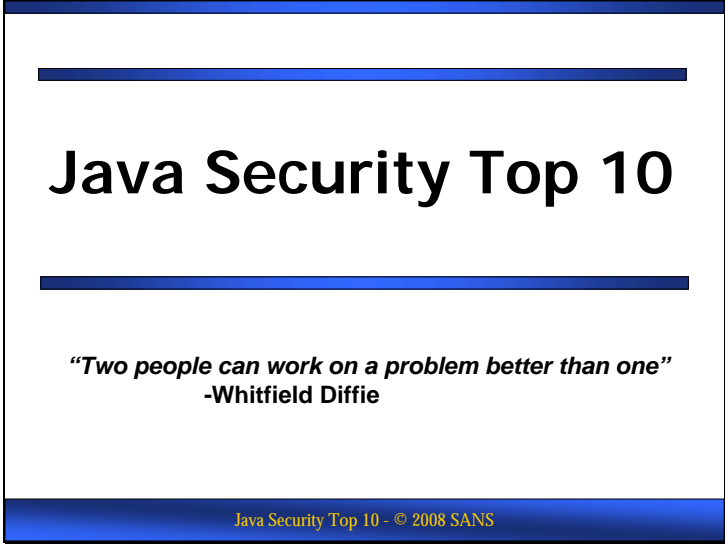


Slide 1



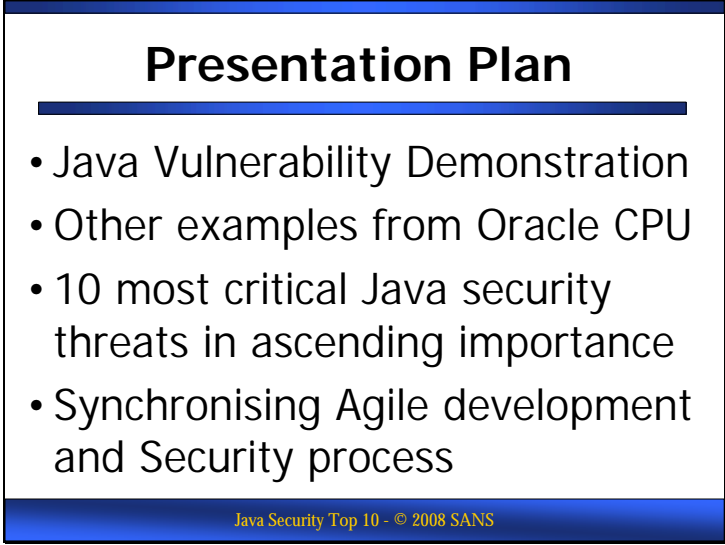
The slide features a white background with a blue border. At the top, there is a thick blue horizontal line. Below this line, the title "Java Security Top 10" is centered in a large, bold, black sans-serif font. Another thick blue horizontal line is positioned below the title. Underneath the second line, a quote is centered: "*Two people can work on a problem better than one*" followed by "-Whitfield Diffie" on the next line. At the bottom of the slide, there is a blue horizontal bar containing the text "Java Security Top 10 - © 2008 SANS" in a small, yellow, sans-serif font.

# Java Security Top 10

*"Two people can work on a problem better than one"*  
-Whitfield Diffie

Java Security Top 10 - © 2008 SANS

Hello my name is Paul Wright and welcome to the "Java Security Top 10".

A presentation slide with a blue border and a blue header bar. The title "Presentation Plan" is centered in the header bar. Below the title is a list of four bullet points. At the bottom of the slide, there is a small yellow text box containing the text "Java Security Top 10 - © 2008 SANS".

## Presentation Plan

- Java Vulnerability Demonstration
- Other examples from Oracle CPU
- 10 most critical Java security threats in ascending importance
- Synchronising Agile development and Security process

Java Security Top 10 - © 2008 SANS

After the introduction I will demonstrate a current Java vulnerability to you and discuss some of the examples to be seen on bugtraq recently.

Then we will go through the Java Security Top 10.

I will then discuss how to synchronize Agile development methodology with the security process.

Then finally I will conclude with questions.

## Introduction

- Business case – 3 banks this year  
[www.privacyrights.org](http://www.privacyrights.org) 224 million
- Developers told Java secure by default
- This is incorrect as I will show
- TTF vulnerability < 1.5.0\_08 JRE
- John Heasman NGS Software.

Java Security Top 10 - © 2008 SANS

### *Introduction*

The business case for Java Security can be conveyed by reading the web site [www.privacyrights.org](http://www.privacyrights.org) where the many data breaches of national companies are made public record and the share prices fall accordingly.

This year OmniAmerican Bank, Great Falls financial services and GE money have all had to publicly admit that they had been the victims of a data breach.

But Java is secure already isn't it ???

When I was taught Java by a leading IBM instructor at the University of Manchester we were told that Java was completely secure without any extra effort on behalf of the developer.

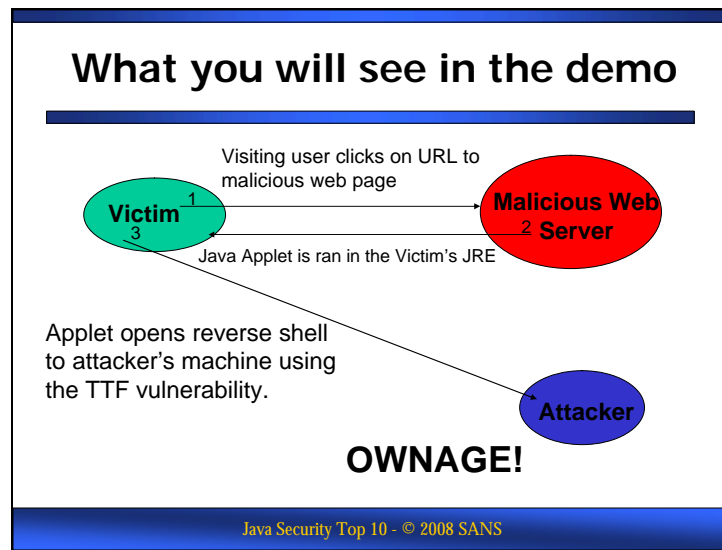
Java may be more secure locally than C++ but it still has many security weaknesses given its network capabilities.

To show that Java is not secure by default we will demonstrate a current Java vulnerability being exploited.

Please note that the bug has been fixed in the latest JVMs but anything prior to 1.5.0\_08 is vulnerable.

The vulnerability was discovered by John Heasman of NGS Software.

## Slide 4



This is how the demo works in basic terms:

Victim browses to a malicious web server via a malicious URL in a phishing email or search engine entry for example.

Java applet from that web server loads in victims browser and exploits the TTF vulnerability in the user's JRE.

Shellcode opens up a reverse shell to the attackers machine via the user's JRE

This demonstration is coded with all the components in the one VM.

Firstly we have the victim's browser in the bottom left hand corner which is running under the account named Victim.

Then we have the attackers shell which is running with the account "Attacker".

Lets echo the username to show this.

The attacker leaves a netcat listener open to receive reverse shells from victims that browse to the MALICIOUS web server.

Now Victim browses to a MALICIOUS WEBSITE.

The Applet exploits their local JVM and connects back to the Attacker's workstation as you can see.

The victim has been owned but they do not know it.

Lessons to learn are that we must update Java JVMs regularly.

Phishing attempts do not require users to interact with the site just visit it.

Egress rules on Vista firewall can be very useful for Windows.

Biggest lesson for us is that Java is not secure by default.

Let's have a look at how the exploit works.

back to powerpoint

## Vulnerable source code

```
InputStream is =
    this.getClass().getResourceAsStream("malformed.txt");
byte[] bFont = new byte[is.available()];
is.read(bFont);
ByteArrayInputStream bais = new
    ByteArrayInputStream(bFont);
Font font = Font.createFont(Font.TRUETYPE_FONT, bais);
font = font.deriveFont(32.0f);
Graphics g = this.getGraphics();
g.setFont(font);
textArea.append("[+] Set font, ready to render text\n");
g.drawString("aaaaaaaaaaaaaaaa", 20, 20);
```

Java Security Top 10 - © 2008 SANS

The vulnerable code is underlined in the slide. This True type font vulnerability is essentially a problem of the `createFont` method not validating input when using TRUETYPE Fonts.

It is read into `bFont` which is then read into `bytearrayinputstream` `bais`.

Then `createFont` method takes this as input without validating it.

The shellcode then interacts with the OS via the JVM and opens up the reverse shell.

This vulnerability does not have to be exploited through an Applet.

Local Java code could also do the same thing.

The mitigation to this is to upgrade the JRE.

One problem with this is that the old JRE is often left behind and it is possible for Java apps to specify the version of the JRE that they need in order to run.

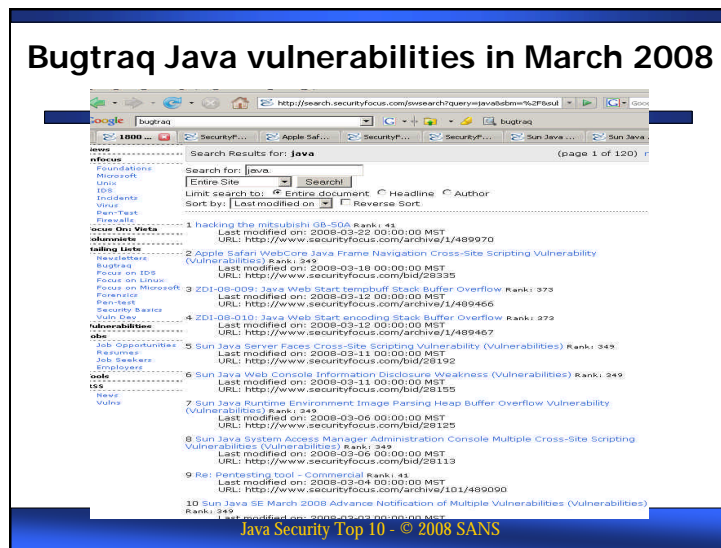
This is because of the incompatibility of certain vintage of application with a certain version of the JRE.

Therefore completely remove the old JVM.

This demonstration proves that Java is not 100% secure.

But is this a one off or are there other Java security vulnerabilities?

Slide 6



A search on bugtraq shows 8 java related security vulnerabilities in the month of March.

There are 8 vulnerabilities reported in the month of March 2 of which are buffer overflows such as the Java web start tempbuff stack buffer overflow.

### Java vulnerabilities in Oracle

- Vulnerabilities in PL are in underlying Java.
- e.g. DBMS\_CDC\_IPUBLISH package calls a Java method called **ChangeTableTrigger** which is the actual source of the SQL Injection vulnerability.
- Therefore vulnerability scanners need to audit Java in Oracle DB. See attached code in Paper.
- In Oracle's 2008 April 15<sup>th</sup> CPU the highest criticality vuln (CVSS 9.3) is in Jiniator, Oracle's Applet JVM.

Java Security Top 10 - © 2008 SANS

Additionally Oracle Security Alerts have shown Java vulnerabilities recently.

SQL injection vulnerabilities in PL packages are often actually in the underlying Java

e.g. Vulnerable DBMS\_CDC\_IPUBLISH package calls a Java method called **ChangeTableTrigger** which is the source of the vulnerability.

Therefore vulnerability scanners need to audit Java in Oracle DB. See attached code in Paper.

In Oracle's April 15<sup>th</sup> CPU the highest criticality vuln (CVSS 9.3) is in Jiniator, Oracle's Applet JVM.

Securing Java is obviously a problem but that's what we are here to solve. What is needed are security professionals who understand how to audit Java software.

--This code can be used to statecheck vulnerable Java code in Oracle.

```
DECLARE V_OBJID NUMBER:=0;
V_HASH NUMBER:=0;
V_BUFFER RAW(32767);
CUR NUMBER;
RES NUMBER;
POS NUMBER;
LEN NUMBER;
BEGIN DBMS_OUTPUT.ENABLE(1000000);
SELECT distinct SYS.OBJ$.OBJ# INTO V_OBJID FROM SYS.OBJ$, SYS.USERS$ WHERE
SYS.USER$.USER#=SYS.OBJ$.OWNER# AND SYS.OBJ$.TYPE#=29
AND SYS.USERS$.NAME='SYS'
and SYS.OBJ$.NAME='oracle/CDC/ChangeTableTrigger';
CUR:=DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(CUR,'SELECT S.PIECE FROM SYS.IDL_UB1$ S WHERE S.OBJ# = :1',DBMS_SQL.NATIVE);
DBMS_SQL.BIND_VARIABLE(CUR, ':1', V_OBJID);
DBMS_SQL.DEFINE_COLUMN_RAW (CUR, 1, V_BUFFER, 32767);
RES := DBMS_SQL.EXECUTE_AND_FETCH (CUR);
IF RES > 0 THEN DBMS_SQL.COLUMN_VALUE_RAW(CUR,1,V_BUFFER);
V_HASH:= V_HASH + SYS.DBMS_UTILITY.GET_HASH_VALUE(V_BUFFER,1,1073741824);
DBMS_SQL.CLOSE_CURSOR (CUR);
DBMS_OUTPUT.PUT_LINE(V_HASH);
V_BUFFER:=NULL;
```

END IF;

END;

/

<http://www.oracleforensics.com/wordpress/wpcontent/uploads/2008/03/ukougFINAL.ppt>

[ppt](#)

This is simplified code from [www.nextgenss.com/research/papers/LiveResponse.pdf](http://www.nextgenss.com/research/papers/LiveResponse.pdf)

by David Litchfield.



## Howto find Java vulnerabilities?

- Dynamic Analysis is during runtime.
- Static Analysis is source code review.
- Manual- **grep**/**findstr** and **regex**
- Free automated source code review tools  
e.g. Findbugs

<http://findbugs.sourceforge.net/>  
<http://findbugs.sourceforge.net/bugDescriptions.html>  
<http://checkstyle.sourceforge.net/>  
<http://www.parasoft.com/jsp/products/home.jsp?product=Jtest>  
<http://jcodereview.sourceforge.net/>

Java Security Top 10 - © 2008 SANS

### How to find Java vulnerabilities so that they can be fixed?

The two main methods are dynamic analysis and static analysis.

Dynamic analysis consists of tests run on the code during execution such as fuzzing.

Static analysis is a process of searching the source code for known vulnerable constructs.

This may be done by manually reading the source code or searching using grep/findstr and regex.

So in the case of the TTF vulnerability simply searching the source code for the string “TRUETYPE\_FONT” would provide a type of check.

Grepping for all the variations could be very time consuming so there are automated tools to make this more time efficient.

Free automated source code review tools such as findbugs can help identify general issues.

<http://findbugs.sourceforge.net/>

Findbugs publishes it's vulnerabilities publicly in an easy to read format at this URL.

<http://findbugs.sourceforge.net/bugDescriptions.html>

Other automated Java code review tools.

<http://checkstyle.sourceforge.net/>

<http://www.parasoft.com/jsp/products/home.jsp?product=Jtest>

<http://jcodereview.sourceforge.net/>

## Java Security Code Review

---

Commercial Security Java code review tools:

- <http://www.fortify.com/products/sca/>
- <http://www.ouncelabs.com/>
- <http://developer.klocwork.com>

Top 10 examples are partly from a Fortify scan on the current SPRING framework  
SPRING is a replacement for NetBeans  
<http://opensource.fortifysoftware.com/welcome.html>

Java Security Top 10 - © 2008 SANS

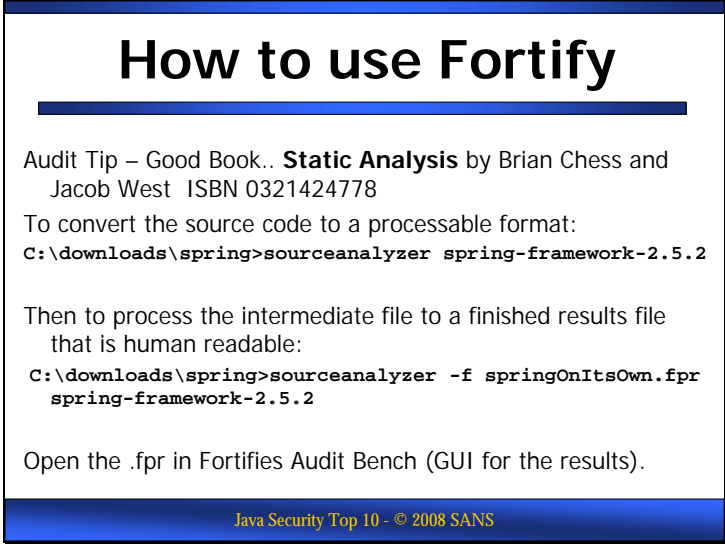
Online tools for searching public code are also available.

For hard core security issues a commercial code review tool such as Fortify's Source Code Analyzer is required.

It's competitors are Ouncelabs and Klocwork.

In order to illustrate some of the issues in our Top 10 we will use the example of running Fortify SCA against SPRING.

SPRING is a popular set of design patterns used instead of Java Beans with additional features such as object lifecycle management. <http://www.springframework.org/>



## How to use Fortify

Audit Tip – Good Book.. **Static Analysis** by Brian Chess and Jacob West ISBN 0321424778

To convert the source code to a processable format:  
`C:\downloads\spring>sourceanalyzer spring-framework-2.5.2`

Then to process the intermediate file to a finished results file that is human readable:  
`C:\downloads\spring>sourceanalyzer -f springOnItsOwn.fpr spring-framework-2.5.2`

Open the .fpr in Fortifies Audit Bench (GUI for the results).

Java Security Top 10 - © 2008 SANS

To convert the source code to a processable format you would issue this command.

```
C:\downloads\spring>sourceanalyzer spring-framework-2.5.2
```

Then to process the intermediate file to a finished results file that is human readable via the Audit Bench application use this command.

```
C:\downloads\spring>sourceanalyzer -f springOnItsOwn.fpr spring-framework-2.5.2
```

Then open the .fpr in Fortifies Audit Bench which is a GUI for the results of the scan.

I would recommend **Static Analysis** book by Brian Chess and Jacob West which includes a demo of the software.

*So what are the security bugs did Fortify find and what other Java security issues make up the Top10?*

## Top 10 Java Security Issues

- Following are the most common vulnerabilities found in Java Applications
- They are all potentially of high criticality depending on the context
- Skilled auditor needs to interpret these findings
- In ascending order of importance countdown to the most vulnerable.

Java Security Top 10 - © 2008 SANS

Following now are the most common vulnerabilities generally found in Java Security Audits.

Potentially they are all of high criticality depending on their context within the application which is where the skilled auditor is needed to interpret these findings.

These are in increasing order of importance building up to the most insecure aspects of Java code at the end.

## 10. System Information leaks

```
out.println("*** Root cause is: "+ rootCause.getMessage());  
rootCause.printStackTrace(new java.io.PrintWriter(out));
```

- **printStackTrace** will print out memory and could result in a system information leak.
- This is commonly used as part of a debugging process.
- Care should be taken to make sure that no potentially sensitive information is included in this debugging information.
- Devs should not see passwords/CC (PCI)

Java Security Top 10 - © 2008 SANS

`printStackTrace` will print out memory and could result in a system information leak.

This is commonly used as part of a debugging process but care should be taken to make sure that no potentially sensitive information is included in this debugging information such as passwords and credit card numbers.

Devs should not see credit card numbers is part of PCI.

At the start of the Java Security audit the status of the code should be agreed i.e. is this development, staging or production code?

If this is production code then it is a finding as debugging code should not be left in production code, though it often is.

## 9. Java Scope

- **Private** means can only be seen in the class.
- **Default** is **Package** scope AKA **Friendly** i.e. anything in the package.
- **Protected** is package and subclass.
- **Public** means any code can see the members of the class.

Java Security Top 10 - © 2008 SANS

**Below are the generally accepted definitions of Java scope key words:**

**Private** means can only be seen in the class.

**Default** is **Package** scope aka **Friendly** i.e. anything in the package/directory(folder)

**Protected** is package and subclass.

**Public** means anyone can see the members of the class.

**Private protected** has been deprecated.

## Java Scope misunderstandings

- **Private** variables can actually be accessed outside of the application if **serializable** unless also declared...? Quiz Question>
- An added **inner class** makes a **private** class downgrade to **protected**
- **Static** does not mean that the value of a variable cannot be changed
- **Final** means cannot be changed
- Not used enough no **const** keyword like C.

Java Security Top 10 - © 2008 SANS

**However many developers do not realize the following additional facts regarding Java Scope:**

**Private** can actually be accessed outside of the class and the whole application i.e. the private variable can be dumped to the OS if **serializable** but not if the variable is also declared **transient**.

The class needs to **implement serializable** to dump its variables but declaring the variable as **transient** means that the variable cannot be dumped even if the class does **implement serializable**.

An added **inner class** makes a **private** class downgrade to **protected** in order for the **inner class** to access its host.

**Static** does not mean that the value of a variable cannot be changed, it means that each reference is pointing to the same value i.e. all instances are the same.

Therefore can change them all in one go (one copy kept in the class of which a reference is replicated to all the objects instantiated from that class).

A **static** variable is subject to the Scope key words above (**public, private** etc).

Good idea to make static variable **final** as well to stop it from being changed if the value should not be changed i.e it is a constant.

e.g. **public final static double PI = 3.141;**

Not like C++ where the keyword **const** is used.

--

A class defined as **final** is useful to restrict inheritance if not required.

Also **published** i.e. external to java bean and **local** scope i.e. just in the block.

## 8.Reverse engineering of source code

- Java decompilers
  - **Javap -c** for string passwords
  - Mocha – for source code
  - <http://jode.sourceforge.net/>
  - JAD – modern C based version
- Obfuscators
  - Klassmaster
  - Jshrink
  - RetroGuard for Java

Java Security Top 10 - © 2008 SANS

Compiled Java byte code can be decompiled back to source code.

Javap -c does a partial job and is included in the Java JDK.

Mocha was the first full decompiler and was written by Mr Van Vliet who decided to keep the source code to his application closed.

But since Mocha was written in Java surely folks could use Mocha on Mocha to find the Java source?

Mr Van Vliet was too clever for this as he had also written the first Java obfuscator which was not released at all in closed or open source.

One problem. Mr Van Vliet unfortunately died so the source to either Mocha or the obfuscator could not be built upon by another developer.

Mocha is still available from this URL <http://www.brouhaha.com/~eric/software/mocha/>

A new tool was written to decompile Java source code called JAD which is a C based replacement and is also closed source.

I hope it's Author Pavel Kouznetsov is feeling fit and well ☺

The application is available at this URL

Pavel Kouznetsov <http://www.kpdus.com/jad.html>

There is also Jode from sourceforge but JAD is most commonly used.

Note not all JVMs can be decompiled by JAD e.g. Blackberry's JVM is not.

There are a number of commercial Java obfuscators now such as Klassmaster, Jshrink and also RetroGuard which is free for academic use.



## 7. Password Management

```
public void testStaticCredentials() throws SQLException {  
    MockControl dsControl =  
MockControl.createControl(DataSource.class);  
    DataSource ds = (DataSource) dsControl.getMock();  
    MockControl conControl =  
MockControl.createControl(Connection.class);  
    Connection con = (Connection) conControl.getMock();  
    ds.getConnection("user", "pw");  
}
```

- `javap -c` can be used to access the disassembled byte code to read the password.
- Part of the JDK so no need to upload.

Java Security Top 10 - © 2008 SANS

Clear text password in source code.

The password should be further obfuscated or encrypted or preferably stored elsewhere so that an attacker cannot access it from this point.

`javap -c` can be used to access the disassembled byte code which will allow an attacker to read the password and as we have said before it is already on the server so no need to upload.

Passwords should not be stored in plaintext whilst the drive is at rest, though they often are.

## 6. Session management

- Jetty web server used `Java.util.random` for session ids
- Predictable ID, Chris Anley my ex-colleague reported it
- Fixed very quickly and published after [www.ngssoftware.com/research/papers/Randomness.pdf](http://www.ngssoftware.com/research/papers/Randomness.pdf)
- Lesson: Never use `Java.util.random` for crypto
- Always use `java.security.SecureRandom`
- Search Krugle, googlecode for Software using `Java.util.random`

Java Security Top 10 - © 2008 SANS

A vulnerability found in Java applications by Chris Anley was within the Jetty webservice which used `java.util.random`.

This package creates predictable numbers i.e. not random.

In this case those non-random numbers were used as session identifiers on a web server called Jetty which meant that when predicted a user's session could be taken over by an attacker. Jetty is used by many including the Apache Geronimo project. Fixed very quickly.

The way to secure against this is to update the Jetty Webserver.

Also never use `java.util.random` for crypto always use **`java.security.SecureRandom`**

This URL has more details.

<http://www.securityfocus.com/archive/1/archive/1/459164/100/0/threaded>

## 5. XSS

```
if (cookies != null) {
for (int i = 0; i < cookies.length; i++) {
out.println(cookies[i].getName() + "[" +
cookies[i].getValue() + "]);
}
}
```

- This code is printed out to a web page using input from the user which means that it may be vulnerable to XSS.
- A more classic example would be:

```
http://www.javanuke.org/user.jsp?op=usrinfo
&unam=&ltscript&gtalert(document.cookie);
</script>
```

Java Security Top 10 - © 2008 SANS

This code is printed out to a web page using input from the user which means that it is vulnerable to XSS.

If an attacker could overwrite a user's cookie then they could get the user to execute JavaScript of their choosing via the cookie name parameter which is published in the web page.

A clearer example of XSS is this example

```
http://www.javanuke.org/user.jsp?op=userinfo&uname=&ltscript&gtalert(document.cookie); </script>
```

Note that the less than and greater than signs have been replaced by their HTML equivalents in order to bypass input validation.

## 4. SQL Injection

```
for (int i = 0; i < results.length; i++) {  
    // BREAKS ON ' in name  
    int dbCount = helper.queryForInt("SELECT  
COUNT(FORENAME) FROM CUSTMR WHERE FORENAME='" +  
results[i] + "'", (Object[]) null);  
    assertTrue("found in db", dbCount == 1);  
}
```

- If the user can control "results" array they could input SQL into the Dynamic SQL statement.
- The mitigation is to use a prepared statement.

Java Security Top 10 - © 2008 SANS

In this example of SQL injection found in the SPRING Framework, if the user can control "results" array they could input SQL into the Dynamic SQL statement and run SQL commands as the application.

A mitigation is to use a prepared statement.

[http://www.giac.org/certified\\_professionals/practicals/gsoc/01.php](http://www.giac.org/certified_professionals/practicals/gsoc/01.php) (page 27 for prepared statement example).

### 3. Buffer Overflows

- Buffer overflows are generally specific to the version of the JRE
- How to find the version?
- Many JREs on the same box often  
`java -version` and `javac -version`  
`echo $PATH $CLASSPATH`  
OR  
`echo %PATH% %CLASSPATH%`

Java Security Top 10 - © 2008 SANS

Remember the bugtraq examples at the beginning. 2 buffer overflows in the Java platform in one month.

Buffer overflows are generally specific to the version of the JRE and not so much a source code review issue.

Either the JRE is vulnerable or it is not. Part of knowing if Java is secure or not is updating the platform.

But how to find the effective version of the JRE that you are running especially when it is common to have multiple JREs on the same machine?

**Audit Tip** Know which version of Java the system is actually using as follows.

Echo `$PATH $CLASSPATH` on UNIX or on `%PATH%` and `%CLASSPATH%` on Windows to find the versions that may be used.

Also `java -version` and `javac -version`.

Note the versions may differ due to numerous JVMs on the same box.

## 2. Directory traversal

- `../..` directory traversal issues are common in Java.
- John Heasman of NGS Software discovered that Java Web Start allows the local Java security policy to be overwritten by an applets cookie using `../..` notation.
- The security privileges of a Java applet are controlled using a file called **java.policy** on the client workstation.
- The location of this file is by default `%USERPROFILE%\java.policy`
- Once the policy can be overwritten by the cookie all security within the JVM can be overridden.
- <http://www.ngssoftware.com/advisories/high-risk-vulnerability-in-java-web-start/>
- One way to deal with this would be to integrity check the policy files on the workstation.
- There is another, more secure way... Can you guess?

Java Security Top 10 - © 2008 SANS

`../..` directory traversal issues are common in Java.

The security privileges of a Java applet are controlled using the Java Policy tool which edits a file called `java.policy` on the client workstation.

The location of this file is by default `%USERPROFILE%\java.policy`

John Heasman of NGSSoftware discovered that Java Web Start allows the Java security policy to be overwritten by its own cookie using `../..` notation.

Once the policy can be overwritten all security within the JVM can be overridden.

<http://www.ngssoftware.com/advisories/high-risk-vulnerability-in-java-web-start/>

One way to deal with this would be to integrity check the policy files on the workstation.

There is another way..... Can you guess?

# 1. Applets

- Don't use Applets at all.
- Applets are virtually impossible to secure.
- For instance the "**same origin**" policy that is applied to applets can be bypassed trivially using a single keyword.
- Similar to flash vulnerabilities around currently.
- Applets are rarely used now partly due to their inherent insecurity.

Java Security Top 10 - © 2008 SANS

Answer ...Don't use Applets at all.

Applets are virtually impossible to secure.

For instance the "**same origin**" policy that is applied to applets can be bypassed trivially using a single word.

Similar to flash vulnerabilities around currently.

Applets are rarely used now partly due to their inherent insecurity.

### Security as part of the development methodology

- *Waterfall* is the traditional model
  1. *Requirements specification*
  2. *Design*
  3. *Construction (implementation or coding)*
  4. *Integration*
  5. *Testing and debugging* **≠ SECURITY GOES HERE.**
  6. *Installation*
  7. *Maintenance*
- [http://en.wikipedia.org/wiki/Waterfall\\_model](http://en.wikipedia.org/wiki/Waterfall_model)
- Generally a long one-hit process
- Straightforward for security

Java Security Top 10 - © 2008 SANS

### ***Methodologies***

Whilst teaching the Java Security Auditing course many of the students asked how the methodology of the Audit could fit in with the Software development process.

This is pertinent as the results from the audit will have to feed back into the development process at some stage.

How does it all fit together?

The classic development methodology is the Waterfall model.

The following phases are followed in Sequential order:

*Requirements specification*

*Design*

*Construction (AKA implementation or coding)*

*Integration*

*Testing and debugging (SO THIS IS WHERE SECURITY GOES).*

*Installation*

*Maintenance*

[http://en.wikipedia.org/wiki/Waterfall\\_model](http://en.wikipedia.org/wiki/Waterfall_model)

Generally a long one-hit process

Straightforward for security

**BUT THIS METHODOLOGY IS BEING REPLACE BY AGILE DEVELOPMENT  
METHODOLOGIES.**



## Modern Agile Development Process

[http://en.wikipedia.org/wiki/Agile\\_software\\_development](http://en.wikipedia.org/wiki/Agile_software_development)

- Continuous fast delivery of software
- Feature driven
- Late changes in requirements are welcomed
- Regular adaptation to changing circumstances
- Projects are built around motivated individuals, **who should be trusted**
- Self-organizing teams --- CHANGE
- Agile process more difficult to secure as continually changing

Java Security Top 10 - © 2008 SANS

[http://en.wikipedia.org/wiki/Agile\\_software\\_development](http://en.wikipedia.org/wiki/Agile_software_development)

Continuous fast delivery of software

Feature driven

Late changes in requirements are welcomed

Regular adaptation to changing circumstances

Projects are built around motivated individuals, **who should be trusted**

Self-organizing teams --- CHANGE

Agile process more difficult to secure as continually changing

## **BIG QUESTION?**

---

- **How to integrate the security auditing process with Agile software development process?**
- And watch out for the Agile-Hackers at GNUCitizen  
<http://www.gnucitizen.org/blog/agile-hacking/>
- A collaborative tool required.
- Fortify manager is an example.

Java Security Top 10 - © 2008 SANS

### **BIG QUESTION?**

How to integrate the software development process with the code auditing process?....

Could call the Agile-Hackers at GNUCitizen <http://www.gnucitizen.org/blog/agile-hacking/> ?

...In an agile environment a collaborative tool would be required that can scan, report, correlate and alert over time in synchronization with the software development cycle  
i.e. Audit and Development integrated into the same process.

An example of such a tool is Fortify Manager.



The Fortify Manager application sits between the security and the development team integrating security into the development process.

Testing occurs at the same time as development in short release cycles and can be automated to carry out regular scanning with correlated results which show how the code base is being improved over time.

It features:

Web-based user interface which is accessible to all stakeholders and can be integrated with the software build.

Comparison view of summary metrics and trends across all the projects in a centralized repository.

Policy Manager for application security PCI etc with flexible alerts which can be captured on the dashboard as well as via e-mail (plus pager and mobile phone).

Fortify Manager is not in the demonstration version included with the Static Analysis book by Brian Chess and Jacob West.

Have to buy the Enterprise version of Fortify.

<http://www.amazon.com/Programming-Analysis-Addison-Wesley-Software-Security/dp/0321424778>

In addition to Fortify Manager, Fortify Team Server enables lower level developer collaboration at code level to fix the issues brought up by Fortify SCA.

Both these products enable security audit and remediation to be carried out alongside the development process.

## Conclusions

---

- Java is certainly not secure by default
- Current code bases need to be audited for security vulnerabilities
- Concurrent security auditing process should be put in place with dev
- <http://www.javasecurity.net>
- "Finally" ~ Questions?

Java Security Top 10 - © 2008 SANS

### *Conclusion*

Java is certainly not secure by default despite the attitudes of early developers, therefore current code bases should be audited for security vulnerabilities. Additionally a concurrent auditing process should be put in place to shadow the development process so that the security of the application can be measured over time and compared to other software projects for benchmarking purposes so that best practice can be adopted across all applications.