International Journal of Computer Applications (0975 – 8887)
Volume 169 – No.8, July 2017

Evaluation of Software Vulnerability Detection Methods and Tools: A Review

Richard Amankwah
Jiangsu University
School of Computer Science $
Telcom Engineering

Patrick Kwaku Kudjo
Jiangsu University
School of Computer Science $
Telcom Engineering

Samuel Yeboah Antwi
Presbyterian College of
Education, Ghana

ABSTRACT
Software vulnerability remains a serious problem among industry players in the world today because of the numerous security related challenges it possess to end-users and stakeholders. Although previous studies have proposed various methods and tools that can be used in reducing or eliminating software vulnerability, those studies, however, raised several additional questions that need be addressed: (1) Can all the tools be used in curbing software vulnerabilities. (2) Can a specific tool detect all software vulnerabilities? To address these questions, we performed a detailed evaluation of the various software vulnerability detection methods and tools to find out their differences and similarities. Our studies also seeks to investigate the most efficient approach for detecting vulnerabilities based on previously proposed benchmarks and present some recommendations for future studies.

General Terms
Software Engineering, Information Security

Keywords
Benchmarks; Software Vulnerability; Vulnerability Detection

1. INTRODUCTION
Software vulnerability remains a serious problem faced by software companies and end users of software product. For the past years, there has been an increased reportage on several security vulnerabilities with high devastating effects on customers; this has brought to the public domain the need to focus on software vulnerability detection tools and methods. Software developers have developed a lot of methods and tools by using several approaches to detect and report these vulnerabilities that pose security threat to systems and users. The CERT/CC (Computer Emergency Response Team Coordination Center) reported that the economic loss caused by the intrusion events has reached about 6.66 billion US dollars in 2003 and this figure is still on the ascendancy with the passage of time. For example, there were a total of 7236 vulnerabilities in 2007, and this number has increased to 4110 by the end of the first two quarter of 2008 [1].

Although there is no universal definition for software vulnerability, previous studies have given varied explanation of the concept. Kuang et al.[2] defined software vulnerability as the "fault that can be viciously used to harm security of software systems". Krsul[3] also define software vulnerability as a defect that allows an attacker to violate an explicit or implicit security policy to achieve some impact. In another study Jimenez et al.[4] defined software vulnerability as a flaw, weakness or even an error in the system that can be exploited by an attacker in order to alter the normal behavior of the system. Schultz et al. [5] defined software vulnerability as ''a defect, which enables an attacker to bypass security measure". Finally the Organization of Internet Safety (OIS) defines security vulnerability as "a flaw within a software system that can cause it to work contrary to its documented design and could be exploited to cause the system to violate its documented security policy". The analysis of these definitions clearly indicates that software errors are the main causes of information security breaches. This is evident in the report presented in 2010 by researchers and expert from more than twenty five universities, international cyber security organizations about the twenty five (25) most dangerous software errors that enable cyber-crime. These errors were classified into three main categories; 1.Software Error based on insecure interaction between components 2. Software Error based on risky resource management 3. Software Error based on Porous Defenses. The recent cyber-attacks on institutions such as Google, SMEs, Universities, governmental organization, and home users were all attributed to software errors [5]. Thus software vulnerability is a subset of faults[6]and as such can be define based on the weakness, defect, errors, fault, and failures that occur in software. Having discussed the main elements that constitute software vulnerability, this paper also seeks to address some of the additional questions raised in those previous studies. The main focus of the study is to evaluate the various tools and methods proposed by previous studies to find out their differences and similarities. Evaluation of these vulnerability detection methods and tools is very key as it help us to: (1) identify which tool and method is suitable for a particular vulnerability detection (2) expand the said method or tool functionality (3) evaluate the weakness and strength of the methods and tools for detection (4) identify known and unknown software vulnerabilities.

Our studies also seeks to investigate the most efficient approach for detecting vulnerabilities based on previously proposed benchmarks and present some recommendations for future studies. The remaining sections of the paper are organized as follows. Section 2 discusses the common causes of software vulnerability. Section 3 presents a presents a detailed analysis of software vulnerability methods. Section 4 presents the evaluation of software vulnerability detection methods. We reports on the vulnerability detection tools in Section 5. The evaluation of the tools and is presented in Section 6. Section 7 concludes the study.

2. COMMON CAUSES OF SOFTWARE VULNNERABILITY
Analyzing the causes of software vulnerability significantly helps reduce vulnerabilities in software[7]. Several studies have empirical verify the causes of software vulnerability over the past decade. Krsul et al. [8] investigated and presented some common causes of system vulnerabilities, this include buffer overflow and IP Fragmentation. Buffer overflow occurs when a program tries to copy some data from one object into another but does not check if the destination object

size is large enough to contain the source object. IP Fragmentation on the other hand can be categorized into two forms, vulnerabilities that occur during the design of protocol and IP fragmentation vulnerabilities known as teardrop. In 2001, e-Eye Digital security as well presented a report about a buffer-overflow that caused vulnerability in Microsoft IIS Web Servers[9]. The Industrial Control System Cyber Emergency Response Team (ICS-CERT) in 2015 reported the following major causes of vulnerabilities [11].

i. Insufficient Entropy: this occurs when attackers guess the random numbers generated by the system and gain unauthorized access to a system.
ii. Use of cryptographically weak Ping: this occurs when a non-cryptographic PRNG is used in a cryptographic context; this exposes the cryptography to certain types of attacks.
iii. Authentication by pass by spoofing: this form of attack is mainly caused by improperly implemented authentication schemes that are subject to spoofing attacks
iv. Improper check for unusual or exceptional conditions

Again based on the report presented in 2010 by expert on the twenty five most dangerous software errors, the following were the major cause of software Vulnerabilities identified.

A. Software Error based on insecure interaction between components
i. Improper neutralization of special elements used in an SQL Command
ii. Improper neutralization of special elements used in an OS command
iii. Improper neutralization of input during Web page Generation
iv. Unrestricted upload of file with dangerous type
v. Cross-site request forgery
vi. URL redirection to untrusted site

B. Software Error based on Risky Resource Management
i. Buffer copy without checking size of input
ii. Improper limitation of a pathname to a restricted Directory
iii. Inclusion of functionality from Untrusted control Sphere
iv. Use of potentially dangerous function
v. Incorrect calculation of buffer size
vi. Uncontrolled format string
vii. Integer overflow or wraparound

C. Software Error Based on Porous Defenses
i. Missing authentication for critical function
ii. Missing authorization
iii. Use of hard-coded credentials
iv. Missing Encryption of sensitive data
v. Reliance on untrusted inputs in a security decision
vi. Execution with unnecessary privileges
vii. Incorrect authorization
viii. Incorrect permission assignment for critical resource
ix. Use of a broken or risky cryptographic algorithm
x. Improper restriction of excessive authentication attempts
xi. Use of one-way hash without a salt

Furthermore, we sample eight causes of software vulnerability reported by the National vulnerability database.

i. Input validation Error (IVE) (boundary condition error(BCE), buffer overflow(BOF): such types of vulnerabilities include failure to verify the incorrect input and read or write involving an invalid memory address
ii. Access Validation error (AVE): these vulnerabilities cause failure in enforcing the correct privilege for a user
iii. Exceptional condition Error Handing (ECHE): these vulnerabilities arise due to failures in responding to unexpected data or conditions.
iv. Environmental Error (EE): These vulnerabilities are triggered by specific conditions of the computational environment.
v. Configuration Error (CE): These vulnerabilities result from improper system settings.
vi. Race Condition Error (RC): These are caused by the improper serialization of the sequences of processes.
vii. Design Error (DE): These are caused by improper design of the software structure.
viii. Others: Includes vulnerabilities that do not belong to the types listed above, sometimes referred to as nonstandard.

3. VULNERABILITY DETECTION METHODS

This section of the paper will present an in-depth analysis of the tools used in detecting vulnerabilities in software applications. The tools and techniques are used in detecting if there are systems gaps that could be capitalized by an attacker to compromise the security of the system or that of the platform the system runs on.

3.1 Fuzzing

Fuzzing is a security detection method that takes an invalid input or random input into the application and output a behavior that is not expected and identity error in the program and suspected vulnerability. This is because it expected that every program contain some level of vulnerability that need to be detected. The key to fuzzing is data generation where pertinent test are carried out to crash the source program and also to choose the right tools to monitor the process. But currently, there is more twist to fuzzing where developers analyze the executable codes rather than the source code to detect vulnerabilities. According to [10] Fuzzed data generation can be performed in two ways. They can be generated randomly by modifying correct data without requiring any knowledge of the application details. This method is known as Black box fuzzing and was the first fuzzing concept. On the other hand, White box fuzzing consists in generating tests assuming a complete knowledge of the application code and behavior. A third type is Gray box fuzzing which stands between the two methods aiming to take advantages of both. It uses only a minimal knowledge of the behavior target. Data generation is the key to fuzzing, according to the data generation methods, fuzzing can be categorized as random fuzzing, mutation-based fuzzing, generation-based fuzzing and direction-based fuzzing. Random fuzzing is the simplest fuzz testing technique, a stream of completely random input data is send to the program under test. The input data can be sending as command line options, events, or network packets. This type of fuzzing is, in particular, useful for test how a program reacts on large or invalid input data. While random fuzzing can find already severe vulnerabilities, modern fuzzers do have a detailed understanding of the input format that is expected by the program under test. Mutation-based fuzzing is one type of fuzzing in which the fuzzer has some knowledge about the input format of the program under test: based on existing data samples, a mutation-based fuzzing tools generated new variants, based on a heuristics, that it uses for fuzzing. The mutation algorithm is the key to improve the efficiency of fuzzing. Generation-based fuzzing generates program inputs according to some specifications. Compared to pure random-based fuzzing, generation-based fuzzing achieves usually a higher coverage of the program under test, in particular if the expected input format is rather complex and has checksums. Direction-based fuzzing use the program control flow to direct the fuzzing, also called test case generation fuzzing. SAGE [11] is the type of Direction-based fuzzing. First, it constructs an initial and valid input IN0, sends the input into program P, and symbol execution engine observes P's processes on IN0 and a path constraint that is in the form of logical formulas; secondly, it negates the path constraint encountered during execution, solves new constraint by a constraint solver, and create a new input IN1 whose execution path is different from IN0's; finally, it processes IN1 in the same way with IN0 and repeats the previous three procedures. There are lots of research [12] and tools on fuzzing, such as Sulley [13], SPIKE [14], Peach [15].

3.2 Web Application Scanners

Web application scanners are an automatic application that examines application on the web for security vulnerabilities. Web security is very difficult since its coverage runs through the public which includes unscrupulous users. Web application takes it input from Hypertext Transfer Protocol request which makes it processing difficult. The volatility of the input either correct or not correct is the most causes of web application vulnerabilities. Web application testing for vulnerability can be carried out by two different methods: white box testing which analysis the source code of the application manually by using tools such as FORTIFY [16], Ounce [16]or Pixy [16]. Once this is done manually it's very difficult because of the complex nature of programming languages and also sometime cannot detect all security vulnerabilities. Black box testing this is where the scanner uses fuzzing approach to detect vulnerabilities. It is sometimes called penetrating testing. The work of the web application scanners is that its examines the application by surfing through the web pages through penetrating testing which is analysis of the web application and come out with malicious input and further assesses it and sees its response. Web application scanners are mainly applied in the testing stage of the system development and it must be able to (1) identify a vulnerability in a the set web application (2) come out with a report what is to be carried out that lead to the vulnerability (3) come out with a low false positive ratio. Aside these web application scanners, the following commercial web application scanners can also be used in detecting software vulnerability, they are; AppScan WebKing WebInspectNTOspider[17].

3.3 Static Analysis Techniques

The use of web application for daily routine work and for commercial purposes is on the high side as the day goes by, this emerging phenomena unfortunately has also presented a lot of security issues where unscrupulous developers look for gap and weakness in web application as an avenue for attack. According to the most recent website security statistics report 63 percent of assessed websites are vulnerable, each having an average of six unsolved flaws [18]. In 2013, Open Web Application Security Project[19] and Common Vulnerabilities and Exposures [20] indicated that, cross site scripting (XSS) and SQL injection (SQLI) are the top ten most serious vulnerabilities in web based system. Static Analysis technique is a defensive and preventive technique that detects vulnerabilities in web application. The primary objective of this approach is to identify the weakness in the program source code before its actual use in the user's environment for the first time. This help to detect vulnerability early enough hence cutting down cost of rectifying it should it happen. The static analysis approach is used in performing the following activities: (1) assess the input code (2) applies set rules or algorithms also called inference (3) generates a list of vulnerabilities present in the program.

There are a lot of Static analysis approaches available which is very effective for detecting Buffer Overflow vulnerabilities before a program come out. Many static analysis approaches have been introduced in various research aims at detecting BOF vulnerabilities [21-23]. These approaches can be classify to these six main areas (1) inference technique (2) analysis sensitivity (3) Analysis granularity (4) soundness (5) completeness (6) language

3.4 Brick

Binary Run-time Integer Based Vulnerability Checker which detects integer based vulnerability at run-time**.** It is very effective approach which result gives low false positive and negative. BRICK process involve three stages: (1) its convert the binary code to intermediate representation VEX on Valgrind (dynamic binary instrumentation framework Valgrind[24] (2) intercept integer related statements at run-time, and record the necessary information (3) detect and locate vulnerability with a set checking scheme.

3.5 CRED: C Range Error Detector

One of the vulnerability detecting approach that is not widely use is the Dynamic Buffer Overrun Detector. This is because its lack the power to protect against all buffer overrun attacks, break existing code and also produce too high overhead. CRED: C Range Error Detector approach corrects the above in- competencies and finds all buffer overruns attacks. CRED proved effective in detecting buffer overrun attacks on programs with known vulnerabilities, and is the only tool found to guard against tested 20 different buffer overflow attacks [25]

4. EVALUATION OF SOFTWARE VULNERABILITY DETECTION METHODS

4.1 Metrics for Benchmarking

Benchmarks are standard tools that allow evaluating and comparing different systems, components and tools according to specific characteristics [26]. This helps all stakeholders in decision making in terms of selecting the right approach and for which vulnerability. The following are the benchmarks for evaluating the vulnerability detection methods: Time Cost, False Positive, False Negative, Coverage, Number of vulnerability Detected, False Prediction Rate and Complexity. Details of the performance comparison of the vulnerability detection methods are presented in figure: 1.

5. SOFTWARE VULNERABILITY DETECTION TOOLS

In other to produce quality application which is devoid of vulnerabilities, developer s uses software quality assurance tools which assist them to detect weakness in every part of the System Development Life Cycle. These tools can be obtained from the market and also some are open source product. According to Defense Information Systems Agency's (DISA) "Application Security Assessment Tool Market Survey," Version 3.0, July 29, 2004 [27] classified this variety of tool into:

5.1 Web Application Tools:

Web application scanner tools are a more specialized class of tool that focuses specifically on web applications only, and are not considered generalized network scanners. Examples of this type of tool and company are as:

i. AppScan DE by Watchfire
ii. N-Stealth by N-Stalker
iii. NTOSpider by NTObjectives
iv. Spike Proxy by Immunity
v. TestMaker by pushtotes
vi. WebScarab by OWASP

5.2 Web Service Tools:

Web service scanner tools are a relatively new class of tool whose purpose is the analysis of web service applications. Examples of this type of tool include:

i. SOAPscope by Mindreef
ii. SOA Test by Parasoft

5.3 Database Tools:

Database Scanner tools are a specialized tool used specifically to identify vulnerabilities in database applications. In addition to performing some "external" functions like "password cracking", the tools also examine the internal configuration of the database for possible exploitable vulnerabilities. Examples of this type of tool include:

i. AppDetective by Application Security Inc.

5.4 Developer Tools:

Developer tools are used to identify software vulnerability during development or after deployment. These tools consist of static source code analysis tool, disassemble debugger decompiles binary code/byte code analysis tool, and dynamic run-time analysis tools. Examples of Static Source Code Analysis Tools include:

i. BOON by D. Wagner
ii. BoundsChecker, Dev Partner by Compuware
iii. Code Assure by Secure Software Inc.
iv. CodeSurfer, CodeSonar by GrammaTech, Inc.
v. Eau Claire by Brian Chess
vi. Prevent/Extend by Coverity
vii. Cqual by Jeff Foster
viii. Flawfinder by David Wheeler
ix. Fortify Source Code Analysis by Fortify
x. ITS4 by Cigital
xi. K7 by Klocworks
xii. Jtest by Parasoft
xiii. PolySpace by PolySpace Technologies
xiv. Prexis by Ounce Labs, Inc.
xv. RATS by Secure Software
xvi. RSM Source code by Msquared Technologies
xvii. Splint by U. of Virginia
xviii. SPIdynamics
xix. Jlint by Artho.com
xx. PMD by InfoEther, Inc.
xxi. UNO by Bell Labs
xxii. xg++ by Stanford

5.5 Disassembler, Debugger, Decompiler tools include:

i. IDA PRO by DataRescue Inc.
ii. VmWareVitual Infrastructure by VmWare
iii. Boomerang by Boomerang Open Source Community Project

5.6 Examples of Binary/Bytecode Analyzer include:

i. AspectCheck by Aspect Security
ii. FindBugs by University of Maryland
iii. BugScan by LogicLab
iv. BEAST Binary Executable Analysis by Security Innovation

5.7 Static Analysis Tools

Here eight types of widely-used software vulnerabilities of static analysis tools from open source are chosen for analysis and comparison. First the main features of these tools are briefly described, and then we compared them from a technical point of view.

A. ITS4

ITS4 [5] is a tool based on lexical analysis technique. It maintains a vulnerability database to read out the contents of the database at runtime and compare with the program codes. The database can be added, modified and deleted.

B. SPLINT

SPLINT (Secure Programming Lint) [6] is the expansion of LCLINT tool (for detecting buffer overflows and other security threats). It employs several lightweight static analyses. SPLINT need to use notes to perform cross-program analysis. SPLINT set up models for control flow and loop structure by using heuristic technology.

C. UNO

UNO [7] uses model checking to find loopholes in the code. UNO is named for the first character of three software defects: the use of uninitialized variables, dereferencing Nil-pointers, and Out-of-bound array indexing.

D. CHECKSTYLE

Checkstyle is the most useful tool to help programmers write standard Java coding. Programmers can integrate Checkstyle in development environment and use it to automatically check whether the Java codes are standard. Checkstyle is configurable and can almost support all the coding standards.

E. ESC / Java

ESC / Java (Extended Static Checker for Java) [8] is a static detection tool based on theorem proving, and can find run-time error in Java code. Programmers can build ESC / Java into the program verification environment, or install ESC / Java plug-in in the Eclipse.

F. FindBugs

FindBugs [9] is an open source static detection tools,which check the class or JAR files? By comparing binary codes with the defect model set, FindBugs can detect latent problems. FindBugs is not to find loopholes through analyzing the form and structure of class files, but by using the visitor pattern. At present FindBugs contains about 50error pattern detectors.

G. PMD

PMD is an open source, rule-based static detection tool. PMD scans Java source codes and finds some potential problems, such as wrong code, duplicate code, fussy code or code to be further optimized. PMD includes a default rule set. In addition, it allows users to develop new rules and use

6. EVALUATIONS VULNERABILITY DETECTION TOOLS

Software quality is very important in application development, as a result there has been several tools developed to ensure that application developed are of good quality. Static analysis tools are one of such that detect vulnerabilities in applications without having to run the code. Figure 2 shows a detail comparison of the static analysis tools

7. CONCLUSION

In this paper we evaluated software vulnerability detection methods and tools. For this reason sample software vulnerability detection methods such fuzzing, scanning, static analysis CRED and BRICK were discussed. The strength and weakness of these methods were also compared. More so, static analysis detection tools were also discussed and their detection rate was also compared quantitatively. In this review I can conclude that there is no single software vulnerability detection method or tool that can defect weakness in a software product, each method or tool has its own advantages and disadvantages, I therefore suggest that moving forward there should be and integration of the detection methods and tools to complement each other whiles trying to improve the weakness of the individual approaches to enhance its efficiency.

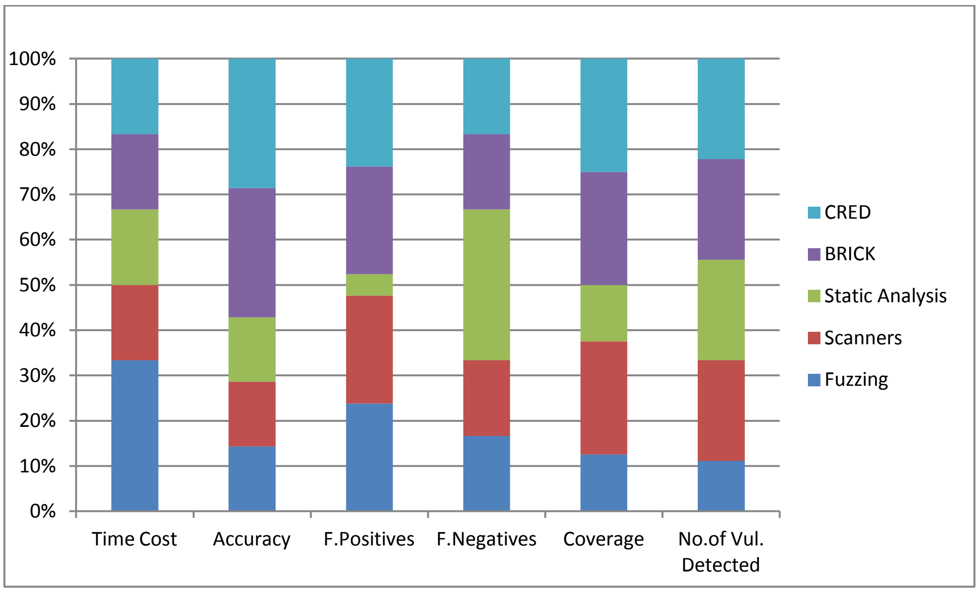

Figure I: Performance Comparison of Vulnerability Detection Methods

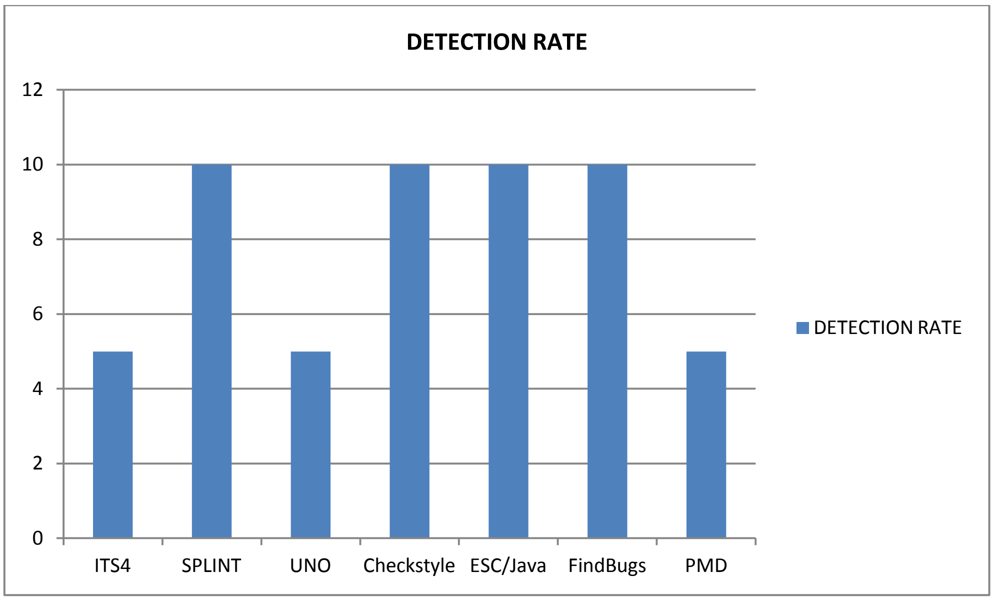

Figure II: Performance Comparison of Static Analysis Tools

8. REFERENCES

[1] M. Alnuaimi, M. A. Al-Fayoumi, and S. J. Aboud, "Protection of e-commerce Using Hybrid Tools."

[2] C. Kuang, Q. Miao, and H. Chen, "Analysis of software vulnerability," *WSEAS Transactions on Computers Research,* vol. 1, p. 45, 2006.

[3] I. V. Krsul, "Software vulnerability analysis," Purdue University, 1998.

[4] W. Jimenez, A. Mammar, and A. Cavalli, "Software Vulnerabilities, Prevention and Detection Methods: A Review1," *Security in Model-Driven Architecture,* p. 6, 2009.

[5] E. E. Schultz Jr, D. S. Brown, and T. A. Longstaff, "Responding to computer security incidents: Guidelines for incident handling," Lawrence Livermore National Lab., CA (USA)1990.

[6] G. McGraw, *Building Secure Software: How to avoid security problems the right way*: Addison-Wesley Professional, 2002.

[7] L. Ping, S. Jin, and Y. Xinfeng, "Research on software security vulnerability detection technology," in *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, 2011, pp. 1873-1876.

[8] I. Krsul, E. Spafford, and M. Tripunitara, "An analysis of some software vulnerabilities," in *Proceesings of the 21st NIST-NCSC National Information Systems Symposium*, 1998, pp. 111-125.

[9] M. Shaneck, "An Overview of Buffer Overflow Vulnerabilities and Internet Worms," *CSCI*, 2003.

[10] S. Bekrar, C. Bekrar, R. Groz, and L. Mounier, "Finding software vulnerabilities by smart fuzzing," in *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, 2011, pp. 427-430.

[11] P. Li and B. Cui, "A comparative study on software vulnerability static analysis techniques and tools," in *Information Theory and Information Security (ICITIS), 2010 IEEE International Conference on*, 2010, pp. 521-524.

[12] T. L. Munea, H. Lim, and T. Shon, "Network protocol fuzz testing for information systems and applications: a survey and taxonomy," *Multimedia Tools and Applications,* vol. 75, pp. 14745-14757, 2016.

[13] P. Amini and A. Portnoy, "Sulley-Pure Python fully automated and unattended fuzzing framework," ed: May, 2013.

[14] D. Aitel, "An introduction to SPIKE, The fuzzer creation kit," *presentation slides), Aug,* vol. 1, 2002.

[15] M. Eddington, "Peach fuzzing platform," *Peach Fuzzer,* p. 34, 2011.

[16] M. Vieira, N. Antunes, and H. Madeira, "Using web security scanners to detect vulnerabilities in web services," in *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, 2009, pp. 566-571.

[17] E. Fong and V. Okun, "Web application scanners: definitions and functions," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, 2007, pp. 280b-280b.

[18] S. Gupta and L. Sharma, "Exploitation of cross-site scripting (XSS) vulnerability on real world web applications and its defense," *International Journal of Computer Applications,* vol. 60, 2012.

[19] M. K. Gupta, M. Govil, and G. Singh, "Static analysis approaches to detect SQL injection and cross site scripting vulnerabilities in web applications: A survey," in *Recent Advances and Innovations in Engineering (ICRAIE), 2014*, 2014, pp. 1-5.

[20] C. Vulnerabilities, "Exposures,"The Standard for Information Security Vulnerability Names"," *Common Vulnerabilities and Exposures: The Standard for Information Security Vulnerability Names. url: http://cve. mitre. org,* 2007.

[21] N. Dor, M. Rodeh, and M. Sagiv, "CSSV: Towards a realistic tool for statically detecting all buffer overflows in C," in *ACM Sigplan Notices*, 2003, pp. 155-167.

[22] B. Hackett, M. Das, D. Wang, and Z. Yang, "Modular checking for buffer overflows in the large," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 232-241.

[23] J. Viega, J. Bloch, T. Kohno, and G. McGraw, "Token-based scanning of source code for security problems," *ACM Transactions on Information and System Security (TISSEC),* vol. 5, pp. 238-261, 2002.

[24] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," in *ACM Sigplan notices*, 2007, pp. 89-100.

[25] J. Wilander and M. Kamkar, "A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention," in *NDSS*, 2003, pp. 149-162.

[26] J. Gray, *Benchmark handbook: for database and transaction processing systems*: Morgan Kaufmann Publishers Inc., 1992.

[27] P. E. Black and E. Fong, "Proceedings of Defining the State of the Art in Software Security Tools Workshop," *NIST Special Publication,* vol. 500, p. 264, 2005.