[12]. Ayyalasomayajula, Madan Mohan Tito, Sathish Kumar Chintala, and SailajaAyyalasomayajula. "A Cost-Effective Analysis of Machine Learning Workloads in Public Clouds: Is AutoML Always Worth Using?."International Journal of Computer Science Trends and Technology (IJCST) – Volume 7 Issue 5, Sep-Oct 2019
[13]. Sravan Kumar Pala, Use and Applications of Data Analytics in Human Resource Management and Talent Acquisition, International Journal of Enhanced Research in Management & Computer Applications ISSN: 2319-7463, Vol. 10 Issue 6, June-2021.
[14]. Ramay, W.Y., Umer, Q., Yin, X.C., Zhu, C., Illahi, I.: Deep neural network-based severity prediction of bug reports. IEEE Access 7, 46846–46857 (2019)
[15]. Chintala, S. "Evaluating the Impact of AI on Mental Health Assessments and Therapies." EDUZONE: International Peer Reviewed/Refereed Multidisciplinary Journal (EIPRMJ) 7.2 (2018): 120-128.
[16]. Antinyan, V., Staron, M., Sandberg, A.: Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time. Empirical Software Engineering 22(6), 3057–3087 (2017)
[17]. MMTA SathishkumarChintala, "Optimizing predictive accuracy with gradient boosted trees infinancial forecasting" Turkish Journal of Computer and Mathematics Education (TURCOMAT) 10.3 (2019).
[18]. Goswami, MaloyJyoti. "Challenges and Solutions in Integrating AI with Multi-Cloud Architectures." International Journal of Enhanced Research in Management & Computer Applications ISSN: 2319-7471, Vol. 10 Issue 10, October, 2021.
[19]. Yuan, H., Zheng, L., Dong, L., Peng, X., Zhuang, Y., Deng, G. (2020). Research and Implementation of Security Vulnerability Detection in Application System of WEB Static Source Code Analysis Based on JAVA. In: Xu, Z., Choo, KK., Dehghantanha, A., Parizi, R., Hammoudeh, M. (eds) Cyber Security Intelligence and Analytics. CSIA 2019. Advances in Intelligent Systems and Computing, vol 928. Springer, Cham.