Malicious Code Detection

SCR FOR MALICIOUS INTENTION, NOT JUST VULNERABILITY
# Guidance: State of Play – “Clean pipes”

<table>
<thead>
<tr>
<th>Security-Enforcing Software</th>
<th>Custom-developed</th>
<th>Pre-existing with Source</th>
<th>Pre-existing with Executable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Acquisition</strong></td>
<td>Trustworthy developers (contractor reputation)</td>
<td>NIAP or FIPS evaluation</td>
<td>NIAP or FIPS evaluation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Acquisition Policy, including:</td>
<td>Acquisition Policy, including:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Background checks on developer/supplier</td>
<td>• Background checks on developer/supplier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Trusted distribution (checksums, signatures, authenticated out of band distribution channel)</td>
<td>• Trusted distribution (checksums, signatures, authenticated out of band distribution channel)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Blind buys</td>
<td>• Blind buys</td>
</tr>
<tr>
<td><strong>Design</strong></td>
<td>Separation of duties: Designers modify only parts for which they’re responsible. Reputable, well-understood high-level design languages and tools. Least Privilege/Fail Secure Design</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>Construction</strong></td>
<td>Separation of duties: Coders modify only code for which they’re responsible. Software evaluation tools. Code review</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
| **Testing**                 | Software evaluation tools. Security testing to ensure security requirements are fulfilled. Inter-team Testing | Software evaluation tools. Security testing to ensure security requirements are fulfilled. Inter-team Testing | Software evaluation tools. Security testing to ensure security requirements are fulfilled.
String query = "SELECT account_balance FROM user_data WHERE user_name = " + request.getParameter("customerName");

try {
    Statement statement = connection.createStatement(...);
    ResultSet results = statement.executeQuery(query);
}

<servlet>
    <servlet-name>base</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>

<servlet>
    <servlet-name>alt</servlet-name>
    <servlet-class>org.yours.spring.web.servlet.DispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>base</servlet-name>
    <url-pattern>app</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>base</servlet-name>
    <url-pattern>alt</url-pattern>
</servlet-mapping>
Vulns & Malware: A Subtle Difference

Script Kiddie
AppSec Pro
Organized Crime
Nation State

IT
Admin
Developer
Config/Change Mgmt
Threats & (Unique) Capabilities

- **Administration / Operations**
  - LAN access
  - Credentials
  - Production system access

- **Developer**
  - Access to design & source code
  - Ability to change application & configuration
  - 3rd party library selection, deployment

- **Change/Build/Control Management**
  - Ability to (re)package software binaries
  - Code promotion, dependency injection
  - Creation, rotation, and modification of keys
  - (Often) access to production system environment
New Techniques for New Risk
Malicious Code Detection Supply Chain

- **Assessment** – detects evidently malicious code & uncovers suspicious behavior

- **Change control** – assures deployed software is assessed & unmodified after assessment

- **Monitoring** – Detects & alerts based on suspicious behavior, cued by assessment results
Detection - Assessment
- Enumerate classes of malware

- Decompose malware design into suspicious elements

- Assign identifiable signatures to the various design elements through:
  - Static analysis of binary structure, configuration, and other data
  - Dynamic monitoring of behavior during production deployment
Example Pattern: Back Door

- **Communication**
  - Out of bound
  - In-band
    - Special control (parameter)

- **Control**
  - Constants
  - Special namespaces (Tricker)

- **Functionality**
  - Inversion of control
    - Its own mini-pattern
  - System access (shell)
    - Easier to detect
  - Functionality (REPL)
    - Its own mini-pattern

- Test, Debug

- **Stealth**
MCD Approach Design

Threat

Elements

Signature(s)

Suspicion

Intent
## Disposition

<table>
<thead>
<tr>
<th>Communication</th>
<th>Control</th>
<th>Functionality</th>
<th>Resulting Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>REPL, Shell, IOC</td>
<td>Malicious Functionality</td>
<td>High</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Malicious Functionality</td>
<td>High</td>
</tr>
<tr>
<td>No</td>
<td>IOC</td>
<td>Malicious Functionality</td>
<td>Not a backdoor (Trojan?)</td>
</tr>
<tr>
<td>Yes</td>
<td>REPL, Shell</td>
<td>Normal Functionality</td>
<td>High</td>
</tr>
<tr>
<td>Yes</td>
<td>IOC</td>
<td>Normal Functionality</td>
<td>Medium</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Normal Functionality</td>
<td>Low</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Normal Functionality</td>
<td>None</td>
</tr>
</tbody>
</table>
Malware Canon - Patterns Inter-relate

Elements of Malicious Design

<table>
<thead>
<tr>
<th>Linking/Loading</th>
<th>Payload Delivery</th>
<th>Communication</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Normal functionality</td>
<td>- Direct, network-based</td>
<td>- Direct communication</td>
<td>- Unusual data flow</td>
</tr>
<tr>
<td>- Dead code</td>
<td>- Side-channel</td>
<td></td>
<td>- Apparently dead code</td>
</tr>
<tr>
<td>- Dynamic update</td>
<td>- DNS</td>
<td></td>
<td>- 'Invariant' exceptions</td>
</tr>
<tr>
<td>- Link in another language</td>
<td>- Via client software</td>
<td></td>
<td>- Shell access</td>
</tr>
<tr>
<td>- Unmanaged code</td>
<td>- Via 3rd party</td>
<td></td>
<td>- Access to unusual Creds.</td>
</tr>
<tr>
<td>- Function hooking</td>
<td>- Data exfiltration &amp; encoding</td>
<td></td>
<td>- Environment modification</td>
</tr>
<tr>
<td>- DLL (library) injection</td>
<td></td>
<td></td>
<td>- REPL</td>
</tr>
<tr>
<td>- Resource-loading</td>
<td></td>
<td></td>
<td>- Direct communication</td>
</tr>
<tr>
<td>- Encrypted classloaders</td>
<td></td>
<td></td>
<td>Inversion of Control</td>
</tr>
</tbody>
</table>

Misuse Goals

- Theft
  - User Creds.
  - Confidential Info
- Corruption
  - Printing $$
  - Siphoning $
- DoS
  - App
  - System

Symptoms

- Filename generation
- High-entropy data
- Duplication of OS services
- Manipulation of critical security parameters

Stealth

- Obfuscation
  - Jar/Assembly Annotation
  - Code transposition
  - Instruction substitution
  - Self-mutating code
  - Encryption

- Code Mutation
  - Self-mutating code
  - Aspects
  - Code injection

Encryption

Trigger

- Hard-wired (Compulsory)
- Time-based conditionals
- Counter-based conditionals
- 'External event'-based

Inversion of Control

- REPL
Assessments – Signatures
Time & Constants – The Easy Part

- if ( System.currentTimeMillis() > XXXX ) { ; }
  - Why would it ever be ‘greater than’? How could you tell this from reporting?

- if ( user.equals(XXXX)) { ; }
  - How do you role-based access control from a privilege escalation?

- if ( action == XXXX) { ; }
  - Controller or hard-coded malware?

- new URL("http://XXXXXXX");
  - Valid production environments should be ‘whitelist-able’
Assessments – Anomaly Detection
Signatures are Brittle

- (Bonus round)

- Signatures find “known bad”
- Anomaly detection finds “unknown bad”

- This could include anything:
  - Libraries & dependencies
  - Binary format & packaging
  - Sizes
  - Encryption
Keys, Credentials

- Hexadecimal encoding
- Structured primitives
  - Byte arrays
  - Strings
- Long constant primitives
Notes on Signatures

- Binary >> Source code
- `grep` + `xpath` + `regexp` >> Commercial tools
- Inaccurate (False Positives) >> Accurate (False Negatives)
- Iterative >> Monolithic scans
- Engineering notes >> Suppressions
- Don’t let the dog go crazy(s)
Configuration Management
Configuration Mgmt Goals and Practice

- Know what you’re running in prod
  - MCD is only as good as its proximity to prod
  - Code deployed after MCD is invisible

- Code authorship/provenance essential
  - Trend analysis
  - Traceability supports follow-up

- Deployment + Configuration Management is a tool
  - Hook for MCD, Automated scanning
  - Can drive prod-only key/credential/config. creation
  - Signing (for privilege, integrity)
  - Back out (back up) to safe versions
Monitoring: Escalation, etc.
Just plug it into the SCR Process

- Normal escalation process includes the threat
- Risk Mitigation != “Fixing the vuln”
- Using intelligence renders the collection mechanism useless
- We need a way to:
  - Separate duties
  - Institute least privilege in each
  - Force collusion (n-1) for success
  - Conspiracy (n) to avoid detection

Respond in line w/ Suspicion
- May use SCR as input
  - Can *not* use it for “vuln mgmt”
- **Suspicion Rating Scheme**
- **Mirror Vuln Mgmt:**
  - As MCD Escalation

Engage Stakeholders
- IR
- Business Owners
- Ops
Escalation Options

1. Do nothing
2. Verify suspicion using a trusted developer
3. Passive Monitoring
   1. Production logging (Dev)
   2. DLP (WAF)
   3. **Must build IR plan
4. Active suppression
   1. Compensating lib/function
   2. (active) WAF rule (fireEye, Guardium)
   3. Dependency injection / weaving
5. Executive-level event
Thank You