Understanding the Java Serialization Attack Surface

Daniel Grzelak, **stratsec**

Ruxcon, November 2010

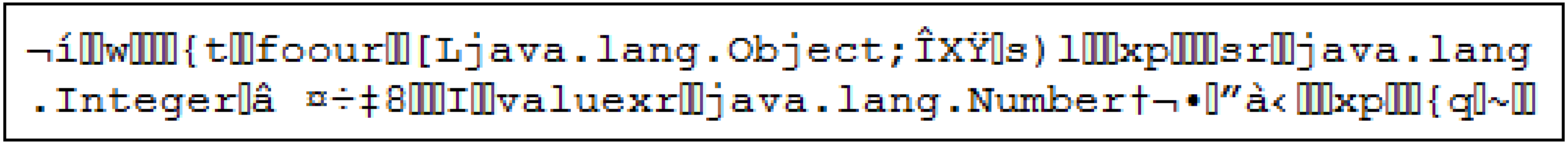
Quick question

Who has tested Java Serialization enabled applications?

OR

Who has seen serialized Java object flying across their web proxy?

For those that haven’t…



```
¬í▯▯w▯▯▯▯{t▯▯foour▯▯[Ljava.lang.Object;ÎXŸ▯s)l▯▯▯xp▯▯▯▯sr▯▯java.lang
.Integer▯â ¤÷‡8▯▯▯I▯▯valuexr▯▯java.lang.Number†¬•▯”à‹▯▯▯xp▯▯▯{q▯~▯▯
```

This presentation

- We will…
  - Figure out what is wrong with serialization
  - Learn how to abuse serialization
  - Analyse client-server usage of serialization

- I won’t…
  - Examine client-side exploitation
  - Drop any 0-day or change the world

If you are interested in client-side

- (Slightly) Random Broken Thoughts
  - Sami Koivu
  - http://slightlyrandombrokenthoughts.blogspot.com/
- Cr0 Blog
  - Julien Tinnes
  - http://blog.cr0.org/

How do you spot serialization?

```
¬í  sr  javax.swing.JFrameÞßØUº¡B    I  defaultCloseOperationZ
rootPaneCheckingEnabledL  accessibleContextt
'Ljavax/accessibility/AccessibleContext;L  rootPanet
Ljavax/swing/JRootPane;L  transferHandlert
Ljavax/swing/TransferHandler;xr
```

- java.io.ObjectStreamConstants

```
final static short STREAM_MAGIC = (short)0xACED;
```

Getting started…

- Java makes everything easy!
  - If you know what is in the stream…
  - All you need is a “java.io.ObjectInputStream”

```
myFileInputStream = new FileInputStream("objectfile");
myObjectInputStream = new ObjectInputStream(myFileInputStream);

Integer myInteger =
        (Integer)myObjectInputStream.readObject();

String myString =
        (String)myObjectInputStream.readObject();

Object[] myObjectArray =
        (Object[])myObjectInputStream.readObject();
```

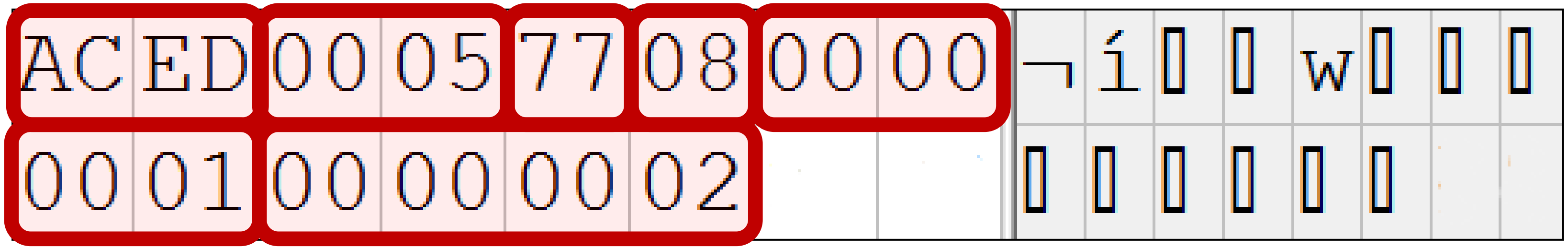
What about basic types?

```
byte b    = myObjectOutputStream.readByte();
char c    = myObjectOutputStream.readChar();
boolean d = myObjectOutputStream.readBoolean();
int i     = myObjectOutputStream.readInt();
long l    = myObjectOutputStream.readLong();
double d  = myObjectOutputStream.readDouble();
float f   = myObjectOutputStream.readFloat();
```

Basic types suck!

- Let’s write some basic types

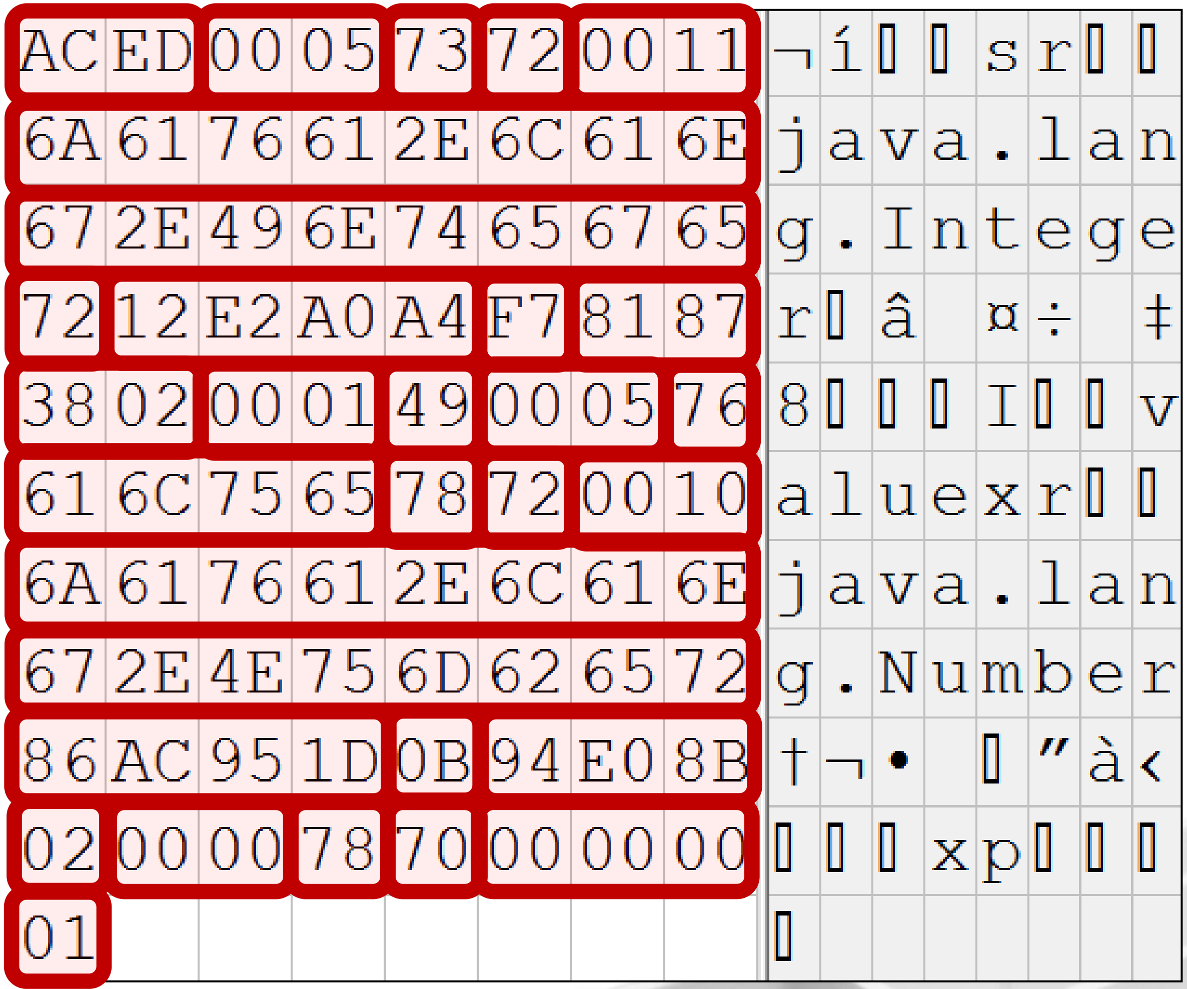
```
myObjectOutputStream.writeInt(1);
myObjectOutputStream.writeInt(2);
```



```
AC ED 00 05 77 08 00 00  ¬í□ □ w□ □ □
00 01 00 00 00 02        □ □ □ □ □ □
```

- Can anyone spot why?

Let’s check out an object (java.lang.Integer=1)



```
AC ED 00 05 73 72 00 11   ¬ í ▯ ▯ s r ▯ ▯
6A 61 76 61 2E 6C 61 6E   j a v a . l a n
67 2E 49 6E 74 65 67 65   g . I n t e g e
72 12 E2 A0 A4 F7 81 87   r ▯ â   ¤ ÷   ‡
38 02 00 01 49 00 05 76   8 ▯ ▯ ▯ I ▯ ▯ v
61 6C 75 65 78 72 00 10   a l u e x r ▯ ▯
6A 61 76 61 2E 6C 61 6E   j a v a . l a n
67 2E 4E 75 6D 62 65 72   g . N u m b e r
86 AC 95 1D 0B 94 E0 8B   † ¬ •   ▯ ” à ‹
02 00 00 78 70 00 00 00   ▯ ▯ ▯ x p ▯ ▯ ▯
01                        ▯
```

STREAM_MAGIC
STREAM_VERSION
TC_OBJECT
TC_CLASSDESC
Class description length (17)
Qualified class name
Serial version UID
Description flags
Object handle
Field count (1)
Field type code (int)
Field name length (5)
Field name
TC_ENDBLOCKDATA
TC_CLASSDESC
Class description length (16)
Qualified class name
Serial version UID
Description flags
Object handle
Field count (1)
TC_ENDBLOCKDATA
TC_NULL
The actual value (1)

stratsec

That was heavy going… any questions?

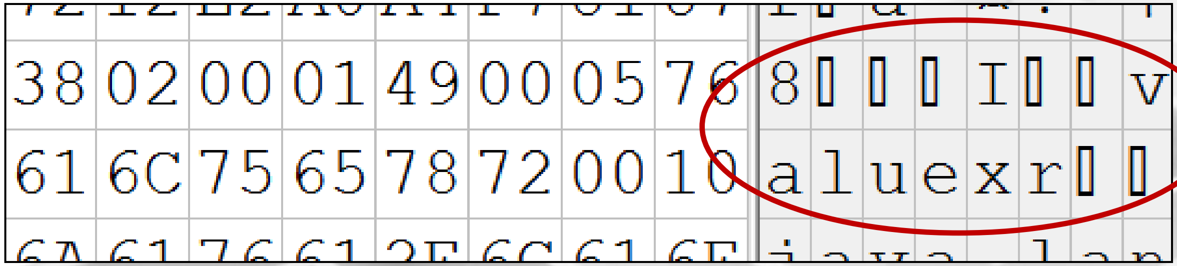
http://download.oracle.com/javase/6/docs/platform/serialization/spec/protocol.html

Those class definitions flying around…

- Don’t class definitions have code?
  - I didn’t see any code!
- Those were not so much class definitions?
  - More object snapshots
  - Sorry. I lied! ☹
- Client-side attacks are more fun
  - Define objects and inheritance hierarchies
  - Define code

Let’s review java.lang.Integer code anyway

```
...
private final int value;
public Integer(int value)
public Integer(String s)
public byte byteValue()
public int compareTo(Integer anotherInteger)
public double doubleValue()
public boolean equals(Object obj)
public float floatValue()
...
```



```
38 02 00 01 49 00 05 76   8 □ □ □ I □ □ v
61 6C 75 65 78 72 00 10   a l u e x r □ □
6A 61 76 61 2E 6C 61 6E   j a v a . l a n
```

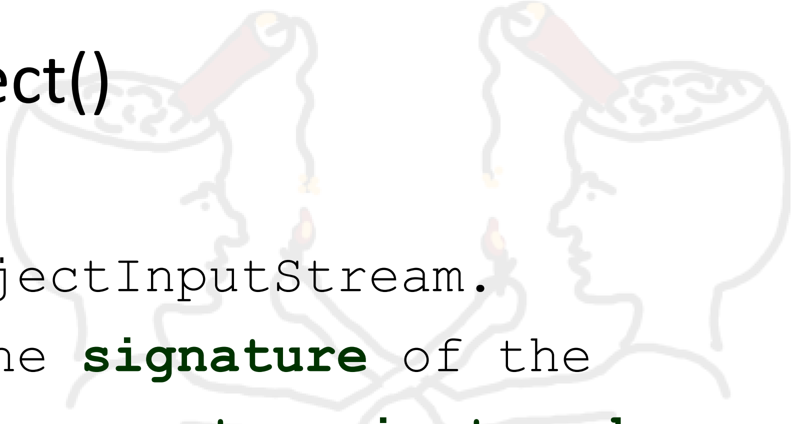
stratsec

What exactly is serialized?

- ObjectInputStream.readObject()

```
/**
 * Read an object from the ObjectInputStream.
 * The class of the object, the signature of the
 * class, and the values of the non-transient and
 * non-static fields of the class and all of its
 * supertypes are read.
 ...
```

- **Private**, **protected**, and **final** fields are all read



Attack scenario: Private/final members

- If a class relies on private or final values being unchangeable, we may be able to attack it

- Consider an exchange rate in a shopping cart
  - This may get sent to the client connect time
  - Or may be sent to the server as part of a transaction

```
public class AustralianDollar
{
     private final double exchangeRate 0.9;
}
```

Side note

- This means the client and server need not have the same definition of an object that is serialized
- They only have to have the same signature
  - Same fully qualified name
  - Same non-static, non-transient fields

- In practice this hardly ever happens
  - But check your assumptions when auditing!

What now?

- We need to modify objects without a hex editor
  - A first attempt:

```
try
{
   Object currentObject = myObjectInputStream.readObject();

   if(currentObject.getClass().getName() == "java.lang.Integer")
      handleInt((Integer)currentObject);
   else if(currentObject.getClass().getName()=="java.lang.String")
      handleString((String)currentObject);
   else if (currentObject.getClass().getName()=="[Ljava.lang.Object;")
      handleObjectArray((Object[])currentObject);
}
```

Reflection to the rescue

```
private static void traverseObject(
   Object currentObject, Class currentClass)
{
   Field[] currentFields = currentClass.getDeclaredFields();
   for(int i=0; <currentFields.length; i++)
   {
      ...  // inspect each field
   }

   if(currentClass.isArray())
   {
      ... // work with each object in the array
   }
}
```

Inspecting fields

```
if(Modifier.isStatic(currentFields[i].getModifiers()) ||
   Modifier.isTransient(currentFields[i].getModifiers()) )
   continue;

try {
   currentFields[i].setAccessible(true);
   Object memberObject = currentFields[i].get(currentObject);
   Class memberType = currentFields[i].getType();

   if(memberType.isPrimitive()) {
      //Do something with memberObject
   } else {
      traverseObject(memberObject, memberType);
   }
} catch (IllegalAccessException iae) {}
```

Working with arrays

```
Class componentType = currentClass.getComponentType();

if(componentType.isPrimitive()) {
   for(int i=0; i<Array.getLength(currentObject); i++)
   {
      //Do something with Array.get(currentObject, i);
   }
} else {
   Object[] componentArray = (Object[])currentObject;
   for(int i=0; i<componentArray.length; i++)
   {
      traverseObject(componentArray[i],
         componentArray[i].getClass());
   }
}
```

So we can work with fields…

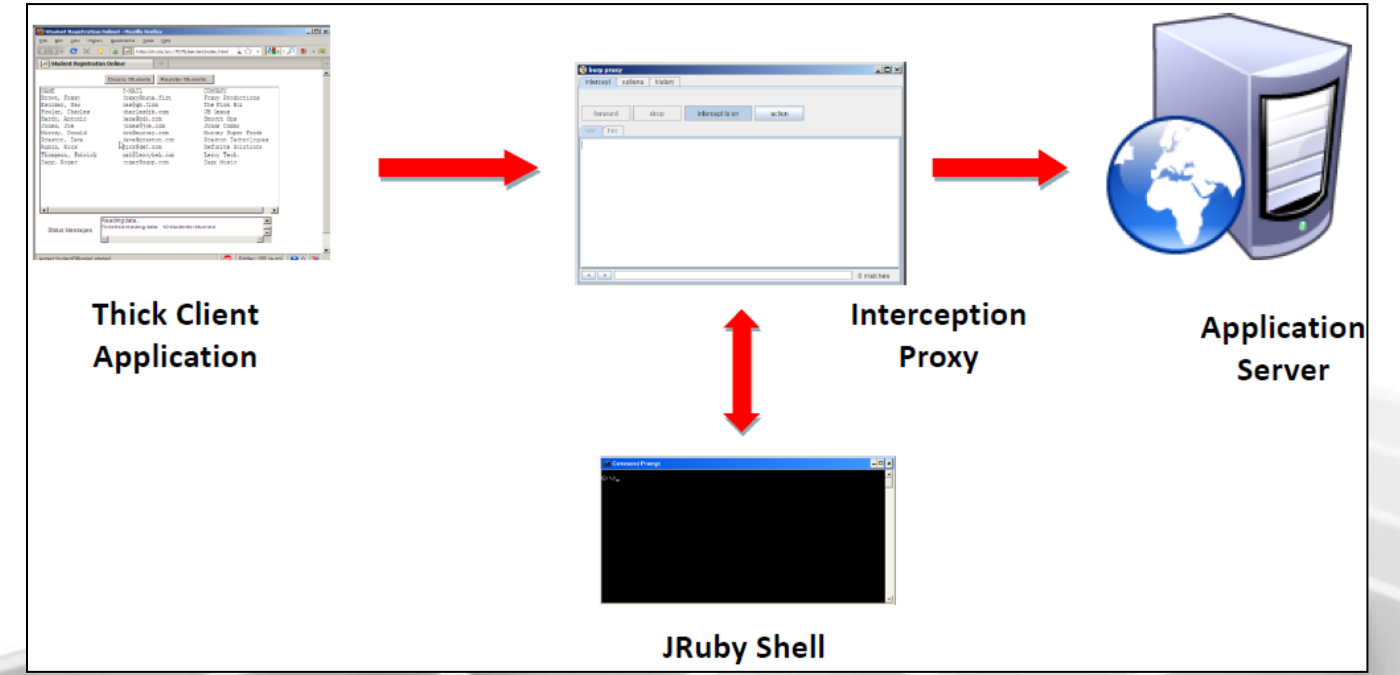
- You can now build a generic fuzzer for serialized objects

```
if(memberType == String.class)
{
   currentFields[i].set(currentObject,
      new String(“FUZZED”));
}
```

- The previous code will also hit private, final, and protected fields

But I just want to hack ☹

- If you want something more ./consult
  - Check out DSer by Manish S. Saindane http://www.andlabs.org/tools.html
  - JRuby shell plugin for burp



Thick Client Application

Interception Proxy

Application Server

JRuby Shell

Serialization from a coder's perspective

- In order for an object to be serialized it must implement the "java.io.Serializable" interface
  - No actual methods required

- If any custom logic is required, it must implement:

  -
    ```
    private void readObject(ObjectInputStream)
      throws IOException, ClassNotFoundException;
    ```

  -
    ```
    private void writeObject(ObjectOutputStream)
      throws IOException;
    ```

Java and readObject()

- Once the JVM has identified and object type (remember TC_CLASSDESC)
  - It will try to find and call that class' readObject()
- Depending on the circumstances it may also call:
  - private void readObjectNoData()
    throws ObjectStreamException;
  - ANY-ACCESS-MODIFIER Object readResolve()
    throws ObjectStreamException;
- Java will also invoke the no-argument constructor of the first non-serializable superclass

Attack Scenario: Busted readObject() etc

- Sometimes the readObject() implementation for a given class will be outright broken
  - Typically you will have access to the object’s defintion

- Consider the following:

```
public static void main(String[] args) {
    try
    {
        String myCommand =
            (String)myObjectInputStream.readObject();
        Runtime.getRuntime().exec(myCommand);
    } catch (IOException ioe) {}
}
```

Notes for reviewers

- Review existing Java classes:
  - ~260 classes implementing readObject
  - ~220 classes implementing writeObject
  - ~3 classes implementing readObjectNoData
  - ~35 classes implementing readResolve
  - ~10 classes implementing writeReplace

More interestingly

- The readObject() called is defined completely by the string after TC_CLASSDESC

- Disassembling a call to ObjectInputStream.readObject gives:

```
35:  invokevirtual  #7; //Method
java/io/ObjectInputStream.readObject:()Ljava/lang/Objec
t;
38:  checkcast      #10; //class java/lang/String
```

  - The call to the custom readObject() is inside the call
  - The cast to its final type, is outside the call

Abusing uncast objects

- Applications will often:
  - Not cast an object at all
  - Cast the object to an interface
  - Delay cast of the object till after some logic has executed

- All of these are potentially dangerous
  - All allow an object to be misinterpreted as something other than what we supply

Object not cast at all

- Without casting, we have a “java.lang.Object”
  - Everything class in java descends from Object
- Many descendants of Object override:
  1. toString()
  2. equals()
  3. clone()
  4. hashCode()
- We can supply any serializable object
  - And execute an alternative to what is expected

Attack Scenario: Uncast object

- Consider:
  -
    ```
    log.writeEntry(“User logged in with username
    “ + deserializedUserObject);
    ```
  - toString() is called implicitly
  - A “User” class is expected
  - What if we supply a “String”
- Or:
  -
    ```
    if(deserializedObject == test)
       doSomethingBad();
    ```
  - equals() of the first class is called implicitly
  - What if supply a class where equals() is less strict?

Object cast to an interface

- The same concept applies to interfaces or other super-classes
  - Just substitute another class that implements the interface but does something unintended

- Some commonly used interfaces which may be fun to explore:
  - java.lang.Comparable
  - java.lang.Runnable
  - Java.util.Enumeration
  - ...

Attack Scenario: Half-cast interfaces

- Consider:

  -
    ```
    Runnable myHarmlessTask =
       Runnable)myObjectInputStream.readObject();
    ```

  -
    ```
    myHarmlessTask.run();
    ```
  - We can replace this with another object that implements Runnable but does something more sinister. A *workerThread* class perhaps?

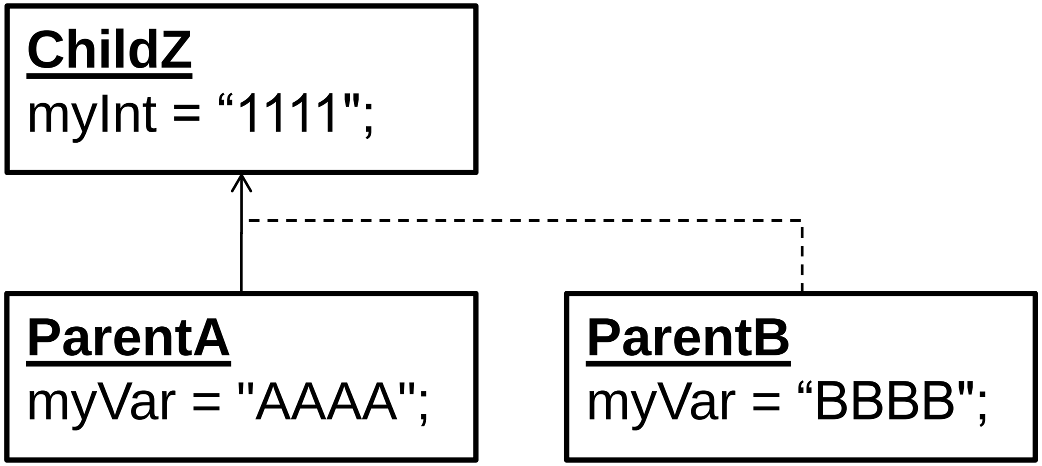
- Note also that all descendants of a serializable class are themselves serializable

Wacky inheritance action

- It is possible to substitute parent classes and cause strange behaviour
  - Maintain real hierarchy when deserialized
  - Null fields of deserialized parent class
  - Prevent correct readObject() from being called
- Are there any security implications?

Inheritance demo

- ParentB.serialVersionUID = ParentA. serialVersionUID



ChildZ
myInt = “1111";

ParentA
myVar = "AAAA";

ParentB
myVar = “BBBB";

Serialized references

- Java is smart
  - If an object references another serialized object, a reference structure is written
  - Cant reference objects outside of the stream

- References to non-serializable objects prevent serialization

Attack scenario: Recursive referencing

- Consider:

```
– MyObj o =
   (MyObj)myObjectInputStream.readObject();
  MyObj next;
  while( (next = o.nextObject) != null)
  {
     o = next;
     o.doSomething();
  }
```

- With serialized references we can create an infinite loop;
  - Make next self referencing

Attack scenario: Information gathering

- Identify valid serial Version UIDs
  - Change class name to existing one

  -
    ```
    java.io.InvalidClassException: WrongClass; local
    class incompatible: stream classdesc
    serialVersionUID = 3277712643214068861, local
    class serialVersionUID = 4720308871306631797
    ```

- Identify existence of classes
  - Change class name to non-existent one

  -
    ```
    java.lang.ClassNotFoundException: WrongClass
    ```

But I want code exec!

- It's fairly unlikely ☹

- The closest I have seen:

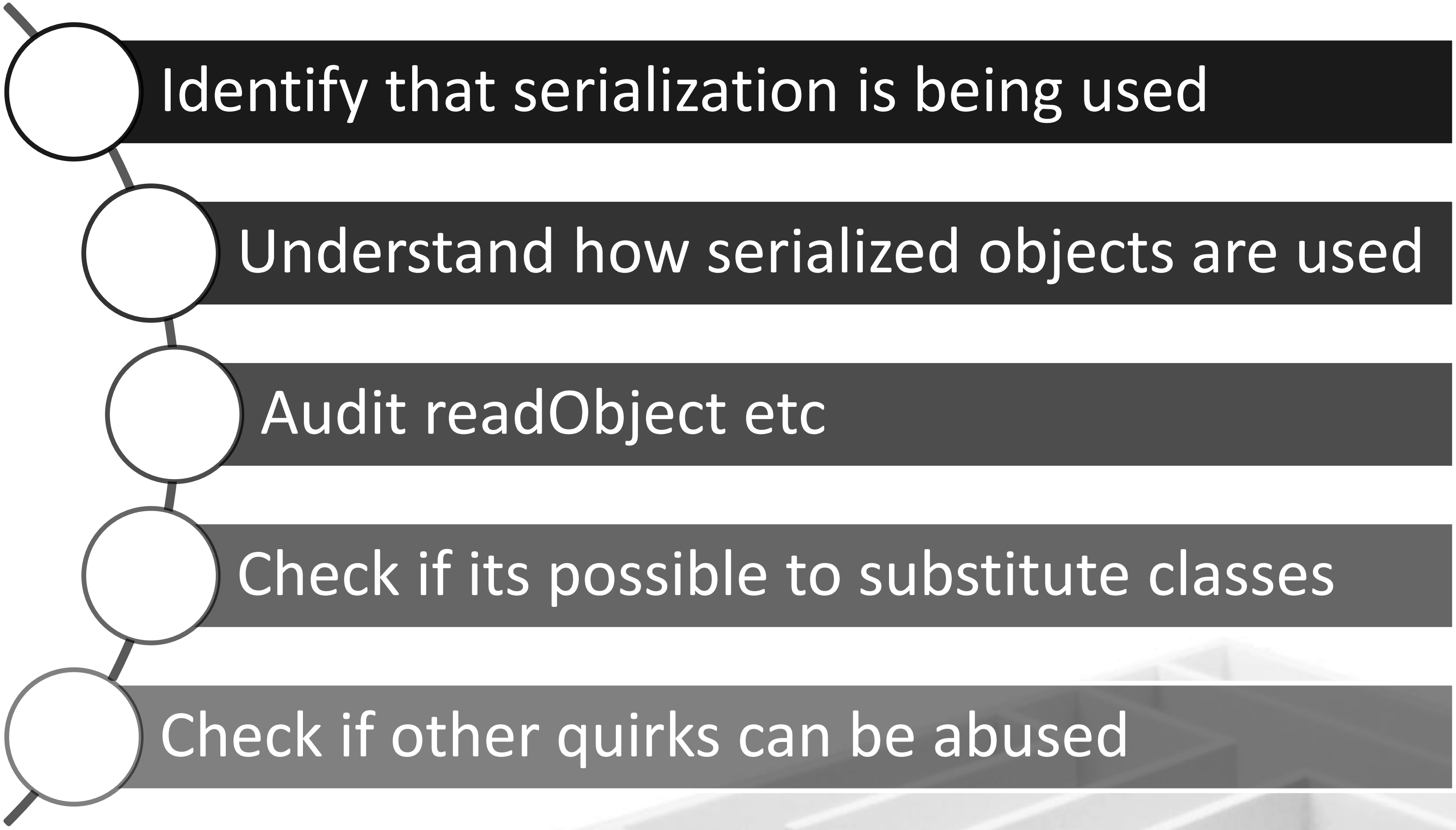
```
Method myMethod = myClass.getMethod(
   userSuppliedMethod, userSuppliedArgsClasses);
myMethod.invoke(myObject, userSuppliedArgs);
```

…where the values come from:

```
http://company.com/application/class/methodname
```

- Has anyone seen anything worse?

Testing process summary



1. Identify that serialization is being used
2. Understand how serialized objects are used
3. Audit readObject etc
4. Check if its possible to substitute classes
5. Check if other quirks can be abused

Some gotchas to avoid

- Ensure you have access to class definitions
  - otherwise you will get nothing but “ClassNotFoundException” exceptions.

- Applications sometimes wrap the output of an “ObjectOutputStream” inside a byte array.
  - Create two “ObjectInputStreams”, one for the byte array, and another to get objects from the byte array

Conclusion

- It’s not as bad as it looks
- Most attacks are logic dependant
- Java works in mysterious ways

Any questions?

stratsec