A Review of Static Code Analysis Methods for Detecting Security Flaws

Abhayakumara S Basutakara
Dr. Jayanthi P N
Electronics and Communication Dept
R. V. College of Engineering,
Bangalore, India

abhayakumarasb.ec17@rvce.edu.inand jayanthipn@rvce.edu.in

***Abstract**: Static checkers are commonly used by programmers; they verify our programmers for flaws without executing them, a process known as static code analysis. It works with a program that has an early indication of correctness in this way, attempting to avoid well-known traps and problems before comparing it to its specifications. Software security is becoming increasingly crucial in order for programmers to be universally accepted for a wide range of transactions. During the development process, automated code analyzers can be used to detect security flaws. The purpose of this paper is to provide an overview of static code analysis and how it may be used to uncover security flaws. This document summarizes and presents the most recent discoveries and publications. The gains, flows, and methods of static code analyzers are discussed in this study. It can be viewed as a steppingstone toward more research in this area. In Java, there are two types of static code checkers: those that work directly on the source code and those that work on the produced bytecode. Although each code checker is unique, they all share some common characteristics. They read the software and build a model of it, an abstract representation that they may use to match the error patterns they notice. They also perform a data-flow analysis, attempting to deduce the probable values of variables at various stages in the program. Vulnerability testing, an increasingly significant field for code checkers, necessitates data-flow analysis.*

Keywords: Static code analysis,Software security,Vulnerability

1. INTRODUCTION

Millions of people utilize web and desktop applications on a variety of electronic devices to complete their everyday duties. The widespread use of software encourages the software business to consider how to improve quality. Software quality of most the software's are determined by the development team's abilities and knowledge [1]. Unfortunately, developers make mistakes that result in software that is susceptible and defective. Exploited security flaws result in unstable software, which can be harmful to both the user and the program vendor.

There exist a variety of approaches that may be used to reduce the number of bugs in a program. One of them is writing tests, which programmers can perform with the help of tools like JUnit. 2 According to research, code reviews are the most effective technique to eradicate errors. Unfortunately, bringing together the necessary personnel to examine program and identify issue areas takes a long time. To accomplish their jobs well, code review teams need practice (and possibly training). As a result, they can't be used across the whole code base of a project.

The CWE (Common Weakness Enumerations) program establishes a uniform, measured set of software flaws that can result in major security flaws [2]. The top 25 vulnerabilities list includes the most common and severe problems that can lead to security breaches that an attacker can

easily exploit. CWE has identified over 700 software flaws that can result in data modification, read data, resource consumption, denial of service (DOS) assaults, unauthorized code execution, privilege escalation, protection bypass, or activity concealment.

Conveniently, technologies that can scan tons of lines of code in a short amount of time and find faults and vulnerabilities are rapidly developing. A wide range of such utilities are currently available as commercial or free solutions that can be used as stand-alone applications or as plug-ins for development frameworks. The early diagnosis of problems can save a lot of money in the long run. Furthermore, the sooner we detect flaws, the easier it is to correct them. We'd prefer to catch errors as soon as we make them, or as soon as possible later, rather than ipso facto with reviewing or testing.Because people tend to fall into the same traps over and over, the majority of errors fit into well-defined categories. It is precisely the predictability of people's fallibility that allows tools like Lint to exist. Lint searched for well-known mistake patterns. It didn't try to run the software and compare the actual behavior to the predicted behavior, which is what we do when we test. Lint, on the other hand, scoured the programs source for patterns to match. Static checks are a type of tool like this. In a method known as static code analysis, they analyze our programs for faults without running them.

Static checks are typically used after compilation and before testing by programmers. They work with a program that has an initial indication of correctness (because it compiles) and try to avoid well-known traps and pitfalls before testing it against its specifications. Although static checking is pretty painless, it might be humiliating the first time you do it. Lint's success has resulted in a slew of open source and private tools for programmers targeting a variety of languages and operating systems.

2. Static Code Analysis

Static code analysis is a popular tool for helping developers create high-quality software. It allows developers to find potential bugs and security flaws in a program's source code without having to run it.While the potential benefits of static analysis tools are undeniable, their usability is frequently criticized, preventing software engineers from fully utilizing static analysis. Researchers have researched developer needs over the last decade and compared them to the capabilities of available static analysis tools.

The primary design issues for designing accessible static analysis tools are summarized in this study, and they are shown to revolve around the concept of explainability, which is a subarea of usability. Existing analysis tools and current research in static analysis usability are presented, along with a description of how they address those problems. As a result, we propose potential future lines of study in explainability for static analysis, such as transforming static analysis tools into assistants and teachers.

Without executing the application, static analysis tools search the source code for vulnerabilities and examine all possible execution routes and variable values. This makes the procedure quick, repeatable, and independent of whether or not the program can be run. Static code analysis can be used throughout the development phase to assist programmers in identifying and correcting faults and security flaws as early as possible. Static code analysis is linked to source code analysis, which makes it programming language dependent. The fact that automated code analyzing tools tend to report false positives and false negatives is one of the biggest hurdles to their widespread implementation [3]. False positives are findings that report security problems that don't exist, analogous to false negatives, which occur when tools don't report flaws that are already present [4]. Unfortunately, if a significant number of resources is employed for the analysis, false positives and false negatives appear to be unavoidable. Although, theoretically, it is possible to develop a tool that has nearly no false negatives, nearly no false positives, and works for a wide range of realistic programs. By carefully exploring all potential program states with all possible inputs and looking for error states, such a tool would identify errors [3].

Every piece of software has its own business context, environment, and goal. Potential threats and dangers should be characterized in terms of the safeguarded values, and vulnerabilities should be prioritized accordingly. The Common Weakness Risk Analysis Framework (CWRAF) [5] and the Common Weakness Scoring System (CWSS) [6] aid in the community's risk-reduction efforts. CWRAF allows for the automatic selection and prioritization of key vulnerabilities, which can be tailored to the business or mission goals of the company. Developers and customers can benefit from such initiatives by introducing more robust and

resilient software into their operational environments [7].It is easier to detect and resolve security vulnerabilities crucial to the application domain if the business value has been tied to deficiencies. Because manual code reviews are expensive, automated tools appear to be a viable alternative. Static and dynamic code analysis can both be performed by automated tools.Because of the well-known drawbacks of static code analysis, there have been many debates concerning the utility of such techniques. A tool that strikes the correct balance between false positives, false negatives, and performance is the most successful. The value of a static analysis tool can be judged in terms of the flaws that have been identified.

It is simple to calculate the precision measuring once the reports have been triaged and the real positives have been sorted out, but it is difficult to calculate the recall value because it is based on the number of false negatives, which cannot be reliably estimated. Also, just because a tool is good at discovering one type of vulnerability doesn't indicate it will have a good recall and precision value for other sorts of vulnerabilities. The real test of a static-analysis tool's utility, according to most people, is whether it can detect a sufficient number of relevant problems in a program without overwhelming the user with worthless data or consuming an excessive amount of computational resources.

A well-structured and straightforward user interface that organizes related problems is a step forward in improving user experience. This may limit the number of reports received and suppresses repetitive defect reports that can be resolved with a single solution. Some tools, such as Parfait, make it simple to store and share flaws. Other team members can review these reports, and they can be used to predict bugs in the future. Because of how they were presented, certain results may be misread or disregarded. It takes time, effort, and experience to distinguish a true positive from a false positive. Tools must be intuitive, simple to install, and transparent in their presentation of results, or they will not be used, regardless of how exact or fast they are.

3. Types of Security Attacks

Security breaches can occur in a variety of ways. The following are a few of the most important types.:

3.1 Cross-Site Scripting (*XSS*)

Cross-site scripting is a type of attack that allows attackers to inject client-side scripts into Web sites that are being viewed by other users. It is used by attackers to get around access barriers, steal user sessions, and reroute users to another website. By executing scripts in the browser, threat actors can employ XSS flaws to hijack user sessions, deface websites, or lead users to malicious websites. An XSS issue happens when an application places untrusted data in a new webpage without proper validation or escape. If an application uses an HTML or JavaScript-creating browser API to update an existing webpage with user-supplied data, this could cause problems.

3.2 SQL Injection

SQL – Injection is a severe hazard to data-intensive software. An attacker can inject SQL instructions into any non-validated textbox that takes user input and uses it in a SQL context. The commands are used by attackers to circumvent authentication systems or put corrupted material into databases.

3.3 Buffer Overflow

Buffer overflows occur when a program overruns the buffer's boundary and overwrites neighboring memory while writing data to it. Non-validated inputs that are intended to execute code can set it off. Buffer overflow can cause unpredictable program behaviour, such as memory access issues, inaccurate results, a crash, or a security compromise.

3.4 OS Command Injection

Applications are vulnerable to an OS command injection attack if they use unvalidated user input in a system level command, which could result in the execution of attacker-injected scripts. In order to achieve the intended result, attackers attempt to execute system level commands.

3.5 Missing Authentication

Lacking authentication is a security flaw that happens when software does not conduct any authentication for functions that require a proved user identity or that use a lot of resources. Critical functionality is being exposed. Gives an attacker access to the functionality's mount point.

4. Discussion: Working of Static code Analysis

A representation of the program to be *analyzed* is used to do static code analysis. Symbol tables, control-flow graphs, abstract syntax trees, and the program call graph are common intermediate representations in the model.This model is queried using a variety of processes, ranging from simple data structure searches to highly sophisticated algorithms based on advanced ideas like data flow analysis, symbolic execution, abstract interpretation, and model validation *[4]*.Since the *analyzing* methods are applied to a model of the programrather than the program itself, false positives and negatives might occur because the model is almost never exact. The program's model is frequently either an over- or under-estimate. Because it captures all potential features of the program's behaviour, an accurate model or over-approximation is generally referred to as sound. These models are likely to detect the majority of real positives, but they also have a tendency to report falsepositives. A model that fails to capture any significant component of the program is called an under-approximation, and it can lead to a variety of false negatives.

It is vital to detect mistakes in the software from the beginning of development to the stage of production in order to avoid the flaws or vulnerabilities indicated. Code review, static code analysis, unit tests, vulnerability checks, and other methods are among them.

When compared to other ways of identifying faults in programs, static code analysis has the advantage of being able to detect vulnerabilities without having to run the program. Only by scanning the program's source code using numerous ways can a vulnerability be discovered.

To detect security flaws in a program model, several methodologies and approaches have been created. The most prevalent vulnerabilities are shown in Table I, along with their classifications depending on the source of the security vulnerability. The approaches and methods that static code analysis is based on for the tree discovered vulnerability classes are detailed in the subsections below.

Table I. Classification of Vulnerabilities [8]

Information Flow Vulnerabilities	**Access Control Vulnerabilities**	***API Conformance Vulnerabilities***
Un-validated input	Broken access control	Insecure storage
Cross-site scripting	Broken authentication and session management	
Injection flaws	Application denial service	

4.1 Information Flow

Integrity and confidentiality are the two basic forms of information flow risks. The noninterference concept states that no information should pass from higher to lower security layers [9]. Disregard for this principle can result in the confidentiality and integrity of sensitive data being jeopardized. Integrity breaches are vulnerabilities such as those caused by non-validated input and injection bugs. SQL injections and OS command injections are two of the most frequent attacks that result from the usage of tainted data (material obtained from an untrusted source) in a trusted environment [10]. A value flowing out of a privilege-asserting block should remain limited inside that component unless a check has been conducted to validate that the value can be properly released [8]. The main concept behind utilizing static analysis to detect information flow is to check that the flow of information between variables in a program is consistent with the security labelling of variables statically. Each variable has a security level assigned to it. There is potential information flow from x to y if a variable x is utilized to determine or affect the value of another variable y [8].

Only if the security policy enables the security level of x to flow to the security level of y is the flow legal. The most effective methodologies and methods for detecting information flow vulnerabilities are presented in studies.

4.2 API Conformance

Violations of API requirements might lead to security issues. For system security services, business software platforms provide a number of APIs. Misuse of these APIs can result in security flaws. Compilers cannot identify this type of abuse, but it can be discovered using simple syntactic code scanning or more advanced methods. A client that wishes to use a package's services must follow the API compliance standards provided in the class documentation.

The application will get an access control exception if the client fails to fulfil certain conformance constraints. API conformance rules can be stated as type state specifications or temporal safety features in a deterministic finite state automaton [14]. A considerable number of studies [15]focus on the application of such principles, while others [16]take a purely analytical approach.

4.3 Access Control

Access control measures based on a user's identity and membership in specified groups restrict access to security-sensitive resources [11]. When an access control policy does not offer a user sufficient permission, runtime authorization errors can occur. Similarly, if an access control policy gives people too many permissions, the system may be vulnerable to security threats. In the Java and.Net Common Language Runtime platforms, there are two major approaches.

4.3.1 Net Common Language Runtime Platforms

- Stack-based Access Control (SBAC)
- Role-based Access Control (RBAC)

Only programs that meet a set of clearance requirements have access to restricted resources, according to SBAC systems. When a program tries to access restricted resources, the runtime system checks to see if all callers on the thread's stack meet a set of permissions. As a result, SBAC systems prevent a caller from executing more trusted code in order to obtain access to limited resources. Permissions can be granted declaratively in a policy database in SBAC systems. Permissions for an SBAC security policy are determined by dynamic program activity. Because numerous components interact in various settings, this can be fairly difficult. Testing can be used to determine the policy configuration. Iteratively add permissions until no problems occur. Even if all tests pass, there's still a risk that some execution pathways were missed, which could result in runtime authorization issues.

RBAC is a system access control method that allows only authorized users to access the system. Permissions and roles are used to control access in RBAC systems. Permissions denote the ability to carry out specific tasks. A collection of permissions that can be assigned to a user

is represented by a role. Each user attempting to perform an operation must be assigned to a role that has the appropriate permissions.

RBAC is an access control method that is based on operations rather than resources, which has its own set of difficulties. To limit access to sensitive data, all operations that access such data must be identified. In this case, a resource-based strategy would be far more intuitive. Furthermore, there is no straightforward mechanism to prevent a person from accessing sensitive information. For the role assigned to the user, any operations that have direct or indirect access to the resource must be denied. The intended data-based security policy would not conflict with the preset operation-based security policy [12] [13] if the operations were mapped to the data they access.

5. Conclusion

Static code analysis can be used to find flaws in a program. Source code contains security flaws. It's a quick and simple process. a method for scanning millions of lines of code that is repetitive at the stage of development This saves both time and money. a method of detecting vulnerabilities before an attacker has a chance to exploit them to take use of them False positives and false negatives, as shown in the paper, Negatives continue to be a significant impediment to acceptance and understanding. Such tools are widely used. The third section explains what this means. Static analysis is based on sophisticated algorithms. Constantly developing. This implies that static analysis is one of them. one of the most promising ways for combating Ensure that your software is secure and trustworthy.

Static code analysis tools are an extraordinarily effective approach to locate and highlight programming flaws to software programmers. It allows problems to be detected long before they cause mayhem when the code is released or deployed to a server. Static analysis is typically thought to be the more complete method of code analysis. It may also prove to be the more cost-effective solution. When code mistakes are detected early on, they are usually less expensive to rectify than when they become stuck in the system.

REFERENCES

[1] Etics, “Software Quality Basics”, http://etics.web.cern.ch/etics/sqbasics.htm [Online; accessed 08/23/13]

[2] Steve Christey,2011 CWE/SANS Top 25 Most Dangerous SoftwareErrors. The Mitre Corporation, http://cwe.mitre.org/top25/, 2011 [Online; accessed 3.7.2013.]

[3] Walden, J.; Doyle, M., "SAVI: Static-Analysis Vulnerability Indicator,Security & Privacy, IEEE, vol.10, no.3, pp.32,39, May-June 2012

[4] Anderson, P., "Measuring the Value of Static-Analysis ToolDeployments," Security & Privacy, IEEE, vol.10, no.3, pp.40,47, May-June 2012.

[5] Common Weakness Risk Analysis Framework[http://cwe.mitre.org/cwraf/ [Online; accessed 09/01/13]

[6] Common Weakness Scoring System, http://cwe.mitre.org/cwss/ [Online; accessed 09/01/13]

[7] Martin, R.A.; Christey,S.M., "The Software Industry's "Clean WaterAct" Alternative," Security &Privacy,IEEE, vol.10, no.3, pp.24,31,May-June 2012.

[8] Pistoia, M.; Chandra, S.; Fink, S.J.; Yahav, E., "A survey of staticanalysis methods for identifying security vulnerabilities in softwaresystems," IBM Systems Journal , vol.46, no.2, pp.265,288, 2007.

[9] J. Newsome and D. Song, „ Dynamic taint analysis for automatic detection, analysis, signature generation of exploits on commodity software“, Proceedings of the 12th Annual Network and Distributed System Security Symposium, San Diego, CA, USA,IEEE Computer Society (February 2005).

[10] D. Volpano, C. Irvine, and G. Smith, "A Sound Type System for Secure Flow Analysis, "Journal of Computer Security 4, Nos. 2-3, 167–187(January 1996).

[11] J. H. Saltzer and M. D. Schroeder, “The protection of information incomputer systems”, Proceedings of the IEEE 63, No. 9, 1278–1308 (September 1975).

[12] Paolina Centonze,GlebNaumovich, Stephen J. Fink, Marco Pistoia,“Role based access control consistency validation”, ISSTA’06, July 17-20, 2006, Portland, Maine, USA.

[13] A. Schaad and J. D. Moffett, "A Lightweight Approach to Specification and Analysis of Role-Based Access Control Extensions," Proceedings of the 7th ACM Symposium on Access Control Models and Technologies, Monterey, CA, USA, ACM Press (2002), pp. 13–22.

[14] J. A. Goguen and J. Meseguer, "Security Policies and Security Models, “Proceedings of the 1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, IEEE Computer Society Press (May 1982), pp. 11-20.

[15] R. E. Strom and S. Yemini, "Type state: A Programming Language Concept for Enhancing Software Reliability", IEEE Transactions on Software Engineering 12, No. 1, pp. 157–171 (1986).
[16] J. C. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, Robby, S. Laubach, and H. Zheng, "Bandera: Extracting Finite- State Models from Java Source Code," Proceedings of International Conference on Software Engineering (June 2000), pp. 439–448.