A Tool for Detection of Software Vulnerabilities in Java Programs

Srinivas.S.P[1], Sowmya K.B[2], Nirja Parida[3], and Anirban Basu[4]
Department Of CSE,
East Point College of Engineering & Technology
Bangalore, Karnataka, India
[1]srinidy@gmail.com,
[2]niraja.epcet31@gmail.com,[3]kbsowmya10@gmail.com,
[4]abasu@anirbanbasu.in

*Project funded by the Government of Karnataka under VGST Scheme (2013-2014)

***Abstract-* A software vulnerability is a flaw or defect in the software construction that can be exploited to hack the code.It means the vulnerability offers a possible entry point to the system.Vulnerability is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, and attacker capability to exploit the flaw so in general it is said to be as the attack surface. To make the software secure, vulnerabilities must be identified and corrected during code inspection. Identifying weaknesses manually in large programs is time consuming and therefore the process of identification needs to be automated. This paper discusses a tool called SecCheck developed at EPCET to identify vulnerabilities in Java code. The tool takes Java source files as input, stores each line in memory and scans to find vulnerabilities. A warning message is displayed when vulnerability is found. The tool can detect some common software vulnerabilities.**

Keywords: Vulnerability, Reachable Assertion, Throttling, External Control of Critical State Data, Cryptographic Signature, Integrity check, Weak cryptography, Neutralization of CRLF, Use of Non-Canonical URL Paths, Reliance on Cookies.

I. INTRODUCTION

The security of software is threatened at various points throughout its lifecycle, both by inadvertent and intentional choices and actions taken by insiders. Software security deals with protecting software against malicious attack and other risks so that the software continues to function correctly under such potential risk and threats. Security aspects have to be taken care during design and implementation as unintentional mistakes during coding by the programmer may make the software vulnerable.

Poor software design and engineering are the root causes of most security vulnerabilities in systems. Efforts are required for protecting software against malicious attacks so that the software continues to function correctly under potential threats. Security threats are also obtained from web sites and web applications (webapps) that may come in many forms. Data centres and other assets used for hosting web sites and their associated systems need to be protected from all types of threat. Threats should be identified using application threat modelling and then evaluated with a vulnerability assessment.

Nowadays a lot of attention is being put to build secure software and remove any vulnerability in source code which can be exploited to hack the code. Many types of vulnerabilities exist in software systems injected in design phase and in the implementation phase, such as local implementation errors, inter procedural interface errors (such as a race condition between an access control check and a file operation), design-level mistakes (such as error handling and recovery systems that fail in an insecure fashion, topology of the web of links, deep linking, unspecified image dimensions), and object-sharing systems.

This paper discusses vulnerabilities that are injected in Java programs during coding phase, making it prone to hacking and describes the development of a tool that detects the vulnerabilities and gives messages alerting developer to correct these. The tool developed by the authors and named SecCheck detects vulnerabilities in any Java program caused by Reachable Assertion, Throttling, External Control of Critical State Data, Cryptographic Signature, Integrity check, Weak cryptography, Neutralization of CRLF, Use of Non-Canonical URL Paths, Reliance on Cookies.

Section 2 discusses about the common weaknesses along with their consequences occurring in the software.section 3 discusses about existing static code analysis tools with their features.Section 4 discusses about the working of tool SecCheck.

II. Common Vulnerabilities in Software

Common Software Vulnerabilities that occur in Java programs as discussed in CWE [1] are:

1. Reachable Assertion
2. Throttling
3. External Control of Critical State Data
4. Improper Verification of Cryptographic signature
5. Weak cryptography for password
6. Missing Support for Integrity Check
7. Improper Neutralization of CRLF Sequences in HTTP Headers
8. Use of Non-Canonical URL Paths for Authorization Decisions
9. Reliance on Cookies without Validation and Integrity Checking in a Security Decision

These vulnerabilities are detected by the SecCheck tool in any Java program and a warning message along with line number is displayed on detection.

These vulnerabilities are briefly discussed below along with the consequences.

A. Reachable Assertion

Reachable assertion occurs when assert is triggered by an attacker causing crash of application or cause a denial of service.

While assertion is good for catching logic errors and reducing the chances of reaching more serious vulnerability conditions

Consequences

1. Chat client allows remote attackers to cause a denial of service (crash) via a long message string when connecting to a server, which causes an assertion failure.
2. Product allows remote attackers to cause a denial of service (crash) via certain queries, which cause an assertion failure.

B. Throttling

The software allocates a reusable resource or group of resources on behalf of an actor without imposing any restrictions on how many resources can be allocated.
When allocating resources without limits, an attacker could prevent other systems, applications, or processes from accessing the same type of resources.

Consequences

1. Driver Large integer value for a length property in an object causes a large amount of memory allocation.
2. CMS does not restrict the number of searches that can occur simultaneously, leading to resource exhaustion.
3. Product allows attackers to cause a denial of service via a large number of directives, each of which opens a separate window.

C. External control of critical state data

External control of critical state data occurs when attacker access security-critical state information causes access unexpected resources

The state variables may contain sensitive information that should not be known by the client

By modifying state variables, the attacker could violate the application's expectations for the contents of the state, leading to a denial of service due to an unexpected error condition.

Consequences

1. Mail client stores password hashes for unrelated accounts in a hidden form field.
2. Application allows admin privileges by setting a cookie value to "admin."
3. Setting of a language preference in a cookie enables path traversal attack.

D. Verification of cryptography signature

The software does not verify, or incorrectly verifies, the cryptographic signature for data.

Digital signature can be used with any kind of message, whether it is encrypted or not, so that receiver can be sure of the sender's identity and that the message arrived intact.

A valid digital signature gives a recipient reason to believe that the message was created by a known sender and that the message was not altered in transit.

Consequences

1. An attacker could gain access to sensitive data.
2. Attacker can modify application data.
3. Although messages may often include information about the entity sending a message, that information may not be accurate. Digital signatures can be used to authenticate the source of messages.

4. If a message is digitally signed, any change in the message after signature invalidates the signature.

E. Weak cryptography for password

Cryptography enables you to store sensitive information or transmit it across insecure networks so that it cannot by anyone except the intended recipient.

Obscuring a password with a trivial encoding does not protect the password.

Consequences

1. When a password is stored in plaintext in an application's properties or configuration file, anyone with access to configuration file can easily determine password .

2. If a devious employee has access to this information, they can use it to break into the system.

F. Missing support for integrity check

Checksumming is a well known method for performing integrity checks.

If the computed checksum for the current data input matches the stored value of a previously computed checksum, there is a very high probability the data has not been accidentally altered or corrupted.

Consequences

1. Data that is parsed and used may be corrupted.
2. Without a checksum it is impossible to determine if any changes have been made to the data after it was sent.
3. Attackers can gain access to the sensitive information and can alter the data.

G. Improper neutralization of CRLF sequences in http headers

CRLF Injection is a software application coding vulnerability that occurs when an attacker injects a CRLF character sequence where it is not expected.

Exploits occur when an attacker is able to inject a CRLF sequence into an HTTP stream.

CRLF Injection vulnerabilities result from data input that is not neutralized, incorrectly neutralized, or otherwise unsanitized.

Consequences

1. CRLF Injection exploits security vulnerabilities at the application layer.
2. Attackers can modify application data compromising integrity.
3. Enables the exploitation of the following vulnerabilities:
   - XSS or Cross Site Scripting vulnerabilities
   - Proxy and web server cache poisoning
4. CR and LF characters in an HTTP header may give attackers control of the remaining headers and body of the response entirely under their control.

H. Use of non-canonical URL paths for authorization decisions

The software defines policy namespaces and makes authorization decisions based on the assumption that a URL is canonical.

This can allow a non-canonical URL to bypass the authorization.

Even if an application defines policy namespaces and makes authorization decisions based on the URL, but it does not convert to a canonical URL before making the authorization decision, then it opens the application to attack.

Consequences

1. If a non-canonical URL is used, the server chooses to return the contents of the file, instead of pre-processing the file.
2. An attacker can bypass the authorization mechanism to gain access to the otherwise-protected URL.

I. Reliance on cookies without validation and integrity checking in a security decision

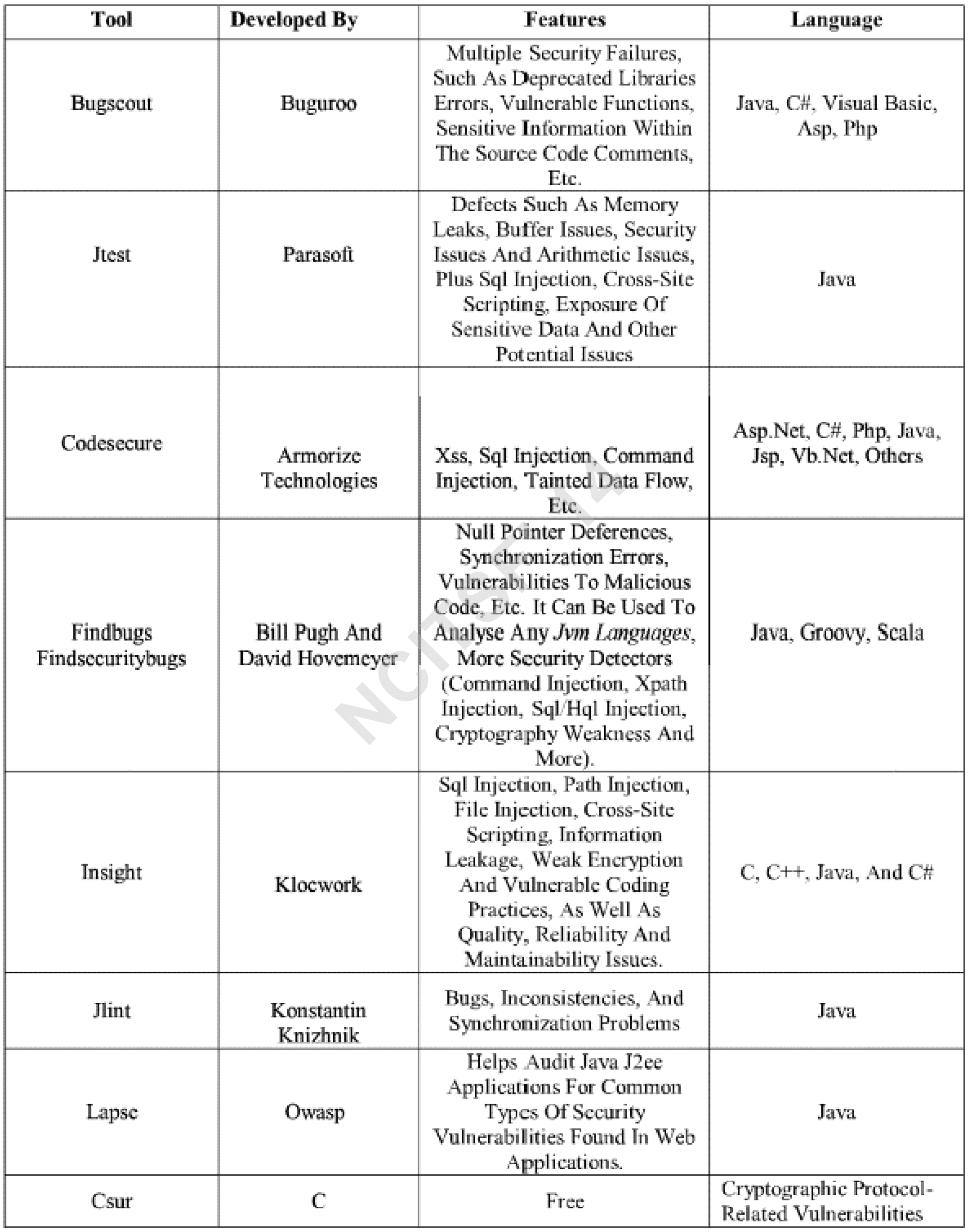
The application uses a protection mechanism that relies on the existence or values of a cookie, but it does not properly ensure that the cookie is valid for the associated user.

Attackers can bypass protection mechanisms such as authorization and authentication by modifying the cookie to contain an expected value.

Consequences

1. The cookie can be manipulated to claim a high level of authorization, or to claim that successful authentication has occurred

III. EXISTING STATIC CODE ANALYSIS TOOLS WITH THEIR FEATURES:



Tool	Developed By	Features	Language
Bugscout	Buguroo	Multiple Security Failures, Such As Deprecated Libraries Errors, Vulnerable Functions, Sensitive Information Within The Source Code Comments, Etc.	Java, C#, Visual Basic, Asp, Php
Jtest	Parasoft	Defects Such As Memory Leaks, Buffer Issues, Security Issues And Arithmetic Issues, Plus Sql Injection, Cross-Site Scripting, Exposure Of Sensitive Data And Other Potential Issues	Java
Codesecure	Armorize Technologies	Xss, Sql Injection, Command Injection, Tainted Data Flow, Etc.	Asp.Net, C#, Php, Java, Jsp, Vb.Net, Others
Findbugs Findsecuritybugs	Bill Pugh And David Hovemeyer	Null Pointer Deferences, Synchronization Errors, Vulnerabilities To Malicious Code, Etc. It Can Be Used To Analyse Any *Jvm Languages*, More Security Detectors (Command Injection, Xpath Injection, Sql/Hql Injection, Cryptography Weakness And More).	Java, Groovy, Scala
Insight	Klocwork	Sql Injection, Path Injection, File Injection, Cross-Site Scripting, Information Leakage, Weak Encryption And Vulnerable Coding Practices, As Well As Quality, Reliability And Maintainability Issues.	C, C++, Java, And C#
Jlint	Konstantin Knizhnik	Bugs, Inconsistencies, And Synchronization Problems	Java
Lapse	Owasp	Helps Audit Java J2ee Applications For Common Types Of Security Vulnerabilities Found In Web Applications.	Java
Csur	C	Free	Cryptographic Protocol-Related Vulnerabilities

IV. WORKING OF SECCHECK

SecCheck tool takes as input any Java program and scans to identify the vulnerabilities. If any vulnerability is detected then it displays warning message. The steps followed are

1. Select the input Java program
2. Select from the drop down list all types of vulnerabilities

The tool displays type of vulnerabilities and the place of occurrence.

CONCLUSION

Vulnerability is a weakness in software. The cause of such a "weakness" can be faults in design and code. It also allows an attacker to reduce a system's information assurance.The presence of vulnerabilities in the software makes it necessary to have tools that can help programmers to avoid or detect them in the development of the code.

There are many tools available to detect the vulnerabilities present in application programs written for various programming languages but no single tool has the capability to detect so many vulnerabilities.

The tool developed by the authors and described in this paper detects nine vulnerabilities in Java source code.

REFERENCES

[1] http://www.cwe.mitre.org

[2] Reachable Assertion: "http://cwe.mitre.org/data/definitions/617.html".

[3] Allocation of resources without limits or throttling: "http://cwe.mitre.org/data/definitions/770.html".

[4] External control of Critical state data: "http://cwe.mitre.org/data/definitions/642.html".

[5] Improper verification of cryptographic signature: "http://cwe.mitre.org/data/definitions/347.html".

[6] Weak cryptography for passwords: "http://cwe.mitre.org/data/definitions/261.html".

[7] Missing support for integrity check: "http://cwe.mitre.org/data/definitions/353.html".

[8] Improper neutralisation of CRLF sequences in HTTP headers: "http://cwe.mitre.org/data/definitions/113.html".

[9] Use of Non-Canonical URL Paths for Authorization Decisions: "http://cwe.mitre.org/data/definitions/647.html".

[10] Reliance on Cookies without Validation and Integrity Checking in a Security Decision: "http://cwe.mitre.org/data/definitions/784.html".