Secure Java Programming

Best Practices, Tools and Techniques

AKA “Don’t Be A Pwned n00b”
Today you gain **L33t Haxxor Skilllz!!!!1111**

- Recognize the most common and dangerous Java coding mistakes that make applications vulnerable
- See L33t Exploits in action
- Refactor code to take defensive measures
- Review what the Java platform has to offer
- Know where the online resources are
- Properly label and categorize vulnerabilities
Fireside Chat time, with Uncle Scott
Agenda

• Java platform security
• Online Resources
• Most common vulnerabilities
• Leet Skillz k\Vlz
  – **See** Java code
  – **Find** the vulnerability
  – **Exploit** the weakness
  – **Harden** the code against haxxors
• And remember, as with all Leet Skillz…
  – Winners get riches
  – Lus0rs get Pwned
  – So pay attention!!!
• Java platform security
• Online Resources
• Most common vulnerabilities
• Leet Skillz kWlz
The Seven “Why’s” of Java Security

• “Y” is Java Secure? (I know, kinda tacky, what can I say – I write code for a living!)
  1. **Immunity** - from malware/viruses/malicious code
  2. **Privacy** – sensitive data in the memory or local storage of a machine hosting a Java program is protected
  3. **Identity** – The “Who” and “From Where”:
     • principals tied to the running of the program are known
     • the source of a program is known, and tampering is prevented
  1. **Secrecy** – data in transit or at rest can be encrypted
  2. **History** – actions that may or may not be permissible are auditable
  3. **Policy** – Rules regarding what code can do, when loaded from a particular “Code Source” and run by a particular “Principal”, are enforced
  4. **Safety** – Runtime adheres to strict rules
Java is Secure Because…

- Security features are baked in
- It provides cryptographic building blocks for message digests, key management, digital signatures, and symmetric and asymmetric encryption
- “Sandbox” runtime policies based on:
  - From where code was loaded
  - By whom the code was digitally signed
  - What principal is running the code
• Language and Runtime Features
  – Type-safe language
  – Magic memory management and garbage collection
  – Automatic range-checking on arrays
  – Class loading is secure and bytecode is verified
  – Access modifiers to help you protect your classes, methods and fields
• Java Programs cannot see memory they don’t “own”
• Applets are unable to see each other
• Java byte-code verifier is key to all of this protection
• Java language has “access levels” like C++ (public/package/protected/private)
  – But… C++ memory is WIDE OPEN
  – Java memory is (for the most part) NOT
  – I will demonstrate later on when we can break Java’s access level protection
Baked In Security – The Rules

- Access modifiers are enforced
- Java programs cannot access arbitrary memory locations
- `final` variables, once initialized, may never be changed
- `final` classes may not be extended
- `final` members may not be overridden
- Variables must be initialized before use
- Array bounds are never violated
- Objects cannot be arbitrarily cast
Baked In – When are Rules Enforced?

• Compile Time
• Classloading ("linking")
• Runtime

Note: Some Java 2 runtime rules are enforced when a "SecurityManager" is present in the JVM. How and what rules are enforced is defined by a "Policy".
Baked In – Bytecode Verifier

- The Verifier can enforce these checks:
  - Class file has correct format
    - Magic Number
    - Minor Version
    - Major Version, etc...
  - Final classes are not subclassed
  - Final methods are not overridden
  - Every class has a single superclass
    - Well almost… what is the exception?
      - java.lang.Object
  - No illegal data conversion of primitive types
  - No illegal data conversion of objects

CAFE BABE
• **LeeT Question – Can you overflow a Java thread’s stack?**
  – Trick question! There are TWO Java stacks for each thread!
  – The **Data Stack** is loaded with method frames, which contain local variables and other data related to a method call
  – The **Operand Stack** hold the values the bytecode operates on
  – The Data Stack may be overflowed, Why?
    • Answer: Recursion
  – The Operand Stack is protected by the verifier against underflow and overflow
Baked In – Runtime Enforcement

- Array bounds
- Object Casting
Are there exceptions to the access modifier rules?
Yes!
Exception #1: Object Serialization
Exception #2: Reflection
Exception #3: You will find out later in this presentation!
Object Serialization sees your PRIVATES!
All your variables belongs to us?
Mitigation
- Don’t extend `java.io.Serializable`
- Implement `readObject` and `writeObject` as `final` methods that throw `IOException`
- If serialize you must
  - use `transient`
  - or use `java.io.Externalizeable` plus Encryption
Reflection – another Exception to the Rules

- Reflection, using `AccessibleObject.setAccessible()`
- Only works if either:
  - No SecurityManager is running in the JVM
  - Or… the current SecurityManager allows access modifiers
- For example, if this has been done:
  - `System.setSecurityManager( new SecurityManager() );`
  - This will result when you try to bypass:
    
    ```
    Exception in thread "main" java.security.AccessControlException: access denied
    (java.lang.reflect.ReflectPermission suppressAccessChecks)
    at java.security.AccessControlContext.checkPermission(AccessControlContext.java:264)
    at java.security.AccessController.checkPermission(AccessController.java:427)
    at java.lang.SecurityManager.checkPermission(SecurityManager.java:532)
    at java.lang.reflect.AccessibleObject.setAccessible(AccessibleObject.java:107)
    ```
- Also see Javadocs for `SecurityManager.checkMemberAccess()`
GoldMine mine = new GoldMine();
Class proggyClass = mine.getClass();
final Field[] faFields = proggyClass.getDeclaredFields();
for( Field fld : faFields ) {
    if( fld.getName().equals( PRIVATE_VARIABLE_NAME ) ) {
        try {
            fld.setAccessible(true);
            fld.set( mine, FOOLS_GOLD );
        } catch (IllegalArgumentException e) {
            System.err.printf( "An IllegalArgumentException has occurred trying to set the value of [%s]%n", PRIVATE_VARIABLE_NAME );
        } catch (IllegalAccessException e) {
            System.err.printf( "An IllegalAccessException has occurred trying to set the value of [%s]%n", PRIVATE_VARIABLE_NAME );
        }
    }
}
• The Java language and runtime environment is Chock full o’ Security
• Programs cannot tamper with memory
• Runtime view of memory and internal state of objects is clearly defined
• This helps protects a host machine from malicious code
2. Building Blocks for Cryptography

• APIs
  – Cryptography and public key infrastructure (PKI)
  – Authentication and access control interfaces
  – Specifically for code signing:
    • Message Digests
    • Cryptographic keys
    • Digital Signatures
    • Encryption Engines
Java 5 Security Enhancements

- JSSE (Java Secure Socket Extension) enhancements
  - Java implementation of SSL and TLS protocols
- JCE (Java Cryptography Extension) enhancements
  - new ciphers and algorithms
  - integration with SmartCards and crypto accelerator ("PKCS#11")
  - API improvements
- Signature Timestamp Support
- PKI Enhancements
- SASL Support (Simple Authentication and Security Layer)
- Java Kerberos enhancements
Java 6 Security Enhancements

- An XML Digital Signature implementation (JSR 105)
- Smart Card API’s (JSR 268)
- Additional Encryption Algorithms
- Expanded options, loosened restrictions with tools
- API improvements
- LDAP support built in to JAAS
- Support for Microsoft’s Crypto API
3. The Sandbox

- SecurityManager
- AccessController
- Policies
- Code Signing
The Java Sandbox

• Typically, “Sandbox” describes the runtime environment of Java applets

• True or False – Java command line applications run within a sandbox too
  – Answer: It Depends
  – `java -Djava.security.manager`
  – Or programmatically:
    ```java
    // set a SecurityManager for the JVM
    System.setSecurityManager(new SecurityManager());
    ```
The Sandbox Police

- Security Manager
- Access Manager
- Classloaders
- Policies

- These all work together to determine what rules to apply to what classes, and make sure the rules aren’t broken
The Security Manager is the original face of the sandbox police. It is called to determine what a Java program can and cannot do:

- File I/O
- Network Access
- Opening toplevel windows
The Access Manager was added in Java 1.2
- Allows easier and fine grained permission granting to specific classes
- Security Manager has remained for backwards compatibility
- Access Controller leverages these constructs:
  - Code Sources (URL and Keys from signed Jar if any)
  - Permissions (Type, Name, and Action)
  - Policies (1 Policy class per JVM)
  - Together these made Protection Domains
- As of Java 1.4, the concept of a Principal was added
public class SecurityManagerExample {

    public static void main(String[] args) {
        SecurityManager sm = System.getSecurityManager();
        if (sm != null) {
            System.out.printf("There is a SecurityManager and it is %s", sm.getClass().getCanonicalName());
        } else {
            System.out.println("No SecurityManager present at first!");
        }

        Properties props = System.getProperties();
        System.out.println("Got System properties!");

        // set a SecurityManager for the JVM
        System.setSecurityManager(new SecurityManager());
        sm = System.getSecurityManager();
        if (sm != null) {
            System.out.println("Now there is a SecurityManager and it is "+ sm.getClass().getCanonicalName());
        } else {
            System.out.println("No SecurityManager present!");
        }

        try {
            props = System.getProperties();
            System.out.println("Got System properties!");
        } catch (Exception e) {
            System.out.println("Access denied to System properties!");
            e.printStackTrace();
            // dang it all, let's get rid of the Security Manager
            System.setSecurityManager(null);
        }
    }
}
Agenda

• Java platform security
• Online Resources
• Most common vulnerabilities
• Leet Skillz k\W\lz
• OWASP for short, found online at http://www.owasp.org

• Open Community focused on security - online resources covering web application security

• Resource of interest:
  – Top 10
  – Guide to Building Secure Web Applications
  – Java Project
• Drawing on numerous sources, CWE is an effort to standardize how “software weaknesses” are identified

• Think “latin names” for security vulnerabilities

• CWE dictionary can be viewed in various ways
  – OWASP Top Ten Slice
  – Java Language List
Agenda

• Java platform security
• Online Resources
• Most common vulnerabilities
• Leet Skillz kWlz
OWASP Top Ten of 2007

1. Cross Site Scripting (XSS)
2. Injection Flaws
3. Malicious File Execution
4. Insecure Direct Object Reference
5. Cross Site Request Forgery (CSRF)
6. Information Leakage and Improper Error Handling
7. Broken Authentication and Session Management
8. Insecure Cryptographic Storage
9. Insecure Communications
10. Failure to Restrict URL Access

Cross reference to CWE Weaknesses in Top Ten List
1. Buffer overflow using long strings of “A” characters in:
   a. username/password during authentication
   b. file or directory name
   c. arguments to most common features of the product or product class

2. XSS using well-formed SCRIPT tags, especially in the:
   a. username/password of an authentication routine
   b. body, subject, title, or to/from of a message

3. SQL injection using ' in the:
   a. username/password of an authentication routine
   b. “id” or other identifier field
   c. numeric field

4. Remote file inclusion from direct input such as:
   a. include($_GET['dir'] . '/config.inc');

5. Directory traversal using "../.." or "/a/b/c" in “GET” or “SEND” commands of frequently-used file sharing functionality, e.g. a GET in a web/FTP server, or a send-file command in a chat client

6. World-writable critical files:
   a. Executables
   b. Libraries
   c. Configuration files

7. Direct requests of administrator scripts

8. Grow-your-own crypto

9. Authentication bypass using "authenticated=1" cookie/form field

10. Turtle race condition - symlink

11. Privilege escalation launching "help" (Windows)

12. Hard-coded or undocumented account/password

13. Unchecked length/width/height/size values passed to malloc()/calloc()
• Java platform security
• Online Resources
• Most common vulnerabilities
• Leet Skillz k\W/Iz
• It is essential that Java professionals become one with Secure coding best practices
• The following slides demonstrates “sploits”
• Each is followed by review and mitigation recommendations
• Let’s Pwn the Suzzor Haxxor!
Leet Skillz kWl/z
Splot #1
SPLOIT #1 – EXAMPLE A

\<% String eid = request.getParameter("eid"); %>\n
Employee ID: \<%= eid %\>

• http://localhost:8080/ExampleOne.jsp?eid=123
• XSS Injection

• eid=<script>javascript:alert('This pop-up confirms vulnerability to Cross-Site Scripting via URL Tampering')</script>
<%...

Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT *
   FROM emp WHERE id= " + eid);
if (rs != null) {
   rs.next();
   final String name = rs.getString("name");
}
%

Employee Name: <%= name %>
CWE ID 79

Sending user generated and/or un-validated data to a web browser is dangerous

Malicious content may be injected and stored in a database or other “trusted” source, then later executed by a victim’s browser

Malicious content might be created by an attacker then executed by a user
  - For example, a malicious URL might be constructed and emailed to a victim
  - When the victim clicks the link, the URL exploits a vulnerable web application, “reflecting” injected code back to the victim’s browser where it is executed with the context of the vulnerable web application
  - This last attack is known as “Reflected XSS”

Fundamental issue is that some characters in HTML are “special” in certain contexts, and can be used to introduce code that will be executed by the user’s browser

Content might be:
  - JavaScript
  - HTML
  - Flash
  - Etc…
Study Cert Advisory CA-2000-02 and Cert’s Understanding Malicious Content Mitigation for

Then... Get Tactical:

• Explicitly set character encoding for all web pages

• Identify special characters
  – Encode special characters in dynamic output
  – Filter dynamic content on in and out
<%@ page contentType="text/html; charset=ISO-8859-1" language="java" %>
<html>
<head><title>Example One</title></head>
<body>
<% String eid = request.getParameter("eid"); %>
Employee ID: <%= eid %>
</body>
</html>
/**
 * This simple pattern is used to do a search and replace below
 */
static final Pattern m_patternLessThan = Pattern.compile( "<" );

/**
 * This simple pattern is used to do a search and replace below
 */
static final Pattern m_patternGreaterThan = Pattern.compile( ">");

/**
 * This function will cleanse a string for usage in an HTML block level element.
 * This is to prevent cross-site scripting - we make sure the dynamic content in
 * this attribute does not include any greater or less than characters.
 * Note that Pattern.matcher has some very minimal synchronization internally.
 * @param p_sElementText The text you want to cleanse
 * @return The cleansed text
 */

public static String cleanseHtmlBlockLevelElement( final String p_sElementText ) {
    if( p_sElementText == null ) return "";
    return m_patternGreaterThan.matcher( m_patternLessThan.matcher( p_sElementText ).replaceAll( "&lt;" )
        ).replaceAll( "&gt;" );
}
<html>
<head><title>Example One</title></head>
<body>
<% String eid = request.getParameter("eid"); %>
Employee ID: <%= Util.cleanseHtmlBlockLevelElement(eid) %>
</body>
</html>

• All your boxxen belongZ to uz?
• What could we do to improve the above?
• You will need to cleanse more than just the above (block level elements). Watch for other forms of dynamic content such as:
  – HTML Attributes enclosed in double quotes
  – HTML Attributes enclosed in single quotes
  – HTML Attributes enclosed in no quotes
  – CDATA

• Also note that XML is NOT SAFE – web service endpoints can be used in XSS attacks.

• Here is a convoluted example active on Google’s widget site.

• It pulls in XML, similar to a simple web service call from here. This XML is reflected back to the user.
Leet Skillz kWlz
Sploit #2
String home = System.getProperty("APP HOME");
String lib = home + LIBNAME;
java.lang.Runtime.getRuntime().load(lib);
System.loadLibrary("library.dll");
• CWE ID 114
Leet Skillz k\W\lz
Sploit #3
public class ShadyServlet extends HttpServlet {

private String m_sGroupLevel;

protected void doGet(final HttpServletRequest p_request, final HttpServletResponse p_response) throws IOException {
    final String name = p_request.getParameter("name");
    _SetAdminLevel(name);
    p_response.getOutputStream().println("<h1>You are Group Level "+m_sGroupLevel+"</h1>);
}

private void _SetAdminLevel(final String p_sName) {
    if (p_sName.equals("Admin")) {
        m_sGroupLevel = p_sName;
    }
}

• If we do this, all is well:
  http://localhost:8080/ShadyServlet?name=Admin

• If we do this, bad things happen:
  http://localhost:8080/ShadyServlet

You are Group Level Admin

HTTP Status 500 -

java.lang.NullPointerException
  at org.fresers.util.ShadyServlet._SetAdminLevel(ShadyServlet.java:23)
  at org.fresers.util.ShadyServlet.doGet(ShadyServlet.java:23)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:690)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:803)

Note: The full stack trace of the root cause is available in the Apache Tomcat/5.5.23 logs.
protected void doPost (HttpServletRequest req,
HttpServletResponse res)
throws IOException {
    String ip = req.getRemoteAddr();
    InetAddress addr = InetAddress.getByName(ip);
    ...
    out.println("hello " + addr.getHostName());
}
• CWE ID 431
• The default behavior of a servlet in most containers is to return a stacktrace and additional debugging information when an unhandled exception is thrown.
• Information revealed could include:
  – Stacktrace, showing internals of your application
  – Database type
  – App server version
  – SQL strings
• Always catch Throwable in your top-level Servlet methods
public class LessShadyServlet extends HttpServlet {
    private String m_sGroupLevel;
    protected void doGet(final HttpServletRequest p_request,
                         final HttpServletResponse p_response)
                       throws IOException {
        try {
            final String name = p_request.getParameter("name");
            _SetAdminLevel(name);
            p_response.getOutputStream().println("<h1>You are Group Level "+ m_sGroupLevel + "</h1>" anunciadas de un programa no fiable, y los detalles son NUN YA BIDNESS!" });
        } catch( Throwable t ) {
            p_response.getOutputStream().println("<h1>Error occurred, and the details are NUN YA BIDNESS!</h1>" );
        }
    }
    private void _SetAdminLevel(final String p_sName)
    {
        if (p_sName.equals("Admin"))
            m_sGroupLevel = p_sName;
    }
}
Leet Skillz kWLlz
Sploit #4
public class GroupAdminApplet extends Applet {
    public final static String[] GROUPS = {
        "ReadOnly", "Managers"};
    // some functions follow here…
}
public final class GroupAdminApplet extends Applet {

    /**
     * This is used by the HM Engine when it needs to return an empty rowid set. It is public
     * because it is immutable and could be used for comparison operations if you wanted to
     * check the return value of rowidSearch to see if no rowids were returned.
     */

    final public static int[] NO_ROWIDS = new int[0];

    ...

}
CWE ID 582

Java's "final" keyword does not protect the contents of arrays

If you are a solid programmer with mad Object Oriented skillz, you will almost always be setting your mutable non-final variables as private

Those who don’t might have to deal with… THIS GUY
public class GroupAdminApplet extends Applet {

    private final static String[] GROUPS = {
        "ReadOnly",
        "Managers"
    };

    public String[] getGroups() {
        return GROUPS.clone();
    }
}
Leet Skillz kWlz
Sploit #5
public final class GoldMine extends Applet {
    private String m_sSecretRawMaterials = "ore";
    public GoldMine() {
        new Miner().doWork();
    }

    private final class Miner {
        final void doWork() {
            if (m_sSecretRawMaterials.equals("ore")) {
                m_sSecretRawMaterials = "gold";
            }
        }
    }

    public String getPayday() {
        if (m_sSecretRawMaterials.equals("gold"))
            return "rich";
        else
            return "poor";
    }
}
• CWE ID 492
• Avoid using inner classes!
• Use javap to see for yourself
package org.frasers.util;
import java.applet.Applet;
public final class GoldMine extends Applet {
    private String m_sSecretRawMaterials = "ore";
    public GoldMine() {
        new Miner().doWork();
    }
}
private final class Miner {
    final void doWork() {
        if (m_sSecretRawMaterials.equals("ore")) {
            m_sSecretRawMaterials = "gold";
        }
    }
}
public String getPayday() {
    if (m_sSecretRawMaterials.equals("gold")) {
        return "rich";
    } else {
        return "poor";
    }
}

Run This:
javap -private -c GoldMine
Compiled from "GoldMine.java"

public final class org.frasers.util.GoldMine extends java.applet.Applet{
private java.lang.String m_sSecretRawMaterials;
public org.frasers.util.GoldMine();

Code:
0:  aload_0
1:  invokespecial  #2; //Method java/applet/Applet."<init>"():V
4:  aload_0
5:  ldc  #3; //String ore
7:  putfield  #1; //Field m_sSecretRawMaterials:Ljava/lang/String;
10:  new  #4; //class org/frasers/util/GoldMine$Miner
13:  dup
14:  aload_0
15:  aconst_null
16:  invokespecial  #5; //Method org/frasers/util/GoldMine$Miner."<init>":
      (Lorg/frasers/util/GoldMine;Lorg/frasers/util/GoldMine$1;)V
19:  invokevirtual  #6; //Method org/frasers/util/GoldMine$Miner.doWork():V
22:  return
public java.lang.String getPayday();

Code:

0:  aload_0
1:  getfield #1;  //Field 
m_sSecretRawMaterials:Ljava/lang/String;
4:  ldc  #7;    //String gold
6:  invokevirtual #8;  //Method java/lang/String.equals:
(Ljava/lang/Object;)Z
9:  ifeq  15
12: ldc  #9;    //String rich
14: areturn
15: ldc  #10;   //String poor
17: areturn
Sploit #5 - GoldMine Bytecode – WTF?

```java
static java.lang.String access$100(org.frasers.util.GoldMine);
Code:
  0:  aload_0
  1:  getfield #1; //Field m_sSecretRawMaterials:Ljava/lang/String;
  4:  areturn

static java.lang.String access$102(org.frasers.util.GoldMine,
     java.lang.String);
Code:
  0:  aload_0
  1:  aload_1
  2:  dup_x1
  3:  putfield #1; //Field m_sSecretRawMaterials:Ljava/lang/String;
  6:  areturn
}
```
static String access$100( GoldMine p_mine ) {
    return p_mine.m_sSecretRawMaterials;
}

static String access$102( GoldMine p_mine, String p_sNewRawMaterials ) {
    return p_mine.m_sSecretRawMaterials = p_sNewRawMaterials;
}
• **javac** added two accessors to GoldMine that you didn’t code!
• These two accessors were added so that the inner class Miner can “get” and “set” the private SecretRawMaterials variable
• The first allows a caller to “get” the private SecretRawMaterials
• The second allows a caller to “set” the private SecretRawMaterials
• No big deal right…
• Hang on! Those accessors have package local access! Why?
• Because the Miner inner class, once compiled, is technically a SEPARATE class in the package. There is NO SUCH THING as an “inner class” at the JVM level.
This line in the Miner inner class:
  - `if (m_sSecretRawMaterials.equals("ore")) {

Becomes this in Java bytecode:
  - `invokestatic `\#4`; //Method
    org/frasers/util/GoldMine.access$100:
    (Lorg/frasers/util/GoldMine;)Ljava/lang/String;
  - `ldc `\#5`; //String ore
  - `invokevirtual `\#6`; //Method
    java/lang/String.equals:(Ljava/lang/Object;)Z

That is MESSED UP – your inner class is not accessing the variable directly – it is using a “getter” function that YOU didn’t write!
• This line in the Miner inner class:
  – m_sSecretRawMaterials = "gold";

• Becomes this in Java bytecode:
  – ldc #7; //String gold
  – invokestatic #8; //Method
    org/frasers/util/GoldMine.access$102:
      (Lorg/frasers/util/GoldMine;Ljava/lang/String;)Ljava/lang/String;

• Again, That is MESSED UP! In this case your inner class is not accessing the variable directly – it is using a “setter” function that YOU didn’t write!
Sploit #5 – Time to dig for some Gold!

• Problem – the accessor functions javac added do not exist at compile time
• So… you can’t write code to use the accessors – you will get a compile time error.
• What do you do?
• Option 1: Use Reflection!
• Option 2: Compile against a “stub” class with fake accessors
• Option 3: Generate Byte Code
public class BreakIntoGoldMine {

    public static void main(String[] argv) throws Exception {
        GoldMine mine = new GoldMine();

        Class proggyClass = mine.getClass();

        // let's see what the gold mine contains
        System.out.println("This Gold Mine could make you: 
            "+ mine.getPayday() + "]");

        // first steal the goods
        Method mtdGetTheGoods = proggyClass.getDeclaredMethod("access$100",
            proggyClass);

        final String sStolenGoods = (String) mtdGetTheGoods.invoke(null, mine);

        // now replace with some fools gold
        Method mtdSetTheGoods = proggyClass.getDeclaredMethod("access$102",
            proggyClass, String.class);

        final String sFoolsGold = "fools gold";
        mtdSetTheGoods.invoke(null, mine, sFoolsGold);

        final String sGoodAfterRobbery = (String) mtdGetTheGoods.invoke(null, mine);

        System.out.println("Using Reflection you stole some: 
            "+ sStolenGoods + "] and replaced it with 
            "+ sGoodAfterRobbery + "]");

        System.out.println("Now the GoldMine will make you: 
            "+ mine.getPayday() +"\]");
    }
}
• Create a “fake” GoldMine class with stub accessor functions to compile against
• Write attacking class that uses those accessors to steal the gold
public final class GoldMine extends Applet {
    public GoldMine(){}
    public String getPayday() { return "stub"; }
    static String access$100( GoldMine p_mine )
    { return "stub"; }
    static String access$102( GoldMine p_mine, String p_sNewRawMaterials )
    { return "stub"; }
}
$ java org.frasers.util.BreakIntoStubGoldMine
This Gold Mine could make you: [rich]
You stole some: [gold] and replaced it with [fools gold]
Now the Gold Mine will make you: [poor]
Leet Skillz kIz BONUS QUESTION!

What happens if we add our homemade accessor stubs to the original GoldMine class?

- Compile error!
  - the symbol access$100 conflicts with a compiler-synthesized symbol
  - the symbol access$102 conflicts with a compiler-synthesized symbol

If you make the SecretRawMaterials "public", or comment out the lines in the Miner Inner Class that access and set the variable, the compile errors go away!
java.lang.SecurityException: class "org.apache.commons.digester.Substitutor"'s signer information does not match signer information of other classes in the same package

at java.lang.ClassLoader.checkCerts(ClassLoader.java:775)
at java.lang.ClassLoader.preDefineClass(ClassLoader.java:487)
at java.lang.ClassLoader.defineClass(ClassLoader.java:614)
at java.security.SecurityClassLoader.defineClass(SecureClassLoader.java:124)
at java.net.URLClassLoader.defineClass(URLClassLoader.java:260)
at java.net.URLClassLoader.access$100(URLClassLoader.java:56)
at java.net.URLClassLoader$1.run(URLClassLoader.java:195)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:188)
at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:268)
at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:319)

**Sploit #5 – How to Mitigate?**

- Most importantly: Avoid Inner Classes
- Sign your jars
  - Since the accessors are package local, the attacking class must be in the same package
  - Signing your jars will prevent someone lacking your private code-signing key from creating attacking classes in your packages
  - Remember that dynamic scripting languages could circumvent this protection
Leet Skillz kWlz
Splotit #6
final String sUserName = tai.getAuthenticatedUsername(req);

final String sTemplateName = req.getParameter("tempName");

final String sSql = "SELECT * FROM user_templates WHERE owner = "" + sUserName + "" AND temp_name = "" + sTemplateName + "";"

final ResultSet rs = stmt.executeQuery(sSql);
Sploit #6 – SQL Injection

• CWE ID 89
• Attacker often starts by determining the schema of target database
• Injection uses numerous techniques to get the presentation layer to give the attacker information they do not need to know
The most commonly known attack is to use a SQL comment to “break out” of the intended statement and add malicious content.

For example this:

```sql
SELECT * FROM orders WHERE order_id = 'K123DMKWO'
```

Is attacked via “UNION Injection” to become this:

```sql
SELECT * FROM orders WHERE order_id = 'K123DMKWO'
UNION
SELECT table_name, null, null, null FROM INFORMATION_SCHEMA.TABLES
```

Leet Haxxor Question – How do we determine the number of columns needed in the UNION?

- Inject ORDER BY’s until you don’t get errors:

```sql
SELECT * FROM orders WHERE order_id = 'K123DMKWO'
ORDER BY 15 -- '
```
• Line comments

```
SELECT * FROM members WHERE username = 'admin'--'AND password = 'password'
```

• Inline comments

```
DR/***/OP/***inline comment to avoid pattern matching*/important_table
```

• Stacked queries

```
SELECT * FROM users; DROP users
```
• String Concatentation can be used to inject without quotes, and to detect backend type
  – MySQL and SQLServer, CHAR() and CONCAT() can be used
  – Hex codes in MySQL can be SELECTED as Strings
• ASCII and CHAR are effective when needing to convert letters to numbers
  
  SELECT * FROM members WHERE username = 'admin' -- 'AND password = 'password'

• Inline comments
  
  DR/**/OP/*inline comment to avoid pattern matching*/important_table

• Stacked queries
  
  SELECT * FROM users; DROP users –
• Determine columns names using HAVING BY
• Determine column datatypes using functions such as SUM() on SQL Server, CAST() and CONVERT()
• UNION with synthetic results can be used also

  UNION SELECT NULL,NULL,NULL WHERE 1=2  --
  – No error? You have the number of columns right.

  UNION SELECT 1,NULL,NULL WHERE 1=2  --
  – No error? First column is an int.

  UNION SELECT 1,2,NULL WHERE 1=2  --
  – No error? Second column is an int.
Sploit #6 – Example: SQL Server Foibles

- Due to tight integration with operating system, a vulnerable SQL Server box is very attractive to hackers
- Via stored procedures, integration of scripting, ActiveX, etc…
  - Get server version
  - Bulk insert system files to temp tables
  - Write TO files from a query
  - Interact with the Registry
  - Execute command lines
  - Manage services
  - Kill processes
  - Write files to UNC or internal path
• What if the presentation layer doesn’t show the errors?
• This is known as “Blind SQL Injection”
  – “Normal Blind” – you can at least get an HTTP error code back
  – “Totally Blind” – website is well designed and not giving you any clues
• Major innovation in recent years by the black-hat community
• MEGA LEET QUESTION – What to do?
• If you can at least get HTTP error codes back, use binary logic to determine table names one ASCII char at a time:

```sql
SELECT id, username FROM usertable
WHERE ID = 1 AND
ISNULL(ASCII(SUBSTRING(((SELECT TOP 1 name FROM sysobjects WHERE xtype=0x55 AND name NOT IN
(SELECT TOP 0 name FROM sysobjects WHERE xtype=0x55)),1,1)),0)) > 78 --
```
• If the website being attacked doesn’t even return HTTP error codes, *Time Based Blind SQL Injection* is employed.

• Heavy queries introduce “time” as your True/False response.

• For example, an IF with “benchmark” on MySQL:
  – Are we root?
    ```sql
    IF EXISTS (SELECT * FROM users WHERE username = 'root') BENCHMARK(1000000000, MD5(1))
    ```
  – Does a table exist?
    ```sql
    IF (SELECT * FROM target_table) BENCHMARK(1000000, MD5(1))
    ```
A simple Faces/Java DB website has been created that uses Dynamic SQL for the RowSet:

```java
final String sSQL = "SELECT ALL TRAVEL.TRIP.DEPDATE AS "Departure Date", " + "TRAVEL.TRIP.DEPCITY AS "Departure City", " + "TRAVEL.TRIP.DESTCITY AS "Destination City", " + "TRAVEL.TRIP.TRIPTYPEID AS "Trip Type", " + "TRAVEL.PERSON.NAME AS "Person Name" " + "FROM TRAVEL.TRIP " + "INNER JOIN TRAVEL.PERSON ON TRAVEL.TRIP.PERSONID = TRAVEL.PERSON.PERSONID " + "WHERE TRAVEL.TRIP.PERSONID IN ( SELECT p2.personid FROM person p2 WHERE p2.name LIKE '%" + sName + "%')";
```
Sploit #6 – Workshop time!

Leet Skillz

]\[\&ND2 ON

POWNage

In 10 EZ Steps
• Step 1 - Visit the site:
Step 2 - Try Some Queries, wildcard on it’s own doesn’t work:
• Step 3 – Eventually a name that works:
Sploit #6 – Java Server Faces P0WNage

- Step 4 – The letter “A” is fruitful. It appears there may be a LIKE clause?
• Step 5 – Same result… HMM…
Sploit #6 – Java Server Faces P0WNage

• Step 6 – Wonder if this is at the end of a SQL statement? Bingo!
Sploit #6 – Java Server Faces P0WNage

• Step 7 – Probing a little more… now we are getting somewhere!
• Step 8 – Let’s run a schema report, shall we?
• Start with a SQL query, and then attempt to inject it into the field as a UNION (note how the column names match up with the column names seen in the HTML):

```
SELECT
    CURRENT_DATE AS "Departure Date",
    s.schemaname AS "Departure City",
    t.tablename AS "Destination City",
    1 AS "Trip Type",
    c.columnname AS "Person Name"
FROM sys.systables t
INNER JOIN sys.sysschemas s ON t.schemaid = s.schemaid
INNER JOIN sys.syscolumns c ON t.tableid = c.referenceid
WHERE t.tabletype = 'T'
```
Step 9 – Paste in the long SQL as one string:

```sql
' AND 1=0 ) UNION SELECT CURRENT_DATE AS "Departure Date", s.schemaname AS "Departure City", t.tablename AS "Destination City", 1 AS "Trip Type", c.columnname AS "Person Name" FROM sys.systables t INNER JOIN sys.sysschemas s ON t.schemaid = s.schemaid INNER JOIN sys.syscolumns c ON t.tableid = c.referenceid WHERE t.tabletype = 'T' --
```
Step 10 – P0WNage, Chapter 1:

```sql
%' AND 1=0 ) UNION SELECT
```

<table>
<thead>
<tr>
<th>Person Name</th>
<th>Departure Date</th>
<th>Departure City</th>
<th>Destination City</th>
<th>Trip Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOKINGSTATUS</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>PR/AR</td>
</tr>
<tr>
<td>CARRENTALID</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>PR/AR</td>
</tr>
<tr>
<td>CARTYPE</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>PR/AR</td>
</tr>
<tr>
<td>CITY</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>OFFSITE</td>
</tr>
<tr>
<td>LASTUPDATED</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>SALES</td>
</tr>
<tr>
<td>PICKUPDATE</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>REM MTG</td>
</tr>
<tr>
<td>PROVIDER</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>PR/AR</td>
</tr>
<tr>
<td>RATE</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>PR/AR</td>
</tr>
<tr>
<td>RETURNDATE</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>PR/AR</td>
</tr>
<tr>
<td>TRIPID</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>PR/AR</td>
</tr>
<tr>
<td>AIRLINENAME</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>CARRENTAL</td>
<td>CONF</td>
</tr>
<tr>
<td>ARRARRPORT</td>
<td>Nov 13, 2007</td>
<td>TRAVEL</td>
<td>FLIGHT</td>
<td>BUS DEV</td>
</tr>
</tbody>
</table>
• Don’t do dynamic SQL!
• User PreparedStatements (parameterized queries) instead
• All user-input is EVIL
• Treat database resident content as potentially EVIL
• Use regexp to find problems
• Never send errors to the presentation layer
• Limit your logging
• WebApps must connect as lesser privileged roles
final String sUserName =
    tai.getAuthenticatedUsername(req);
final String sTemplateName =
    req.getParameter("tempName");
final String sSql = "SELECT * FROM user_template
    WHERE owner = ? AND temp_name = ?";
final PreparedStatement stmt =
    conn.prepareStatement(sSql);
stmt.setString(1, sUserName);
stmt.setString(2, sTemplateName);
final ResultSet rs = stmt.executeQuery(sSql);
This is highly complex, and very CPU intensive. It is OK in some situations, but not all.

You could roll your own… but it won’t be perfect, and you will be reinventing wheels big time.

You could use an already existing grammar and parser:
- ANTLR has a grammar for Oracle PL/SQL
- JavaCC has some for PL/SQL

But seriously… do you really want to go there?
• Make sure that the database account executing user-generated dynamic SQL has no DDL permissions, and only SELECT permissions on the appropriate tables
• When possible (i.e. when not within a transaction), use JDBC’s Connection.setReadOnly API
  • http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html#setReadOnly(boolean)
There are some types of applications that require user generated dynamic SQL, for example report writers.

In these cases SQL Injection protection can take one of many approaches:

- Filtering input fields and SQL query
- Parsing SQL query
- Leveraging database security
- Combinations of the above
This is SQL Injection 101, and is ineffective

```java
if (sSql.indexOf("--") != -1)
    sSql = searchAndReplace(sSql, "--", "-"下班;
```
• If the dynamic SQL statement is supposed to be a SELECT, *make sure it is*

• Otherwise, it might contain an anonymous block of PL/SQL (for example) that has an attack
  – BEGIN evil(); END;

• Once a SQL statement starts with SELECT, an attacker cannot add a block of PL/SQL to the end like this:
  – SELECT 1 FROM dual; -- BEGIN evil(); END;
Sploit #6 – Filtering for SELECT

• Best approach is to use a Java Pattern
• Pattern is a compiled regular expression
• It is a factory for Matchers
• Pattern is immutable and therefore thread-safe
• A Matcher is NOT thread-safe
• Regex to find SELECT at the start of a string:
  
  (?si)^[\s]*SELECT.*
/**
 * This pattern will be used to test strings to see if they start with the token SELECT, which is handy for SQL Injection protection. Combine that with a call to Connection.setReadOnly(true) and you will have a pretty tight defense against SQL Injection.
 */

static final Pattern ms_patternSelectStatement = Pattern.compile("(?si)^\s*SELECT.*");

/**
 * simple check to confirm that a String starts with the token SELECT
 * @param p_sSql SQL string that should start with SELECT (case insensitive)
 * @return True if the String starts with a SELECT token
 *
*/

public static boolean stringStartsWithSelect( final String p_sSql ) {
    if( p_sSql == null ) return false;
    return ms_patternSelectStatement.matcher( p_sSql ).matches();
}
Sploit #6 – setReadOnly + filtering is SOLID

If all the following is true:

- You are not in a transaction
- You can parse to confirm the dynamic SQL starts with “SELECT”
- Your RDBMS backend supports Connection.setReadOnly

• You can lock yourself down L33T Style using the following pattern
  - Confirm string starts with SELECT
  - Throw JDBC connection into “read only mode”
  - Execute SQL
  - Take JDBC connection back out of “read only mode”

• If the attacker attempts any DML, a SQL Exception will result
  - java.sql.SQLException: ORA-01456: may not perform insert/delete/update operation inside a READ ONLY transaction
Sploit #6 – setReadOnly + filtering is SOLID

```java
if (Util.stringStartsWithSelect(sSQL)) {
    try {
        // this combined with a check to make sure the SQL starts with SELECT will defend against SQL Injection
        // We will get this SQLException if they attempt something malicious:
        // java.sql.SQLException: ORA-01456: may not perform insert/delete/update operation inside a READ ONLY transaction
        pool_connection.setReadOnly(true);
        sstmt = pool_connection.prepareStatement(sSQL);
        rs = sstmt.executeQuery();
        if (!rs.next()) {
            final String sMsg = "Query returned no data";
            Output.setError(sMsg);
        }
    } finally {
        // make sure we turn this back off
        pool_connection.setReadOnly(false);
    }
} else {
    final String sMsg = "Query must start with 'SELECT'";
    Output.setError(sMsg);
}
```
public void execute(HttpServletRequest request, HttpServletResponse response)
    throws IOException, xDataIncomplete, xGeneralDataError, ServletException {

    final String sFieldId = request.getParameter("field_id");
    response.sendRedirect(Const.TASK_SERVLET_URL + "?actionlink=" + Const.GOTO_FIELD + 
                        
                        Const.FIELD_ID + "=" + sFieldId);  
}
// force the MIME-TYPE and filename for the file download link.
if (request.getParameter("filename") != null) {
    response.setHeader("Content-disposition", "attachment;filename=" + request.getParameter("filename"));
    response.setContentType("application/octet-stream");
    InputStream fis = null;
    try {
        String sSeparator = System.getProperty("file.separator");
        String sPath = pageContext.getServletContext().getRealPath("/");
        File fileIn = new File(sPath + sSeparator + request.getParameter("filename"));
        fis = new FileInputStream(fileIn);
        int bit = 256;
        while (bit >= 0) {

```java
final String sUsername = request.getParameter(USER_NAME_PARAM);
final Cookie cookie = new Cookie("username", sUsername);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```
Sploit #7 – HTTP Response Splitting

• CWE 113

• As per usual, including user generated input without filtering in dynamic output is dangerous

• All of the previous examples would allow an attacker to “split” the HTTP response
  – Example 1 – via an HTTP redirect
  – Example 2 – via HTTP Header splitting
  – Example 3 – via an HTTP Cookie header
• All of the previous examples would allow an attacker to “split” the HTTP response
  – Example 1 – via an HTTP redirect
  – Example 2 – via HTTP Header splitting
  – Example 3 – via an HTTP Cookie header

• The attacker need to get a carriage return (CR) and line feed (LF) into the header
  – Once injected, everything following is under the attackers control, including whole new HTTP responses
  – Note that the characters can be encoded
• For example, you might expect this as your Cookie header:

   HTTP/1.1 200 OK
   (etc...)
   Set-Cookie: username=John Smith

• An attacker could split the response like this:

   HTTP/1.1 200 OK
   (etc...)
   Set-Cookie: username=UROWNED!
   HTTP/1.1 200 OK
   (malicious content here)
• Cross Site Scripting (XSS)
• HTTP Cache poisoning
• Page Hijacking
• Site Defacement
// we need to cleanse sFieldId of CR/LF to avoid HTTP Response Splitting attack

response.sendRedirect( Const.TASK_SERVLET_URL + '?actionlink=' + Const.GOTO_FIELD + '&' + Const.FIELD_ID + '==' + Util.cleanseStringForHttpHeaders( sFieldId ));
/**
 * This represents a CR byte
 */

public static final byte CR = (byte) '';

/**
 * This represents a LF byte
 */

public static final byte LF = (byte) '
';

/**
 * This function will cleanse a string of all CRLF characters, which can be used in HTTP
 * response splitting attacks
 * @param p_sHttpHeaderString String that is going to be included in an HTTP header
 * @return Cleansed string free of any CRLF's
 */

public static String cleanseStringForHttpHeaders(final String p_sHttpHeaderString) {
    if (p_sHttpHeaderString == null) return "";
    return thoroughSearchAndReplace(
        thoroughSearchAndReplace(p_sHttpHeaderString, "\r", ""),
        "\n", "" );
}
Leet Skillz k\V\Iz
Splotit #8
// force the MIME-TYPE and filename for the file download link.
if (request.getParameter("filename") != null) {
    response.setHeader("Content-disposition", "attachment;filename=" + request.getParameter("filename"));
    response.setContentType("application/octet-stream");
    InputStream fis = null;
    try {
        String sSeparator = System.getProperty("file.separator");
        String sPath = pageContext.getServletContext().getRealPath("/");
        File fileIn = new File(sPath + sSeparator + request.getParameter("filename"));
        fis = new FileInputStream(fileIn);
        int bit = 256;
        while (bit >= 0) {

• CWE 73

• What if the attacker had passed this in as the filename?

../../../etc/passwd

• If an attacker can specify the path to a resource, there will only be trouble!
Leet Skillz kWlz
Sploit #9
```java
final String sID = request.getParameter("id");
int id;
try {
    id = Integer.parseInt(sID);
} catch (NumberFormatException x) {
    log.info("Invalid ID: " + sID);
}
```
CWE 117

You open your application to log forging by passing user generated content through to the logs without validation.

This is related to the previous Sploit, HTTP Response Splitting.

Again, embedded CRLF’s in the input can allow an attacker to inject forged information into your logfiles.

This could be employed along with social engineering to attack your application.
In our previous example, if the ID passed in on the HTTP request looked like this:

```
bogusID\0\0FATAL:
+Reboot+Window+for+Freshness+N
```

Your sysadmin is going to see this:

```
INFO: Invalid ID: bogusID
FATAL: Reboot Windows for Freshness NOW
```
String sRet;

try {
    sRet = p_sString.substring(p_beginIndex, p_endIndex);
}

catch (StringIndexOutOfBoundsException x) {
    x.printStackTrace();
    ms_log.debug("safeSubstring() failed.");
}
• CWE 497

• Blindly printing stack traces to Standard Out (or even to the logfiles) should be avoided

• Minimizing debug information, especially in a production environment, is wise in general

• An attacker can purposely cause error conditions to occur, revealing internal information about the application
Leet Skillz k\W\lz
Sploit #10
<!-- Header page tile -->
<tiles:insert definition="def.header"/>

<!-- Nav bar -->
<tr>
<td height="1" bgcolor="#000000"></td></tr>
<tr>
<td class="menubar" height="10">
<table width="100%">
<tr align="left" class="menubar">
<html:link styleClass="menubar">
<bean:message key="link.mainMenu"/>
</html:link>
	Report Template Configuration
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td height="1" bgcolor="#000000"></td></tr>
</table>

<!-- End nav bar -->
• CWE 615

• Comments in HTML give attackers information about how your application is structured

• Change HTML comments to JSP comments:
  – From this: <!-- Header page tile -->
  – To this: <%- Header page tile -->
Thank You!