ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

Java Program Vulnerabilities

Sheetal Thakare, Dr.B.B.Meshram

***Abstract*— The Java programming language provides a lot of security features, build directly into the language and also supplied by security relevant APIs and implementations. Nevertheless, simply by choosing Java as the programming language for some program, will not guarantee that the program will be safe against secrecy attacks, integrity or availability attacks. Security concerns should be considered throughout the whole software development process, independent of the particular programming language chosen. This paper discusses all the above mentioned aspects of "Java Security", but focuses on "secure programming techniques" & "vulnerabilities" that should be considered when programming in Java various classifications of vulnerability and corresponding attacks are mentioned with mitigation techniques The paper discusses the input data causing attack and how to prevent it.** [1]

***Index Terms*— Attack, JavaScript, Security, Vulnerability.**

I. INTRODUCTION

The landscape of security vulnerabilities has changed dramatically in the last several years. While buffer overruns and format string violations accounted for a large fraction of all exploited vulnerabilities in the 1990s, the picture started to change in the first decade of the new millennium. As Web-based applications became more prominent, familiar buffer overruns are now far outnumbered by Web application vulnerabilities such as SQL injections and cross-site scripting attacks. These vulnerabilities have been responsible for a multitude of attacks against large e-commerce sites, financial institutions and other sites, leading to millions of dollars in damages. [1]

II. BACKGROUND

In the last twenty years, web applications have grown from simple, static pages to complex, full-fledged dynamic applications. Typically, these applications are built using heterogeneous technologies and consist of code that runs on the client (e.g., JavaScript) and code that runs on the server (e.g., Java servlets). Even simple web applications today may accept and process hundreds of different HTTP parameters to be able to provide users with rich, interactive services. As a result, dynamic web applications may contain a wide range of input validation vulnerabilities such as cross site scripting, SQL injection etc. Unfortunately, because of their high popularity and a user base that consists of millions of Internet users, web applications have become prime targets for attackers. [2]

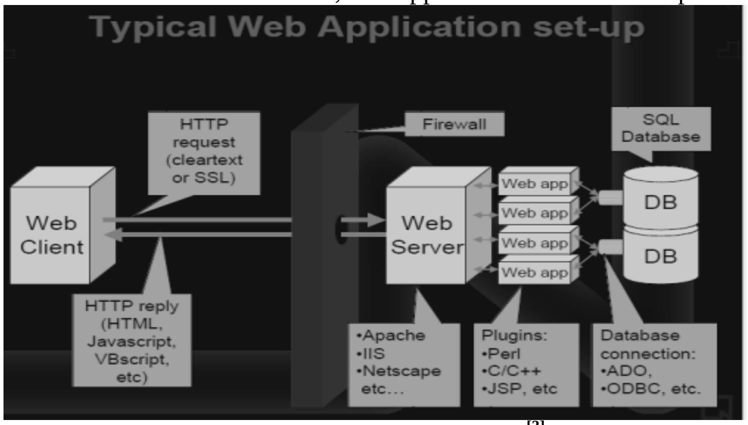
Typical Web Application set-up
HTTP request (cleartext or SSL)
Firewall
SQL Database
Web Client
Web Server
Web app
Web app
Web app
Web app
DB
DB
HTTP reply (HTML, Javascript, VBscript, etc)
•Apache •IIS •Netscape etc…
Plugins: •Perl •C/C++ •JSP, etc
Database connection: •ADO, •ODBC, etc.

Fig 1 Web Application set-up[3]

III. SECURITY IN THE SOFTWARE DEVELOPMENT LIFE CYCLE

Although the software development life cycle can be divided in different ways, it usually includes the following phases, which application developers can repeat iteratively: initialization, specification and design, implementation (coding), testing, deployment, and decommissioning. Although developers should address code security concerns during the entire software product development life cycle, they should specifically focus on three key phases:

. • Implementation.

• Testing.
• Deployment.

A. Implementation

During coding, developers must use best practices that avoid the most critical vulnerabilities in the specific application domain. Example practices include input and output validation, the identification of malicious characters, and the use of parameterized commands. Although these techniques are usually effective in avoiding most Web security vulnerabilities, developers do not always apply them or they apply them incorrectly because they lack security-related knowledge.

B. Testing

Many techniques are available for identifying security vulnerabilities during testing, including penetration testing (by far the most popular technique), static analysis, dynamic analysis, and runtime anomaly detection. The problem is that developers often focus on testing functional requirements and disregard security aspects. Furthermore, existing automated tools usually provide poor results—either low vulnerability detection coverage or too many false positives.

C. Deployment

At runtime, it is possible to include different attack detection mechanisms in the environment. These mechanisms can operate at different levels and use various detection approaches. Obstacles to their use relate to performance overhead and to the false positives that disrupt normal system behaviour. [4]

IV. Secure Programming Guidelines

Secure programming is not possible without obeying some general good programming practices. Therefore guidelines are divided into two parts. The first part contains general rules that should be followed to write secure programs, while the second part concentrates on Java specific topics. The guidelines given in the first part are of a somewhat general nature and similar rules can be formulated for other programming languages as well. [5][6]

Security Guidelines [5] [6]

1. Validate Input and Output
2. Fail Securely (Closed)
3. Keep it Simple
4. Use and Reuse Trusted Components
5. Defense in Depth
6. Only as Secure as the Weakest Link
7. Security By Obscurity Won't Work
8. Least Privilege
9. Compartmentalization (Separation of Privileges)

Java Specific Guidelines [5][6]

1. *Garbage Collection*
2. *Exception Handling*
3. *Serialization and Deserialization*
4. *Java Native Interface (JNI)*

V. Vulnerabilities

An attacker, the "Threat" can exploit Vulnerability, which is a security bug in an application. Collectively this is a Risk.[5] In the following we define and describe common categories of Web Vulnerabilities.[7]

Code Injection (COD)
Cookie Security (COO)
Cross Site Scripting (XSS)
Flow Injection (FLO)
Information Disclosure (INF)
Input Validation (INP)
Path Traversal (PAT)
Resource Injection (RES)
SQL Code Injection (SQL)
Unreleased Resources (UNR)
Logic Errors (LOG)

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

Table 1 Web Application Vulnerability Category [7]

Category Vulnerability	**Description**	**Related to** (attacks)
Code Injection (COD)	• The injection of system and script commands into a web application or an application's server. • This kind of attack mostly applies to server side script languages like PHP or Perl.	INP PAT RES
Cookie Security (COO)	• This category includes several security vulnerabilities based on cookies, e.g., unfiltered cookie content, cookie poisoning, and flow injection via cookies. • In a broader sense, this section is related to session management.	INP
Cross Site Scripting (XSS)	• Here, the attacker inserts code into a URL or link. • The malicious URL must be executed by a web application's user to have an effect. • Misleading users to execute such URLs is supported by the URL itself which looks like a trustworthy URL to the application. • This only works when the application is vulnerable to XSS. • The result can be, e.g., the execution of malicious script (e.g., JavaScript) commands on the client side.	INP
Directory Browsing	*Path Traversal*	
Directory Traversal	*Path Traversal*	
Flow Injection (FLO)	• It is a special case of logic errors and is usually not detectable by security scanners. • This vulnerability is based on setting application states which depend on untrustworthy user data. • Thus, the control flow of an application's code could be influenced by an attacker.	LOG
Information Disclosure (INF)	• An information disclosure security flaw can be defined as the emission of data or information which is not intended to become available to the public. • This can be internal or private data. • There are several issues in this category which are not only programming errors, like the wrong or public storage of sensitive data.	*Information Disclosure (INF)*
Input Validation (INP)	• Usually any input/external data – not only from users – of an application has to be checked to see whether it conforms to intended formats or properties. • Such procedures usually also involve data filtering (sanitization) and adequate *output encoding*. • If input validation, filtering, and output encoding are missing or incomplete, this can enable a variety of attacks.	COD COO RES SQL XSS
Logic Errors (LOG)	• All programming errors, but also errors in system design or specification, which cannot be classified in another security category are called logic errors. • Thus, these errors are not typical programming errors. • Moreover, it is usually not possible to test for resulting security flaws.	FLO
Path Browsing	*see Path Traversal*	
Path Traversal (PAT)	• Can be generally defined as unintended access to application files or directories by injecting (sub) paths and filenames. • The injection, for instance, can take place into application URLs.	COD INP RES
Category Vulnerability	**Definition for the area of IT security**	**Related to** (attacks)
Resource Injection (RES)	• Resource injection flaws can be defined as a category of security vulnerability related to unintentional access to system resources via the application layer, like in the case of path traversal.	COD INP PAT
SQL Code Injection (SQL)	• Results of successful attacks of this category are the execution of arbitrary SQL statements and commands on the application's database backend(s).	INP

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

Unreleased Resources (UNR)	• Some program resources, which are, e.g., variables and class instances (objects), have to be explicitly unloaded for freeing application memory. • If they are not released properly and not caught by the Java garbage collector, they might lead to increased memory consumption. • Thus, in a broader sense, unreleased resources can enable "Denial of Service" attacks and are a concern for an application's security.	*Unreleased Resources (UNR)*

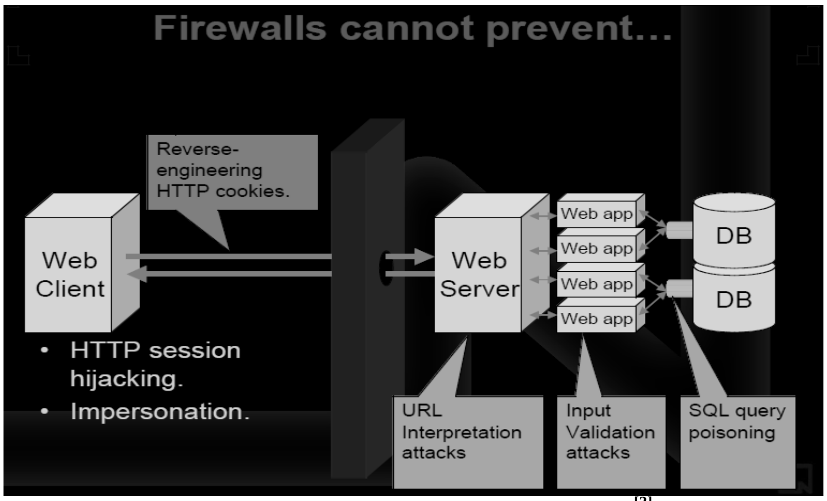

Fig 2 Web Application set-up with Firewall [3]

VI. DETECTING VULNERABILITIES

Identifying security issues requires not only focusing on testing the application's functionalities but also on finding dangerous hidden flaws in the code that attackers can exploit. The two main approaches for detecting vulnerabilities are white-box analysis and black-box testing.[4]

A. White-box analysis

White-box analysis consists of examining the code without executing it. Developers can do this in one of two ways: manually, during code inspections and reviews; or automatically, using automated analysis tools. Code inspection is the process in which a programmer's peers systematically examine the delivered code, searching for programming mistakes. Security inspections are the most effective way to minimize vulnerabilities in an application; it is a crucial procedure when developing software for critical systems. Nevertheless, such inspections usually take a long time, are expensive, and require deep knowledge of Web security. A less expensive alternative is code review, a simplified version of inspections that is useful for analyzing less critical code. Reviews are also done manually, but they do not include a formal inspection meeting. Several experts perform the review individually, and a moderator filters and merges the outcomes. Although also an effective approach, code review is still quite expensive. To reduce the cost of white-box analysis, developers sometimes rely on automated tools, such as static code analyzers. Static code analysis tools vet software code, either in source or binary form, in an attempt to identify common implementation-level bugs. The analysis performed using existing tools varies depending on their sophistication, ranging from those that consider only individual statements and declarations to others that consider dependencies between lines of code. Among their other uses, such as for model checking and data flow analysis, these tools automatically highlight possible coding errors. The main problem is that exhaustive analysis is difficult and cannot find many security flaws because of the source code's complexity and the lack of a dynamic (runtime) view.

B. Black-box testing

Black-box testing refers to the analysis of program execution from an external point of view. In short, it consists of comparing the software execution outcome with the expected result. Testing is probably the most used technique for software verification and validation. There are several levels for applying black-box testing, ranging from unit to integration to system testing. The testing approach also can be formal (based on models and well-defined test specifications) or less formal (referred to as "smoke testing," a type of rough testing intended to quickly reveal simple bugs).The goal of robustness testing, a specific form of black-box testing, is to characterize the system's behavior in the presence of erroneous input conditions. Penetration testing is a special type of robustness testing that analyzes program execution in the presence of malicious inputs, searching for potential vulnerabilities. In this

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

approach, testers apply fuzzing techniques, which consist of submitting unexpected or invalid items of data, to a Web application and review its responses, using HTTP requests. Testers do not need to know the implementation details—they test the application inputs from the user's point of view. The number of tests can reach hundreds or even thousands for each vulnerability type.

C. Limitations of vulnerability detection

Penetration testing and static code analysis can be manual or automatic. Because manual tests or inspections require specialized security resources and are time-consuming, automated tools are the typical choice of Web application developers. An important fact when considering the limitations of vulnerability detection tools is that testing for security is difficult. Indeed, measuring an application's security is challenging: although finding some vulnerabilities can be easy, guaranteeing that the application has no vulnerabilities is difficult. Both penetration testing and static code analysis tools have intrinsic limitations. Penetration testing relies on effective code execution; however, in practice, vulnerability identification only examines the Web application's output. Thus, the lack of visibility into the application's internal behavior limits penetration testing's effectiveness. On the other hand, exhaustive source code analysis can be difficult. Code complexity and the lack of a dynamic (runtime) view might prevent finding many security flaws. Of course, penetration testing does not require access to the source code, while static code analysis does. Using the wrong detection tool can lead to the deployment of applications with undetected vulnerabilities.

VII. DETECTING ATTACKS

Attack detection consists of identifying deviations from learned behavior. Attack detection tools use approaches based on either anomaly detection or signatures.

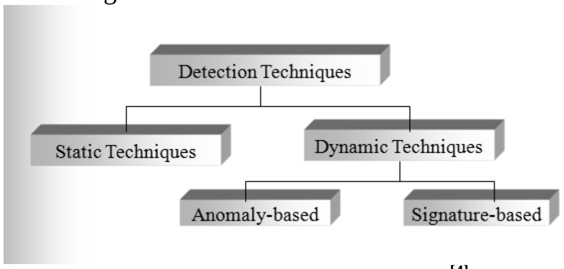

Fig 3 Attack Detection Approach [4]

- We need the following fields for an effective investigation:
  – Source IP
  – Timestamp
  – HTTP Method
  – URI requested
  – Full HTTP data sent
- Attack data could be in:
  – URI(uniform resource identifier)
  – HTTP headers from client
  – Cookie

- Basically anywhere

Detection Techniques

- Using static techniques
  – Happens post-occurrence of event
  – Parse log files using standard tools/techniques
  – Aim is forensics investigation
- Using dynamic techniques
  – Detect the attack as it happens
  – Trigger alarms when attack is happening

Aim is detect/prevent in real-time

Static detection techniques

Data sources to look at:

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

– Web Server Logs
– Application Server Logs
– Web Application's custom audit trail
– Operating system logs

What's missing?

– POST data (only GET data available)
– HTTP Headers only partially represented

Cookie or Referrer data depends on web server

Static Detection Fails to detect:

● HTTP Header attacks can't be detected:
  – The Template of attack can't be detected
  – Attacks that overflow various HTTP header fields
● Web application attacks in a POST form
  – SQL injection
  – Cross-site scripting

Forceful browsing – user tries to access page without going through prior pages that would ensure proper authentication and authorization.

Static Detection does detect:

● Automated attacks using tools such as Nikto or Whisker or Nessus
● Attacks that check for server misconfiguration (../../winnt/system32/cmd.exe)
● HTML hidden field attacks (only if GET data –rare)
● Authentication brute-forcing attacks

Order ID brute-forcing attacks (possibly) – but if it is POST data, then order IDs cannot be seen

Dynamic detection techniques

Methods:

– Application Firewall
– In-line Application IDS
– Network-based IDS (possibly) adapted for applications

Advantages:

– Complete packet headers and payload available
– Including HTTP headers
– POST request data

URI request data

The web application intrusion detection space is divided into two possibilities:

– Signature-based
– Anomaly-based

Each has its own implementation and effectiveness issues.

Table 2 Signature Based Approach Vs Anomaly Based [10]

Signature-based	**Anomaly-based**
Easier to implement	More complicated
Cheaper to modify, without expert help	Mostly commercial solutions
False positives	False positives are fewer
As well as false negatives	False negatives as well
Popular for detecting known web server attacks. Can be tweaked to do decent web application detection.	Used for both web server, as well as web application attacks

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 3, May 2013

VIII. TYPE OF ATTACKS

1. Inject malicious data into java applications.

1.1 Parameter Tampering
1.2 URL Manipulation
1.3 Hidden Field Manipulation
1.4 Http Header Tampering
1.5 Cookie Poisoning [8]

1.1 Parameter tampering:
- pass specially crafted malicious values in the fields of HTML forms.

1.2 URL manipulation:
- use specially crafted parameters to be submitted to the Web application as part of the URL.

1.3 Hidden field manipulation:
- set hidden fields of HTML forms in Web pages to malicious values.

1.4 HTTP header tampering:
- manipulate parts of HTTP requests sent to the application.

1.5 Cookie poisoning:
- Place malicious data in cookies, small files sent to Web-based applications.

2. Manipulate applications using malicious data.

2.1 SQL Injection
2.2 Cross-Site Scripting
2.3 Http Response Splitting
2.4 Path Traversal
2.5 Command Injection[8]

2.1 SQL injection:
- pass input containing SQL commands to a database server for execution.

2.2 Cross-site scripting:
- exploit applications that output unchecked input, this tricks the user to execute malicious scripts.

2.3 HTTP response splitting:
- exploit applications that output input verbatim to perform Web page defacements or Web cache poisoning attacks.

2.4 Path traversal:
- exploit unchecked user input to control which files are accessed on the server.

2.5 Command injection:
-exploit user input to execute shell commands.

IX. CONCLUSION

JavaScript are being exploited to wreak havoc on the user’s browser and operating system without even violating the security policies of the browser. Simple code can be written that eats up memory or other resources and quickly crash the browser and even the operating system itself. Further deceptive programming practices can be employed to annoy or trick the user into actions they might not intend. So the Web applications accepting user input need to be careful to properly validate such data before accepting it, and to sanitize it before writing it into a Web page. Failing to do so can result in cross-site scripting vulnerabilities, which are as harmful as violations of the same origin policy would be. It is the responsibility of individual developers to write clean, careful code that improves the user experience and always be on the lookout for malicious users trying to bypass their checks. Also the Software should be compounded with counter measures for above listed attacks. This can allow for expanded features. It can also reduce post attack coding efforts.

REFERENCES

[1] Whitepaper Secure Programming in Java (c) 2005, EUROSEC GmbH Chiffriertechnik & Sicherheit.

[2] S. Institute. Top Cyber Security Risks, September 2009. http://www.sans.org/ top-cyber-security-risks/summary.php

[3] www.net-square.com,saumil@net-square.com, top ten web attacks.

[4] Defending against Web Application Vulnerabilities, Nuno Antunes and Marco Vieira, University of Coimbra, Portugal, Published by the IEEE Computer Society, 0018-9162/12/$31.00 © 2012 IEEE,

[5] www.cgisecurity.com.

[6] Security Code Guidelines, Sun Microsystems, Inc. (http://java.sun.com/security/seccodeguide.html).

[7] Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.

[8] Finding Security Vulnerabilities in Java Applications with Static Analysis, V. Benjamin Livshits and Monica S. Lam Computer Science Department Stanford University {livshits, lam}@cs.stanford.edu.

[9] A Process for Performing Security Code Reviews, published by the IEEE computer society ■ 1540-7993/06/$20.00 © 2006 IEEE ■ IEEE security & privacy.