'National Conference on Recent Advances of Computational Intelligence Techniques in Science, Engineering and Technology'
International Journal of Scientific Research in Computer Science, Engineering and Information Technology | ISSN : 2456-3307 (www.ijsrcseit.com)

A Detection Tool for Finding Software Vulnerabilities in JAVA Code

Dr. Chandramouli H[1], Dr. I Manimozhi[2], Kesavan M V[3]
[1]Professor, [2]Associate Professor, [3]Assistant Professor
Department of CSE, East Point College of Engineering and Technology, Bengaluru, Karnataka, India

ABSTRACT

A vulnerability is a weakness in software that enables an attacker to compromise the integrity, availability, and confidentiality of the program and data that it processes. Software applications or programs are implemented in different languages and most of them contain serious vulnerabilities which can be exploited to cause security breaches. Many security vulnerabilities may be present in programs [1] and a number of techniques have been developed to detect these [2].

However, it is essential that these vulnerabilities are not only detected but corrected. Vulnerability management is the cyclical practice of identifying, classifying, remediating, and mitigating vulnerabilities [3]. The paper discusses a tool developed by the authors which not only detects software vulnerabilities but provides solutions for correcting them.

The tool also calculates the Degree of Insecurity in a Java program first defined in [2]. It applies the proposed mitigating methods and recalculates the Degree of Insecurity. Experimental results as discussed in the paper indicates that the mitigation methods suggested in the tool are very effective.

Keywords: Vulnerability, CWE, SecCheck, Mitigation, Degree of Insecurity (ISM).

I. INTRODUCTION

It is essential that software is engineered so that it continues to function correctly under malicious attacks. Software security is the process of designing, building, and testing software for security: it needs to identify and mitigate weaknesses so that software can withstand attacks. Many recent security breaches have been found to be due to inadequate design and improper coding. The vulnerabilities have unwanted consequences such as hijacking of session information, deletion or alteration of sensitive data, and execution of arbitrary code supplied by hackers [1].

It is essential that software vulnerabilities are not only detected but techniques used for correction of such vulnerabilities in order to prevent attacks and minimize operational disruption due to these. While Vulnerability Management [3][4][5] is being discussed by researchers, there are very few tools which actually does this.

The tool SecCheck developed by the authors can detect a number of vulnerabilities and has been described in [2] while the tool discussed here not only detects vulnerabilities in Java programs but also guides the developers by offering possible solutions for mitigating these. The tool detects vulnerabilities in any Java program caused by Null Point Dereference, Reachable Assertion, Allocation Of Resources Without Limits Or Throttling, Improper Validation of Integrity Check Value, Serializable Class Containing Sensitive Data, cleartext transmission of sensitive information, Improper Validation of Certificate With Host Mismatch, Improper Encoding or Escaping of Output, Improper Neutralization of CRLF Sequences in HTTP Headers, Use of Non-Canonical URL Paths for Authorization Decisions. It also offers solutions for mitigating these.
In Section 2 the pros and cons of the presence of weaknesses are discussed along with ways of mitigating these. Section 3 discusses Degree of Insecurity in a code. Section 4 discusses about the working of the tool while Section 5 describes results of experiments conducted to analyse the effectiveness of the tool.

II. COMMON VULNERABILITIES IN JAVA CODE

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

1. Reachable Assertion
2. Throttling
3. Null Pointer Dereference
4. Use of a One-Way Hash without a Salt
5. Use of Insufficiently Random Values
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
10. Authentication Bypass by Spoofing
11. Unrestricted Upload of File with Dangerous Type
12. Improper Encoding or Escaping of Output

These vulnerabilities are detected in any Java program by the tool developed by the authours and solutions for mitigating these are offered. These are briefly discussed below along with the consequences and mitigations .

2.1. REACHABLE ASSERTION

Reachable assertion occurs when assert is triggered by an attacker causing crash of application or cause a denial of service[6].
While assertion is good for catching logic errors and reducing the chances of reaching more serious vulnerability conditions

Pros and Cons

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

2.2. 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 [7].

Pros and Cons

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

2.3. NULL POINTER DEREFERENCE

Null pointer dereferencing occurs when a variable bound to the null value is treated as if it were a valid object reference and used without checking its state. This condition results in a NullPointerException, which can in turn result in a denial of service [8].

Pros and Cons

1. NULL pointer dereferences usually result in the failure of the process unless exception handling (on some platforms) is available and implemented. Even when exception handling is being used, it can still be very difficult to return the software to a safe state of operation.
2. In very rare circumstances and environments, code execution is possible.

2.4. USE OF A ONE-WAY HASH WITHOUT A SALT

The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software does not also use a salt as part of the
input [9].

In cryptography, a **salt** is random data that is used as an additional input to a one-way function that hashes a password or passphrase.

The primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.

Pros and Cons

1. If an attacker can gain access to the hashes, then the lack of a salt makes it easier to conduct brute force attacks.
2. While it is good to avoid storing a cleartext password, the program does not provide a salt to the hashing function, thus increasing the chances of an attacker being able to reverse the hash and discover the original password if the database is compromised.

2.5. USE OF INSUFFICIENTLY RANDOM VALUES

The software may use insufficiently random numbers or values in a security context that depends on unpredictable numbers.

Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed [10]. A random number generator (RNG) is a computational or physical device designed to generate a sequence of numbers or symbols that lack any pattern, i.e. appear random

Pros and Cons

1. When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated.
2. And use this guess to impersonate another user or access sensitive informat ion

2.6. MISSING SUPPORT FOR INTEGRITY CHECK

Checksumming is a well known method for performing integrity checks [11]. 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.

Pros and Cons

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

2.7. 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 [12].

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

Pros and Cons

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

2.8. 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 [13].

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.

Pros and Cons

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 UR

2.9. 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 [14].

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

Pros and Cons

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

2.10. AUTHENTICATION BYPASS BY SPOOFING

This attack-focused weakness is caused by improperly implemented authentication schemes that are subjected to spoofing attacks.

An authentication mechanism implemented in java relies on an IP address for source validation. If an attacker is able to spoof the IP, however he may be able to bypass such authentication mechanism [15].

Pros and Cons

1. This weakness can allow an attacker to access resources which are not otherwise accessible without proper authentication

2.11. UNRESTRICTED UPLOAD OF FILE WITH DANGEROUS TYPE

The software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment [16].

Consequences

1. Arbitrary code execution is possible if an uploaded file is interpreted and executed as code by the recipient
2. The lack of restrictions on the size or number of uploaded files, which is a consumption issue

2.12. IMPROPER ENCODING OR ESCAPING OF OUTPUT

The software prepares a structured message for communication with another component, but encoding or escaping of the data is either missing or done incorrectly. As a result, the intended structured of the message is not preserved

If an application uses attacker-supplied inputs to construct a structured message without properly encoding or escaping, then the attacker could insert special characters that will cause the data to be interpreted as control information or metadata [17]. The component that receives the output will perform the wrong operations, or otherwise interpret the data incorrectly

Pros and Cons

1. The communications between components can be modified in unexpected ways
2. Unexpected commands can be executed, bypassing other security mechanisms. Incoming data can be misinterpreted

III. DEGREE OF INSECURITY IN JAVA CODE

Each of the weaknesses discussed in this paper has been assigned a severity level defined in CWE as shown in Table 1. We use a metric for calculating the Degree of Insecurity (referred to as ISM) [2].

$ISM = \sum_{i=1}^{m} Wi * N i$

where,

ISM stands for the Degree of Insecurity,

i is the Type of Vulnerability 1,2,....m

W_i is the Severity of Vulnerability in the software

Ni is the frequency of occurrence of vulnerability i.

Table 1: Severity of Vulnerabilities

Type of Vulnerability = i	Severity = Wi
Reachable Assertion	5
Throttling	4
Null Pointer Dereference	19
Use of a One-Way Hash without a Salt	12
Use of InsufficientlyRandom Values	15
Missing Support for Integrity Check	1
Improper Neutralization of CRLF Sequences in HTTP Headers	8
Use of Non-Canonical URL Paths for Authorization Decisions	1
Reliance on Cookies without Validation and Integrity Checking in a Security Decision	5
Authentication Bypass by Spoofing.	1
Unrestricted Upload of File with Dangerous Type	10
Improper Encoding or Escaping of Output.	6

IV. WORKING OF THE TOOL

The tool takes as input any Java program and scans to identify the vulnerabilities. If any vulnerability is detected then it displays warning message and suggests steps for its mitigation.

The steps followed are :

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

As shown in Figure 1, for a Java program given as an input to the Tool, it displays type of vulnerability found and the place of its occurrence. It also gives the Degree of Insecurity in the input program

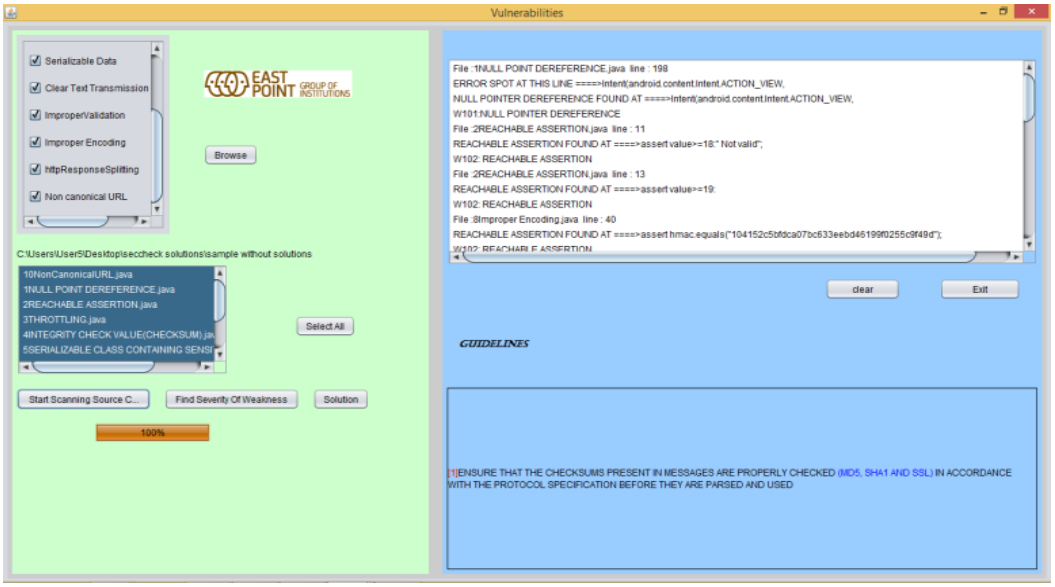

Figure 1: Front end of the Tool

The functional modules in the Tool are shown in Figure 2.*Scanner:* This module scans each line of source code one by one.

Pattern Matching Module: After scanning, the tool compares each line to find out if it contains a set of keywords which makes the program vulnerable to security threats. This is done by matching each line with the list of strings stored in a database.

Display Module: If there is a string match then a warning message is flagged to the user.

After the entire program is scanned, the *Degree of Insecurity* is calculated and displayed.

Mitigation Module: After the program is scanned, and Degree of Insecurity is found out, the next step is to mitigate the vulnerabilities detected. The tool provides solution for each of the vulnerabilities. The user has to incorpore the suggestions, so that the software becomes more secure.

Degreee of Insecurity is calculated again after this. If the Degree of Insecurity is acceptable, then the iteration stops. Otherwise, the steps are repeated and 'delinquent' statements are replaced.

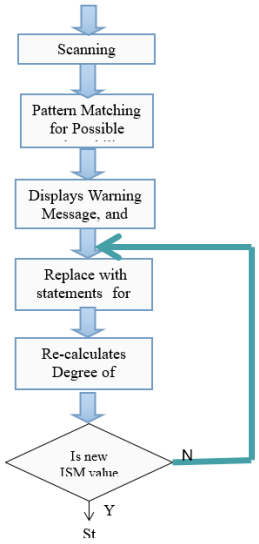

Figure 2: Working Scenario of the Tool

V. EXPERIMENTAL OUTCOMES

For detection of vulnerabilities by the tool we used programs written by the professionals taken from different vulnerability tracking sites and some were taken from Common Weakness Enumeration (CWE) site.
The results of measurements are given in Table 2. The vulnerabilities detected and the Degree of Insecurity Before Mitigation (BM) and After Mitigation (AM) are shown in these programs calculated as per the expression in Section 4 of this paper.

Table 2: Degree of Insecurity calculated in JAVA Code

Example name	Source	ISM (BM)	ISM (AM)
assert	http://www.javapractices.com/topic/TopicAction.do?Id=102	80	0
serverSocket	http://stackoverflow.com/questions/15541804/creating-the-serversocket-in-a-separate-thread	42	0
contentIntent	http://www.programcreek.com/java-api-examples/index.php?api=android.content.Intent.ShortcutIconResource (example 2)	95	0
hashmd5	http://howtodoinjava.com/2013/07/22/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples	60	0
httpurlconnection	http://www.mkyong.com/java/java-httpurlconnection-follow-redirect-example	60	0
UseDatagram	http://www.java2s.com/Code/Java/Network-Protocol/UseDatagramSockettosendoutandreceiveDatagramPacket.htm	22	0
Neutralization	http://stringpool.com/servlet-sendredirect-example	32	0
Non-Canonical URL Paths	https://code.google.com/p/crawler4j/source/browse/src/main/java/edu/uci/ics/crawler4j/url/URLCanonicalizer.java?r=b5b88a4d5c649a03e522b4e0557e7bbca1cc737b	48	0
Cookies	http://javabynataraj.blogspot.in/2011/04/what-is-deserialization-in-java-write.html	52	0
Authentication Bypass by Spoofing.	http://www.codereye.com/2010/01/get-real-ip-from-request-in-java.html	40	0
Unrestricted Upload	http://www.javacodegeeks.com/2013/08/servlet-upload-file-and-download-file-example.html	330	0

of File with Dangerous Type			
Improper Encoding or Escaping of Output.	http://www.programcreek.com/java-api-examples/index.php?api=javax.crypto.spec.SecretKeySpec (example 27)	18	0
Average Value of ISM Calculated from these Programs		**73.25**	**0**

VI. CONCLUSION

Vulnerability is a weakness in software. The causes of such "weakness" can be faults in design and in code and 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 detect and correct them during development of the code.

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

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

VII. REFERENCES

[1]. http://www.cwe.mitre.org
[2]. Priyadarshini. R, Nivedita Ghosh and Anirban Basu "SecCheck: A Tool for Detection of Vulnerabilities and for Measuring Insecurity in Java Programs": International Journal of Software Engineering, Vol.7, No.2, July 2014, pp.67-93
[3]. Park Foreman, Vulnerability Management, Auerbach Publications, 2009
[4]. http://download.microsoft.com/download/5/0/5/505646ED-5EDF-4E23-8E84 6119E4BF82E0/Mitigating_ Software_ Vulnerabilities.pdf
[5]. A Agrawal and R A Khan, A Framework to Detect and Analyze Software Vulnerabilities -Development Phase Perspective", International Journal of Recent Trends in Engineering, Vol 2, No. 2, November 2009, pp 82-84
[6]. Reachable Assertion: "http://cwe.mitre.org/data/definitions/617.html"
[7]. Allocation of resources without limits or throttling: "http://cwe.mitre.org/data/definitions/770.html"
[8]. Null Pointer Dereference : "http://cwe.mitre.org/data/definitions/476.html"
[9]. use of a one-way hash without a salt: http://cwe.mitre.org/data/definitions/759.html
[10]. use of insufficiently random values: "http://cwe.mitre.org/data/definitions/330.html"

[11]. Missing support for integrity check: “http://cwe.mitre.org/data/definitions/353.html”
[12]. Improper neutralisation of CRLF sequences in HTTP headers:“http://cwe.mitre.org/data/definitions/113.html”
[13]. Use of Non-Canonical URL Paths for Authorization Decisions: “http://cwe.mitre.org/data/definitions/647.html”
[14]. Reliance on Cookies without Validation and Integrity Checking in a Security Decision: “http://cwe.mitre.org/data/definitions/784.html”
[15]. Authentication Bypass by Spoofing. http://cwe.mitre.org/data/definitions/290.html
[16]. Unrestricted Upload of File with Dangerous Type. http://cwe.mitre.org/data/definitions/434.html
[17]. Improper Encoding or Escaping of Output. http://cwe.mitre.org/data/definitions/116.html