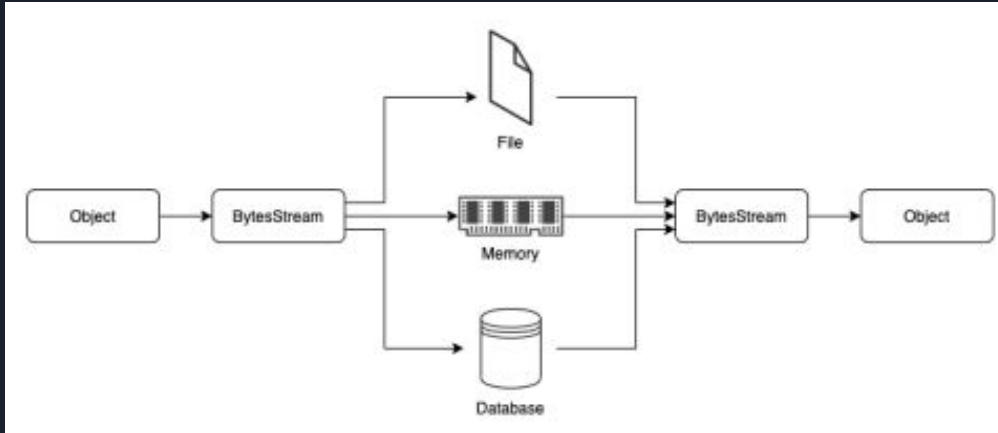




Java Deserialization

By Rohit Narayanan M

Serialization & Deserialization



- Serialization is the process of packaging program-internal object-related data in a way that allows it to be externally stored or transferred.
- The process of reconstructing an object from a byte sequence is called deserialization often referred to as unmarshalling

Why do We do it

Serialization allows services and applications to communicate with each other by sending data that can be processed

Serialization is also used for caching frequently used data

eg: Used in Sun Java web console, where a vulnerability was found later

Serialization in java also allows to preserve objects as different objects have different time spans


```
AC ED 00 05 73 72 00 31 63 6F 6D 2E 64 65 73 65 72
69 61 6C 69 7A 61 74 69 6F 6E 2E 65 78 70 6C 6F 69
74 2E 6D 6F 64 65 6C 73 2E 4A 61 76 61 43 6C 61 73
73 45 6D 70 74 79 99 84 88 80 5A F0 F4 1D 02 00 00
78 70
```

0xAC ED	STREAM_VERSION	The deserialization output always starts with this value, which is often referred to as magic number. The purpose of the magic number is to indicate to the Java runtime that this is indeed a serialization protocol
0x00 05	STREAM_VERSION	The next 16 bits indicates which serialization version was used to serialize the object.
0x73	TC_OBJECT	The TC_OBJECT tells Java that it is a class that has been deserialized.
0x 72	TC_CLASSDESC	Indicates the start of a class description
0x00 31	classname length	The length of the class name in hex, the full class path of JavaEmptyClass is 49 characters long. The hex value 31 equals 49 when converted to decimal. The class name is represented with the package path.
0x99 84 88 80 5A F0 F4 1D	serialVersionUID	the serialVersionUID specifies which class version is used. The hash is calculated using the NIST Secure Hash Algorithm (SHA-1) where the first two 32 bit quantities are used to form a 64-bit stream identifier (Oracle, 2020).
0x02	SC_SERIALIZABLE	This flag indicates that the class implements the Serializable interface.
0x00 00	fieldCount	The next 16 bites are used to represent how many fields are in the class.
0x78	TC_ENDBLOCKDATA	This indicates the end of the data block for the object.
0x70	TC_NULL	A TC_NULL after TC_ENDBLOCKDATA indicates that the class has no super class.

Why is it dangerous

readObject	Invoked during deserialization if defined in the serializable class.
readResolve	Invoked during deserialization if defined in the serializable class.
validateObject	Invoked during deserialization if defined in the serializable class.
readObjectNoData	Invoked during deserialization if defined in the serializable class.
Finalize	Invoked during during garbage collection by the garbage collector. Can be used for deferred execution of exploitable methods.
readExternal	Invoked during deserialization if defined in the serializable class. Defined in the classes implementing the Externalizable interface.

Magic methods get executed automatically by the deserializer, even before deserialization finishes!



Many serializable JDK classes implement these magic methods and call other methods, so there's a lot of additional "known entry points."

HashMap

- `Object.hashCode()`
- `Object.equals()`

PriorityQueue

- `Comparator.compare()`
- `Comparable.compareTo()`



Deserialization vulnerabilities in java

For an application to be vulnerable to deserialization attacks it needs to meet two criteria.

1. The application must accept serialized data from a location accessible to an attacker.
2. The vulnerable class must be present on the classpath of the application accepting serialized data



Deserialization Gadgets

A deserialization gadget is a class residing within the application code or a library, it must be reachable by the Java class loader, the class can be used to facilitate an attack.

Gadget classes that are present in the core Java class libraries are often referred to as a "Golden Gadget"

Magic Methods to Gadget chains

Payload

Code

```
public class HashMap<K,V> implements Map<K,V> {  
    private void readObject(ObjectInputStream s) {  
        int mappings = s.readInt();  
        for (int i = 0; i < mappings; i++) {  
            K key = (K) s.readObject();  
            V value = (V) s.readObject();  
            putVal(key.hashCode(), key, value);  
        }  
    }  
}
```

```
public class AbstractTableModel$fff19274a {  
    private IPersistentMap __closureFnMap;  
    public int hashCode() {  
        IFn f = __closureFnMap.get("hashCode");  
        return (int) f.invoke(this);  
    }  
}
```

```
public class FnCompose implements IFn {  
    private IFn f1, f2;  
    public Object invoke(Object arg) {  
        return f2.invoke(f1.invoke(arg));  
    }  
}
```

```
public class FnConstant implements IFn {  
    private Object value;  
    public Object invoke(Object arg) {  
        return value;  
    }  
}
```

```
public class FnEval implements IFn {  
    public Object invoke(Object arg) {  
        return Runtime.exec(arg);  
    }  
}
```

```
{  
    "@class": "java.util.HashMap"  
    "members": [  
        2,  
        {  
            "@class": "AbstractTableModel$fff19274a"  
            __closureFnMap: {  
                hashCode: {  
                    "@class": "FnCompose"  
                    f2: { "@class": "FnConstant", value: "/usr/bin/calc" },  
                    f1: { "@class": "FnEval" }  
                }  
            }  
        },  
        "val"  
    ]  
}
```




How to find it

To find deserialization vulnerabilities Look whether any serialization functions are used and check whether we can control the data to these functions

Also if we don't have code we can check for magic bytes `0xAc 0xED` or `rOO` in the network traffic.

When we find that we can deserialize data of our like, we search for gadget chains



Tools

Locating the gadget chains is the complex part. For that we can use tools

- Ysoserial

It is a collection of known gadget chains and exploits

- Gadget inspector

It is a Java bytecode analysis tool for finding gadget chains in Java applications or packages.

- Jooble

Programmatically query about types/methods of the classpath

- Marshalsec

Deserialization payload generator for numerous libraries and gadget chains



Variable modification attack

It is a type of modification attack where we modify a variable in a serialized byte stream.

We can do that using tools like serialization dumper which converts byte streams into more human readable form and back to byte streams.

Deferred Execution Attack

It's a type of attack where the execution of the payload is deferred, until after the deserialization process has returned the object. So the payload is only executed after the object is destroyed by garbage collector.

For that we can use the magic methods like `finalize` which is executed during garbage collection.



Polymorphism attack

It is a type of attack where polymorphism is exploited in order to have methods in unintended objects invoked.

So if there is 2 classes User and AdminUser and AdminUser class extends User class. Then if the attacker knows about the AdminUser class, then the he can create an Adminuser class byte stream and pass it to deserialize and then whatever is executed as user will be executed as AdminUser instead.



Proxy attack

It is a type of gadget chain attack, where a proxy is used to intercept methods calls to an object, forwarding them to a abuse gadget. This can be used if no interesting methods can be reached by magic methods in any of the Serializable classes in the application.

These are some methods which can be used for this type of attack

We can specify an argument `targetMethod` in some functions, which we can give as “exec” and for `targetObject` we can give any class which have `Runtime.class`. And arguments as an array of Strings.

- `java.lang.reflect.InvocationHandler.invoke()`
- `javassist.util.proxy.MethodHandler.invoke()`
- `org.jboss.weld.bean.proxy.MethodHandler.invoke()`



How to prevent it

- Developer could only include libraries that are strictly necessary for the application
- If the class is not supposed to be serialized Implement magic methods by throwing a `NotSerializableException`

```
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException{
    throw new NotSerializableException("This class is
    not intended to be deserialized");
}
```

- Do not serialize untrusted data
- Blacklisting and whitelisting
- Signing the serialized data

Parrot0x

Gadget chain found using gadget inspector

1. java/security/cert/CertificateRevokedException.readObject(Ljava/io/ObjectInputStream;)V (1)
2. java/util/Collections\$CheckedMap.put(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object; (1)
3. java/util/TreeMap.put(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object; (0)
4. com/fword/utis/UserComparator.compare(Ljava/lang/Object;Ljava/lang/Object;)I (0)
5. com/fword/utis/UserComparator.compare(Lcom/fword/utis/User;Lcom/fword/utis/User;)I (0)
6. com/fword/utis/UtilityEval.handle(Ljava/lang/Object;)Ljava/lang/Object; (1)
7. java/lang/Runtime.exec(Ljava/lang/String;)Ljava/lang/Process; (1)

```
public Object handle(final Object arg) {
    try {
        Runtime.getRuntime().exec((String)arg);
        return 1;
    }
    catch (IOException ex) {
        System.out.println("Exception in runtime.exec");
        return 0;
    }
}
```

```
@Override
public int compare(final User o, final User ob) {
    if (this.questionObj.getCategory() != null) {
        final Manager m = this.questionObj.getCategory();
        return (int)m.handle((Object)this);
    }
    final Manager m = (Manager)new Category();
    return (int)m.handle((Object)this);
}
```



Tools

- <https://github.com/frohoff/ysoserial>
- <https://github.com/Contrast-Security-OSS/joogle>
- <https://github.com/mbechler/marshalsec>
- <https://github.com/JackOfMostTrades/gadgetinspector>
- <https://github.com/ikkisoft/SerialKiller>
- <https://github.com/NickstaDB/SerializationDumper>



References

- <https://www.blackhat.com/us-18/briefings/schedule/#automated-discovery-of-deserialization-gadget-chains-10668>
- <https://www.youtube.com/watch?v=MTfE2OgUIKc>
- <http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Schoenefeld-up.pdf>
- <https://frohoff.github.io/appseccali-marshalling-pickles/>
- <https://appsecus2018.sched.com/event/F04J>