Java Serialization

Übersicht

- Einfache Möglichkeit Objekte zu speichern (ObjectOutputStream) bzw. einzulesen (ObjectInputStream)
- Ein oder mehrere Objekte in einem Stream
- Attribute werden mit aktuellem Wert geschrieben
- Beim Lesen wird das Objekt rekonstruiert ohne Konstruktor-Aufrufe
- Risikoreich

Gliederung

- Motivation
- Prinzipien
- Konsequenzen
- Best Practices

Motivation

- Im Frühjahr 2018 las ich David Svoboda: Exploiting Java Deserialization for Fun and Profit. Fand ich spannend.
- Bloch schreibt in Effective :
  When serialization was added to Java in 1997, it was known to be somewhat risky. The approach had been tried in a research language (Modula-3) but never in a production language. While the promise of distributed objects with little effort on the part of the programmer was appealing, the price was invisible constructors and blurred lines between API and implementation, with the potential for problems with correctness, performance, security and maintenance. Proponents believed the benefits outweighed the risk, but history has shown otherwise.
- Ein Java-Profi muss die Techniken kennen.

Prinzipien

```
Clazz obj = …
try(ObjectOutputStream oos = new ObjectOutputStream(“file“) {
      oos.writeObject(obj);
}
try(ObjectInputStream ois = new ObjectInputStream(“file“) {
      obj = (Clazz)ois.readObject();
}
```

Prinzipien

- Schlüsselwort transient: solche Attribute werden nicht serialisiert. Beispiel: `modcount` in den Iteratoren.
- `static final long SerialVersionUID` deklarieren. Wird beim Deserialisieren überprüft.
- Anpassbar mittels

```
private void writeObject(java.io.ObjectOutputStream out) throws
IOException
private void readObject(java.io.ObjectInputStream in) throws
IOException, ClassNotFoundException;
private void readObjectNoData() throws ObjectStreamException;
Object writeReplace() throws ObjectStreamException;
Object readResolve() throws ObjectStreamException;
```

- Exakte Signatur ist wichtig

Probleme

- Interna werden Bestandteil des APIs
- Spätere Änderungen praktisch unmöglich
- Attacken sind möglich
- Es gibt unterverschiedenen Gesichtspunkten bessere Möglichkeiten

Best Practices

- Avoid Serialization if possible
- If you are forced to write a serializable class, consider
  - Custom Serialized Form
  - Serialization Proxy

Best Practices - Custom Serialized Form

- Decide on the attribute that should be excluded from default (de)serialization and mark them `transient`
- Add the appropriate Javadoc tags @serial, @serialdata
- Write the necessary methods to customize the (de)serialization mechanism
- Write a private static class with a name like SerializationProxy
- Examples from the Java Collection Classes: ArrayList, LinkedList, …

Best Practices - Serialization Proxy

- Decide on the attribute the should be excluded from default (de)serialization and mark them `transient`
- Add the appropriate Javadoc tags @serial, @serialdata
- Write the necessary methods to customize the (de)serialization mechanism
- Examples from the Java Collection Classes: EnumSet, …

DoS Attack (Siehe Source Code im pub)

```
static byte[] payload() throws IOException {
  Set<Object> root = new HashSet<>();
  Set<Object> s1 = root;
  Set<Object> s2 = new HashSet<>();
  for (int i = 0; i < 100; i++) {
    Set<Object> t1 = new HashSet<>();
    Set<Object> t2 = new HashSet<>();
    t1.add("foo"); // make it not equal to t2
    s1.add(t1);
    s1.add(t2);
    s2.add(t1);
    s2.add(t2);
    s1 = t1;
    s2 = t2;
  }
  return serialize(root);
}
```