Escaping the Java Sandbox

Alternatively:

3 Reasons Why Browsers don’t Support Java Web Applets Anymore

Loosely based off of the Phrack Magazine article:
Twenty Years of Escaping the Java Sandbox

By Ieu Eauvidoum and disk noise

Disclaimer

- I am not an expert in Java Security. (If you think I am wrong, you are probably right, in which case please feel free to correct me)
- I am not liable for any damage you do to your machine attempting to reproduce these exploits
- I am not advocating for the use of these very outdated java exploits that only work on specific JVM versions

Theory: Java

- What is Java?
- A combination of a Virtual Machine (JVM) and a complementary set of programing libraries for that machine (JCL).
- Made by Sun Microsystems 1995, currently owned by Oracle.
- Designed to be widely deployed across a variety of machines.

Theory: Sandbox

- Usually as a result of a Security policy
- Checks the Java.Security permissions system before any calls that can affect the outside of the applet.
- Designed so that web applets cannot:
  - Access your filesystem (outside of temp files defined by your browser policy)
  - Access anything major outside the sandbox (ie. Socket calls)
  - Call anything major that may (or may not) have the ability to change the sandbox (Ie. Reflection, Serialization, Class Loaders, Altering the java.security framework)

Theory: Applets

Java “Applet”

- Implicit security policy
- Usually involves the execution of “untrustworthy” remote code.
- Fine grained security policy as defined by the Java.Security permissions system.
- The default for web java applications and some graphical java applications.

- What the exploits covered in this talk are designed to target.

Java “Application”

- Explicit or no security policy.
- Somewhat trusted code execution.
- No security checks when accessing outside the JVM (Filesystem, peripherals, etc).
- The default after java compilation and execution.

- Most exploits covered in this talk are not required or cannot target these types of applications (no need for privilege escalation)

Theory: Topology of a Java Applet's Security System

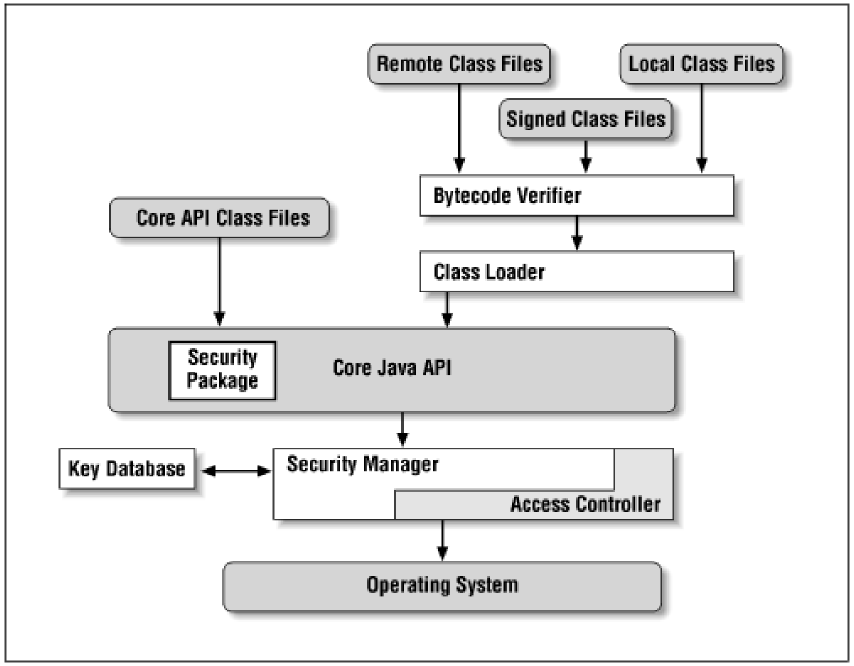


Image from *Java Security* (S. Oaks, 1998)

- Local, Signed, and Remote class files (All written in java byte code) are sent into the bytecode verifier (checks for type errors, among other things)
- Those classes are then sent to the Class Loader and are loaded into the main java program.
- When those classes need to access the OS, or OS Specific calls (for filesystem or socket or other reasons) the security manager and access controller check the privilege of the executing functions (via a stack walk).
- NOTE: the JCL has FULL PERMISSIONS access in the eyes of the JVM and security manager.

Theory: Internal JVM Calls

- Consists of restricted code solely for the use of internal JVM.
- Usually in packages com.sun.* or sun.*
- Very little public documentation (but its open source).
- Always subject to change, either between small JVM updates or an entirely different JVM version.
- Can contain machine specific instructions.
- Full privilege permissions with the java security manager. (Part of the JCL)
- Usually act as the bridge between the JVM and the outside world.
- Can be written in java or native machine languages (ASM, C, C++).

Theory: doPrivileged()

- Security policy does a stack walk to ensure compliance.
- Stack walk ensures that all the functions called in that stack have proper permissions according to the security policy.

Theory: doPrivileged()

- Security policy does a stack walk to ensure compliance.
- Stack walk ensures that all the functions called in that stack have proper permissions according to the security policy.

- Problem: What if the JVM needs to execute sensitive functions in the execution of regular code, (ie. Getting the architecture type when interpreting streams in unprivileged code).

Theory: doPrivileged()

- Security policy does a stack walk to ensure compliance.
- Stack walk ensures that all the functions called in that stack have proper permissions according to the security policy.

- Problem: What if the JVM needs to execute sensitive functions in the execution of regular code, (ie. Getting the architecture type when interpreting streams in unprivileged code).
- Solution: doPrivileged()
  - doPrivileged will halt any security stack checks and use the current function’s permission (usually a JCL function which will have all permissions).
  - Naturally designed to escalate privileges.

Theory: Security Manager Initialisation

- Everything in java is a class, the security manager is no exception.
- All failed security checks throw the java.security.SecurityException;
- Security system checks (due to “performance”) are only done if the Security manager class is not null.

Theory: Security Manager Initialisation

- Everything in java is a class, the security manager is no exception.
- All failed security checks throw the java.security.SecurityException;
- Security system checks (due to “performance”) are only done if the Security manager class is not null.

What if a privileged class was to do *SetSecurityManager(null)*?

Theory: Commonly exploited Java libraries

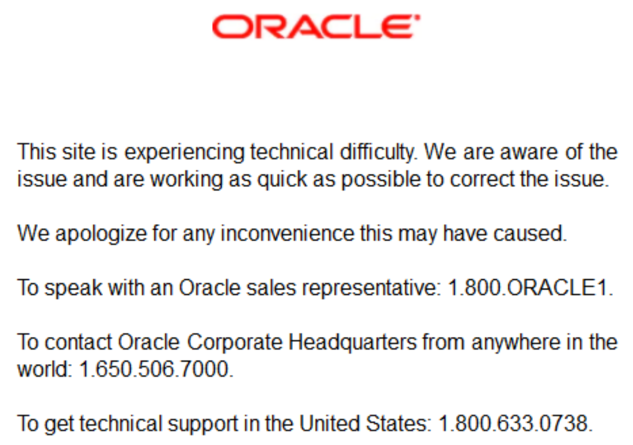
- Class Loader: The library that allows for the creation of custom classes at runtime.
- Reflection: The library which allows for the modification of access variables (ie. Private, public) at runtime.
  - Rated the most common vulnerability in java (Gorenc and Spelman, 2013)
- Serialisation: The library which converts classes to and from serialisable bytes for network transmission.

Practical

PRACTICAL: TYPE CONFUSION

Type Confusion Introduction

- Occurs when the JVM thinks an object is of Type A when it is actually Type B
- Most of the time Type Confusion attacks are class loader based.
- Usually involves overwriting a class defining function and making the JVM think that your functions are privileged class definitions.


ORACLE

This site is experiencing technical difficulty. We are aware of the issue and are working as quick as possible to correct the issue.

We apologize for any inconvenience this may have caused.

To speak with an Oracle sales representative: 1.800.ORACLE1.

To contact Oracle Corporate Headquarters from anywhere in the world: 1.650.506.7000.

To get technical support in the United States: 1.800.633.0738.

Type Confusion Example: CVE-2017-3272

(Not working, oracle won't send me the link to jdk 1.6.01)

Post Mortem: Type Confusion

- Uses Atomic Field reference updater (initially designed for concurrency and altering atomic class values)
- New Updater does the following:
- Updates a field (Via reflection called in doPrivileged)
- Uses the sun.misc.Unsafe class to directly alter the memory of the program.
- Then proceeds to use the set() function which uses sun.misc.Unsafe to alter the new value via memory.
- The set() function, (in java 1.8 u112) has no checks for classes when overwriting the value (only checks for casting), which in the case of the demo, we manage to overwrite a string class with an integer class.

Uses and Fixes: Type Confusion

- Uses: Hypothetically overwriting JCL classes or other privileged classes with different values or casting a separate function in place of a protected function can allow for privilege escalation.
- Fixes: Consistent type checking (too much overhead)

PRACTICAL: CONFUSED DEPUTY

Confused Deputy Introduction

- Occurs when a rogue class aims to trick a system class into performing access violations
- Confused deputy attacks commonly use reflection based attacks to alter private fields.
- Usually involves dodgy sun classes that have full privilege.

Confused Deputy Example: CVE-2012-4681

Post Mortem: Confused Deputy

- Uses sun.awt.suntoolkit, a restricted internal JVM class.
- Sun.awt.suntoolkit is grabed using java.beans.expression a JCL class with improper permission checking (The Deputy).
- This uses custom logic which allows for the creation of a new class without the permission checking
- Suntoolkit.getfield is then used to get access to other private methods of system classes (yet another deputy)
- The getField can now be used to access the private permissions class and therefore add permissions to new classes.
- Eventually you can escalate your privileges using dummy access control contexts.
- The original author of the article hypothesizes that he arbitrarily executed a setSecurityManager(null) command to disable the security manager via a new class with escalated privileges.

Uses and Fixes: Confused Deputy

- Uses: Powerful privilege escalation inside applets allowing for the nullification of a security manager.
- Fixes: correctly ensuring that systems classes are “safe” to use.

PRACTICAL: TRUSTED METHOD CHAIN

Trusted Method Chain Introduction

- Occurs when the call stack only visibly contains “trusted classes”, thus tricking privilege checking into thinking that its safe.
- Trusted Method Chain attacks commonly use reflection based attacks to hide the exploit code or execute the exploit code from the clean call stack.
- Must find a way to hide the main method, as its usually considered untrusted.

Trusted Method Chain Example: CVE-2010-0840 (ONLY CODE REVIEW)

:(sorry no JDK 1.6.01 download avaliable

Post Mortem: Trusted Method Chain

- Uses java.beans.Statement (The Link class) to execute methods via reflection.
- Forces a Jlist Swing class to attempt to “draw” something (instead it executes a fake GUI element)
- Link also implements a fake getValue() method of the Map.Entry interface
- As a result of this the bytecode of Link must be modified after compilation but before runtime.
- AbstractMap.toString() is invoked, which then invokes the link object (but its not on the stack).
- As a result of this the stack appears clean to the security manager and the link object (which has an arbitrary target method) is allowed to execute with full permissions.

Uses and Fixes: Trusted Method Chain

- Uses: Powerful privilege escalation inside applets allowing for the nullification of a security manager.
- Fixes: correctly ensuring that systems classes are “safe” to use.

Conclusion

- Java is big, larger systems are the more prone they are to risk.
- No real viability for untrusted applets (java browser games)

Extra????