Java Web Application Security

RJUG

Nov 11, 2003

RDC

Ralph Durkee Consulting

Ralph Durkee

SANS Certified Mentor/Instructor

SANS GIAC

Network Security and

Software Development

Consulting

Agenda

- Intro and Definition
- Web Application Risk Overview
- Threat Categories
- Overview Top 10 Vulnerabilities
- Examine some vulnerabilities in detail
- Sun Security Code Guidelines for Java

Definition of Web Application Vulnerabilities

- **Web Applications:** Software applications that interact with users or other applications using HTTP/s
- Could include Web services which communicate between Applications via XML
- **Web Application Vulnerabilities:**

Weakness in custom Web Application, architecture, design, configuration, or code.

Web Applications
What's the Risk?

- **Risk = Threat * Vulnerability * Asset**
- Threat Level for Internet Web Servers?
  - Web attacks are very frequent
    (3-8 attacks / probes per day per IP is normal)
  - Port 80 consistently one of the top 10 attacked
    (www.incidents.org)
- Vulnerabilities
  - Plenty to come on Vulnerabilities
- Asset
  - Estimate of all potential losses and costs.

Traditional Threat Categories

Threat Target	Mitigation	Sophistication
Network Protocols	Firewalls, Routers etc	Automated
Operating System	Patching, Hardening, Minimize Services	Automated
Commercial Applications	Patching, Configuration	Automated

Custom Application 4th Threat Category

Threat Target	**Mitigation**	**Sophistication**
Network Protocols	Firewalls, Routers etc	Automated
Operating System	Patching, Hardening, Minimize Services	Automated
Commercial Applications	Patching, Configuration	Automated
Custom Application Software	**Arch. Design & Code Reviews** **Appl. Testing** **Appl. Scanners**	**Not Yet Automated**

How Bad Is It?

- Sanctum reports 97% of 300 Web Applications Audited were Vulnerable
- Gartner reports 75% of attacks today are at the Application Level
- If it really is that bad, why aren't majority of web sites defaced and infected with worms?

If it really is that bad, Why?

Why aren’t majority of web sites defaced and infected with worms?

- Very difficult to write automated worms against custom software.
- Good news: What can be automated by attackers, can also be discovered by security scanners.
- Without automation, attack of web applications is semi-manual one-off process.

If it really is that bad, Why?

(continued)

- Technical difficulty eliminates the lowest level script kiddies, but do-able by even intermediate attackers.
- Difficult to estimate the number of Web Applications already compromised especially if attackers are quietly keeping "ownership" rather than defacing.

OWASP

Open Web Application Security Project

WWW.OWASP.ORG

- Dedicated to helping organizations understand and improve the security of their web application and web services.
- Publish Top 10 Web App. Vulnerabilities
- Open Source Projects (WebGoat, WebScarab)

OWASP Top 10

OWASP	Description
A1 - Unvalidated Parameters	Malicious input may attack server or back-end components.
A2 - Broken Access Control	Access not well defined with controls which may be bypassed with client side manipulation.
A3 - Broken Account & Session Management	Session or Account authentication may be disclosed or guessed.

OWASP Top 10

OWASP	Description
A4 - Cross Site Scripting	Malicious script is stored by the Web Application and given to an unsuspecting victim.
A5 - Buffer Overflows	Providing too much input allows code execution to be manipulated.
A6 - Cmd Injection	Manipulates server evaluation of input to execute commands.

OWASP Top 10

OWASP	Description
A7 - Error Handling	Diagnostics reveal platforms, architecture and identifiers.
A8 - Insecure Cryptography	Improper usage and home grown algorithms
A9 - Remote Admin flaws	Inadequate controls and protection.
A10 - Server misconfiguration	Not using security configuration guidelines.

Types of Input for Validation

- Form Input parameters

```
<INPUT …  name=userid  value="shmoe" >
```

- Hidden form parameters

```
<INPUT TYPE=hidden … name=sessionid
  value="928302757461044230129736" >
```

- **Keep mind all input parameters are visible and can be modified.**

URL Query String Parameters

- **Example**

  https://www.rd1.net/servlet/login**?userid=shmoe&password=dumb…**
- Least secure place for parameters
- Stored in browser history cache
- Visible to shoulder surfing
- Could be Book Marked
- App. Servers often allow this transparently.

Cookies

- Flavors
  - Persistent with expiration date / time, stored on client hard drive
  - Non-persistent, no expiration, stored in memory until browser closed.
  - Secure option (request https transmission)

Cookies

- Example: Server Response and Client Request

Set-Cookie:

siteid=91d3dc13713aa579d0f148972384f4;
path=/;
expires=Wednesday, 12-Oct-2003 02:12:40
domain=.www.rd1.net
secure

Cookie: siteid=91d3dc13713aa579d0f148972384f4

HTTP Headers

- Carry a good deal of information
- Access through various program API’s.
- Easy to use HTTP header input without considering the need for validation.

HTTP Headers

```
Accept: image/gif, image/x-xbitmap,
  image/jpeg, . . .  , */*
Referer: http://rd1.net/index.html
Accept-Language: en-us
Content-Type: application/x-www-form-
  urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE
  5.5; Windows NT 5.0; T312)
Host: rd1.net
Content-Length: 46
```

HTTP Headers Java Sample

```
URL server_url = new URL( urlstr);
   URLConnection conn = server_url.openConnection();
… // Additional code
int len = conn.getContentLength();
```

- What if len < 0 ?
- Or a very large value?
- Is the call getting the http header value or the actual length?
- What happens if they differ?

Examples of Malicious Input

- Buffer overflows
- Command or Script injection
- SQL injection
- Cross Site Scripting (XSS)
- Improper Error Handling
- Input encoding

Buffer Overflows

- Traditional C/C++ software
  - Particularly dangerous
  - May Allow arbitrary remote code execution.
- Not as Serious, but still a problem for Java.
  - Needs to be handled gracefully
  - Check before usage
  - Catch exceptions to prevent a Denial of Service.
- Front End software should help protect back-end.
- Example: Check size before passing to DB or OS.

Command or SQL injection

- Input may contain special Meta-characters
- Some Meta-characters will have significance to the script, OS or database interpreter
- Meta-characters may be encoded to attempt to circumvent filtering
- CERT URL http://www.cert.org/tech_tips/malicious_code_mitigation.html

SQL Injection

- JSP Example

```
String squery = “select userid from users where
   uname=“ + request.getParameter("user_nm”) +“;”;
```

- But What if ..

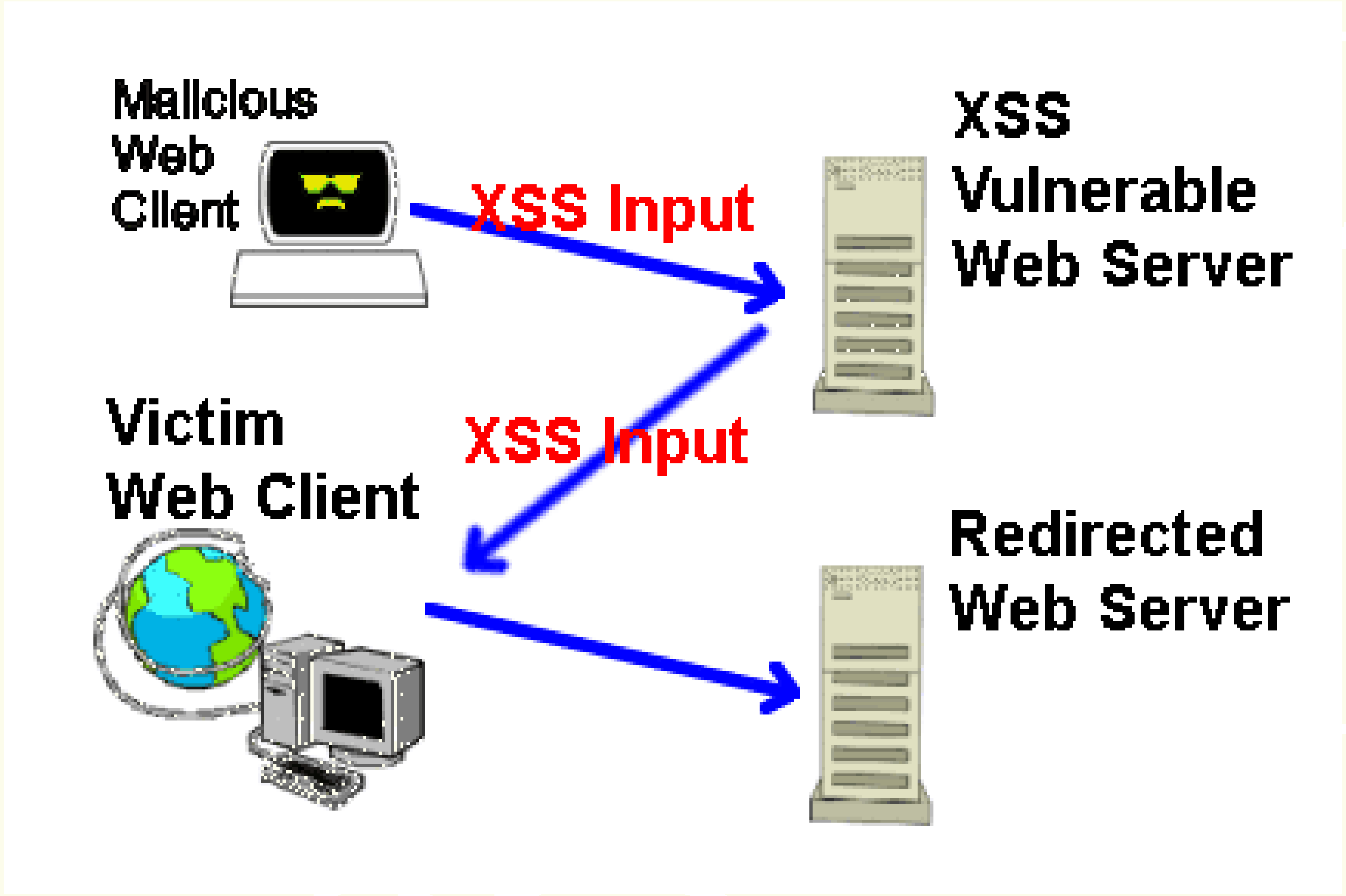
```
user_nm='%27 or %27x%27=%27x'
```

- The %27 is an encoded quote and the <u>or</u> <u>‘x’=‘x’</u> will always be true,
- May bypass the authentication and execute arbitrary SQL statements

Cross Site Scripting (XSS)

- Any dynamic web page using unvalidated data is vulnerable.
- Data may contain html or client side scripting
- May originated from a malicious source
- May attack another client via web server

XSS – Example


Malicious Web Client
XSS Input
XSS Vulnerable Web Server
Victim Web Client
XSS Input
Redirected Web Server

XSS – Example

- Malicious XSS Input

  `comment=<SCRIPT>malicious code</SCRIPT>`
- Comment is placed in a DB
- Served up on a web page to an unsuspecting victim.
- Victims browser execute malicious code, and/or is sent to another site.

Error Response Information

- Helpful Debug messages
- Provides way too much information!
- Very helpful to potential attacker.

```
Microsoft OLE DB Provider for ODBC
  Drivers error '80004005'
[Microsoft][ODBC Microsoft Access
  97 Driver] Can't open database
  'VDPROD'.
```

Improper Error Handling

- Possible Denial of Service
- Java Exceptions and Stack traces
- Very revealing!

```
java.sql.SQLException: ORA-00600: internal error
   code, arguments: [ttcgnd-1], [0], [], [], [],
at oracle.jdbc.dbaccess.DBError.throwSqlException
   (DBError.java:169)
at oracle.jdbc.ttc7.TTIoer.processError
   (TTIoer.java:208)
```

Inappropriate Information Disclosure

- Web Responses may provide inappropriate information.
- Example 1: Helpful web pages that let you know when a valid user id has been guessed.
  - Response for valid user/invalid password should be exactly identical to invalid user.
  - Even subtle differences are sufficient
- Example 2: Source html comments.

Input Encoding

- Malicious Input can be encoded in many ways
- Each software layer and script languages has additional encoding.
- Attempts to avoid negative filtration.
- Examples; &amp; &#38;
- Double encoding: &#38;amp&#59;
- Triple ?

Where to Validate

- Client validation -- is helpful but does not provide security
- Server validation – Everything received from client must be suspect
- Validate before usage or interpretation.

How to validate

- Positive filtering preferred rather than Negative
- Canonical form (decode) where appropriate
- Encode special characters where appropriate
- Many types of encoding

What to check

- Minimally Allowed Character Set
  (Specific to data field)
- Numeric Range
- Length too short or too long
- Optional or required
- Encoding of potential meta-characters that must be allowed.
- Null bytes

Tips from Sun Security Code Guidelines

Full text Available On-line From

http://java.sun.com/security/seccodeguide.html

- Public Static fields
- Reducing scope
- Public methods and Variables
- Protecting packages
- Make objects immutable if possible
- Serialization
- Native methods
- Clear sensitive information

Public Static Variables

- All Public Static variables should be `final`
- Ensure that only the appropriate code has permission to change

Reducing scope

- Each class, method and variable provides an additional access point.
- Make classes method private where appropriate
- Make all variables private
- Restrict scope to the minimal

Public Methods & Variables

- Avoid Public Variables
- Methods modifying sensitive internal states need to include security check

Protecting packages

- Use sealed Jar files
- Attacker may try to gain access to package members by defining new classes within the attacked package by extending it.

Make objects immutable if possible

- Especially arrays, vectors etc.
- Prevent modifications
- Provides better concurrency
- Avoid returning reference to sensitive data
- Never store user given data directly

Protect Serialization

- Serialized Object is outside Java Security controls
- Requires additional controls to protect data
- Consider encryption or Digital Signatures
- Additional tips available on-line

java.sun.com/security/seccodeguide.html

Native methods

CAREFUL! Examined :

- Return values and parameters
- bypass security checks
- Are they public, private, ...
- Whether they contain method calls which bypass package-boundaries, thus bypassing package protection

Clear Sensitive Information

- Such as Passwords etc.
- Prefer Mutable (such as array)
- Rather than immutable (such as a string)
- Perform explicate clearing of the information
- Do not leave it for the garbage collection.

The Future for Web Application Security

- Application Security Testing tools
  - Available but expensive
  - still a bit green
- Application Firewalls
  - also a bit new and bit overpriced.
- Better understanding of Vulnerabilities
- Better Security input validation support from Development tool Vendors
- More Design & Code Reviews.

Resources

- www.OWASP.org/
  - Top Ten Web Application Vulnerabilities
  - OWASP guide
  - WebGoat
  - News, e-mail lists, articles etc.
- www.SecurityFocus.com/
  - Vulnerability information
  - News, e-mail lists articles etc.