

Sichere Software vom Java-Entwickler

Dominik Schadow
Java Forum Stuttgart
05.07.2012

BASEL

BERN

LAUSANNE

ZÜRICH

DÜSSELDORF

FRANKFURT A.M.

FREIBURG I.BR.

HAMBURG

MÜNCHEN

STUTTGART

WIEN

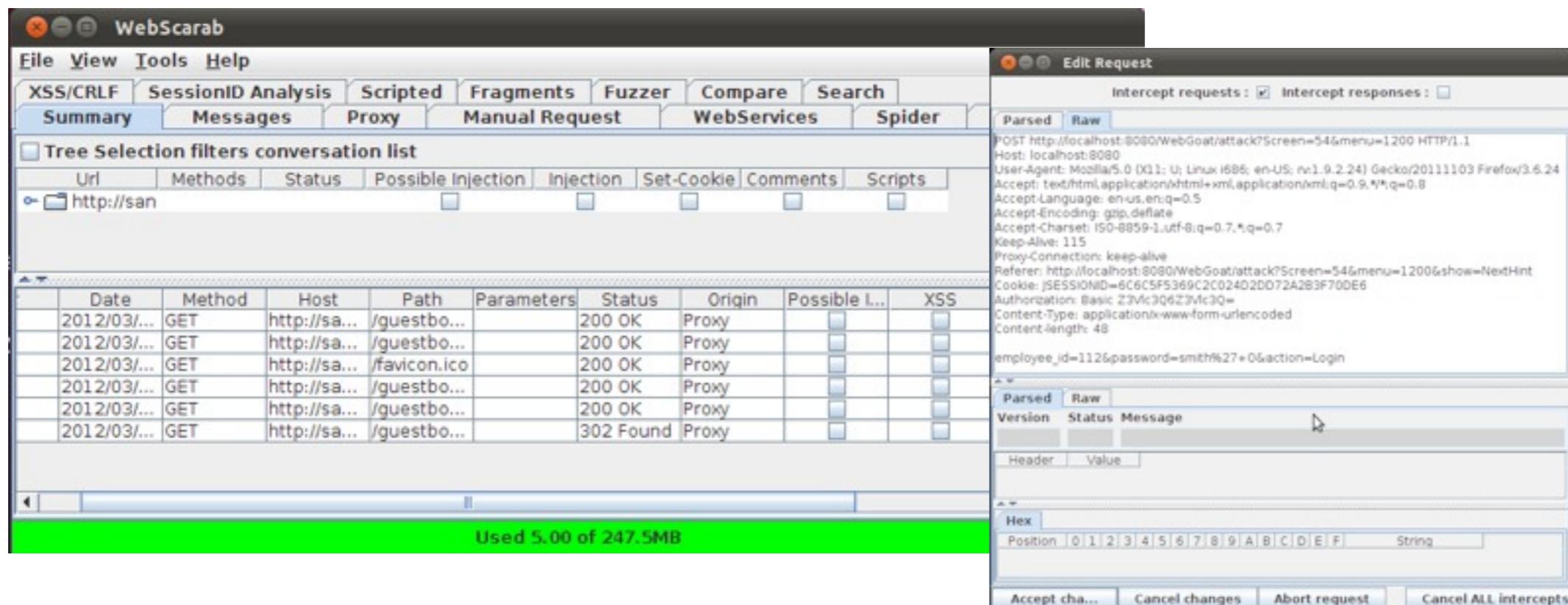


“We can no longer afford to tolerate relatively simple security problems like those presented in the OWASP Top 10.”

— OWASP

Secure software is not developed accidentally

- Applications must be protected from the beginning
 - Security fix does not bring back stolen data
 - Problem may be caused by the architecture
- 100% secure software will never exist
 - But we can stop making it that easy for attackers



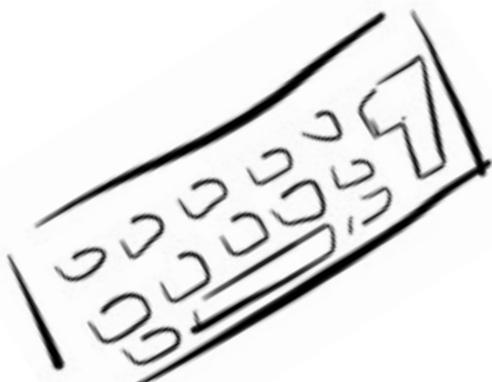
Open Web Application Security Project (OWASP)

Improving the security of (web) application software

- ▶ Not-for-profit organization since 2001
- ▶ Raise interest in secure development



- ▶ Top 10
- ▶ Cheat Sheets
- ▶ Development Guides
- ▶ ...



- ▶ Enterprise Security API (ESAPI)
- ▶ WebScarab
- ▶ WebGoat
- ▶ ...



2x

A1: Injection

A2: Cross-Site Scripting (XSS)

A3: Broken Authentication and Session Management

A4: Insecure Direct Object References

A5: Cross Site Request Forgery (CSRF)

A6: Security Misconfiguration

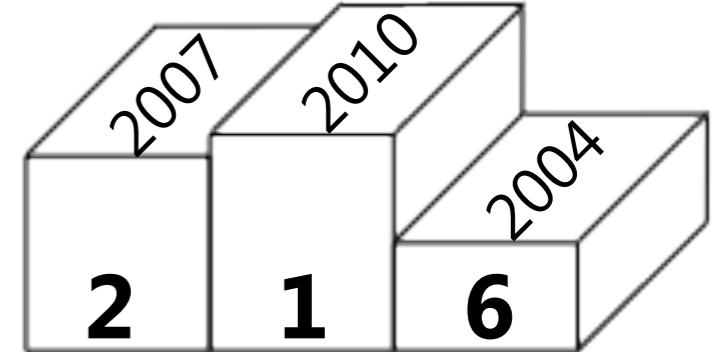
A7: Insecure Cryptographic Storage

A8: Failure to Restrict URL Access

A9: Insufficient Transport Layer Protection

A10: Unvalidated Redirects and Forwards

(1) Injection



■ The famous (and least necessary) **SQL injection**

■ Simple to avoid with **prepared statements**

- Use OR-Mapper like Hibernate or Spring JDBCTemplate
- Use it correctly

```
String query = "Select * from users where user_id = '" + userId + "'";  
PreparedStatement pstmt = con.prepareStatement(query);  
ResultSet rs = pstmt.executeQuery();
```

■ Limit database user permissions

■ **Other injections** (like LDAP injection, XPath injection)

■ **White list validation** for all user supplied input



(6) Security Misconfiguration

Some other guys job

- Server/ database configuration, firewall, user rights
 - Disable unnecessary features, services, ports, ...

Developer's job

- Configure logging/ exception handling
 - No technical errors in frontend
 - Never serve log over web application in production environment
- Framework security configuration
 - Security updates, new versions



(7) Insecure Cryptographic Storage

Most of the time, the problem is not the algorithm

- Data isn't protected at all
 - Identify and protect all sensitive data in all places
 - Never log any sensitive data (unencrypted)
- Real threats not identified
 - DB encryption protects data from DBA/ stolen disks, not SQL injection
- Use standards, never invent your own *'algorithm'*
- Prepare key exchange and revocation
 - Change keys periodically

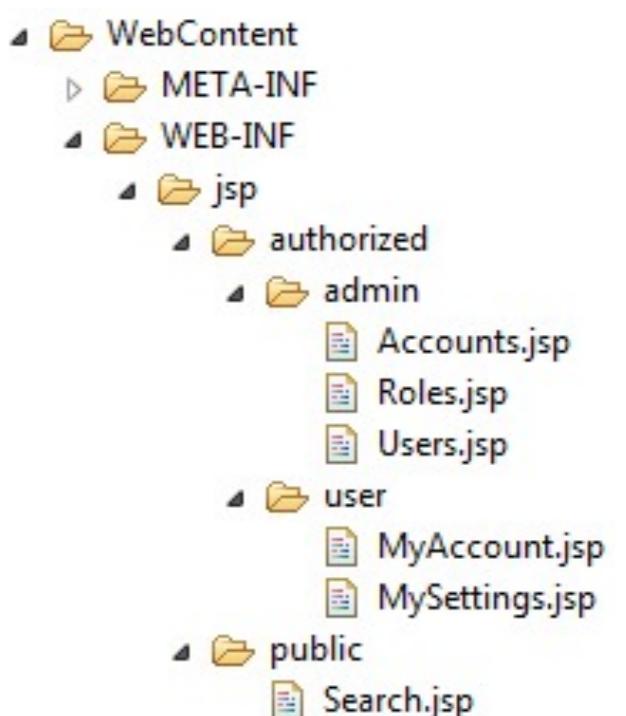


(8) Failure to Restrict URL Access



Think about roles from the beginning

- ▶ Store view files (JSP, JSF, ...) in different folders based on roles
- ▶ Makes filter configuration much easier



(10) Unvalidated Redirects and Forwards

Redirects send request to new external page

- ▶ **Target:** Phishing, pharming, malware installation

Forwards send request to new page in same application

- ▶ **Target:** Bypass authentication/ authorization checks

Avoid redirects and
forwards wherever
possible

Don't allow user
parameters in target
URL

User parameters in
target URL required



Validate final URL



Call access controller



A1: Injection

A2: Cross-Site Scripting (XSS)

A3: Broken Authentication and Session Management

A4: Insecure Direct Object References

A5: Cross Site Request Forgery (CSRF)

A6: Security Misconfiguration

A7: Insecure Cryptographic Storage

A8: Failure to Restrict URL Access

A9: Insufficient Transport Layer Protection

A10: Unvalidated Redirects and Forwards

(2) Cross-Site Scripting (XSS)

- Execute code in victim's browser
 - Steal user session, sensitive user data, ...
 - Redirect to phishing sites, malware installation, ...
- Different XSS types
 - Stored
 - Reflected 
 - DOM based

Often because of missing input validation

```
  
    <input type="image" src="javascript:alert('XSS');">  
        <b onmouseover=alert('XSS')>click me!</b>  
  
<body onload=alert('XSS')>
```



Input validation and output escaping for every input

- Input **validate** with a white list

- Output **escape**

- JSF implicitly escapes output

```
<h:outputText value="#{user.firstname}" escape="false" />
```

- ESAPI.encoder() provides different encoding methods

```
private void escapeOutput() {  
    String input = "<script>alert(12345)</script>";  
  
    String safeOutput = ESAPI.encoder().encodeForHTML(input);  
    // &lt;script&gt;alert&#x28;12345&#x29;&lt;&#x2f;script&gt;  
  
    safeOutput = ESAPI.encoder().encodeForJavaScript(input);  
    // \x3Cscript\x3Ealert\x2812345\x29\x3C\x2Fscript\x3E  
}
```



Prevent scripts from accessing cookie with http-only

```
<cookie-config>
    <!-- block script access to cookie -->
    <http-only>true</http-only>
    <!-- protect cookie transport -->
    <secure>true</secure>
</cookie-config>
```



(3) Broken Authentication and Session Management

One of the most complex security tasks to develop

Simply: Don't invent it again, use existing frameworks

- ▶ Spring Security <http://www.springsource.org/spring-security>
- ▶ Apache Shiro <http://shiro.apache.org>

Centralize: One library, one place

- ▶ Independent of authentication system (LDAP, AD, DB, ...)
- ▶ Know exactly how to use it

***HTTP is a stateless protocol -->
credentials (session id) are included in every request***



Protect all connections with authentication data with SSL

- Session id and credentials must be protected at all times
 - As valuable as username and password
 - Unprotected connection exposes session id
- ▶ Don't include session information (like session id) in URLs



@

Google



web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebXMLParameters" version="3.0">
    <display-name>WebXMLParameters</display-name>

    <session-config>
        <!-- soft session timeout -->
        <session-timeout>30</session-timeout>

        <cookie-config>
            <!-- block script access to cookie -->
            <http-only>true</http-only>
            <!-- protect cookie transport -->
            <secure>true</secure>
        </cookie-config>

        <!-- store JSESSIONID in cookie -->
        <tracking-mode>COOKIE</tracking-mode>
    </session-config>
</web-app>
```

(4) Insecure Direct Object References



1. User logs in with username/ password
URL is **https://www.fakesite.com/account?no=123456789**
2. User experiments with URL *no* parameter
URL is **https://www.fakesite.com/account?no=987654321**
3. User can view/ change other accounts

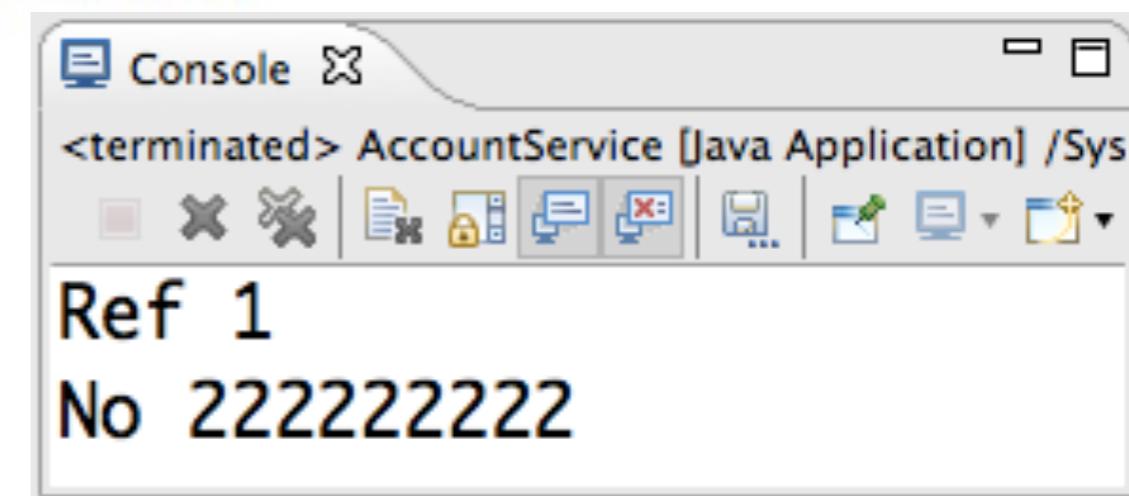


```
private Set<Object> accounts;
private Account accountA = new Account(111111111);
private Account accountB = new Account(222222222);
private Account accountC = new Account(333333333);
private Account accountD = new Account(444444444);

public AccountService() {
    accounts = new HashSet<Object>();
    accounts.add(accountA);
    // add all accounts
}

public void accessMap() throws AccessControlException {
    IntegerAccessReferenceMap map = new IntegerAccessReferenceMap(accounts);
    String indRef = map.getIndirectReference(accountB);
    System.out.println("Ref " + indRef);

    String mapRef = indRef; // e.g. accessed via request parameter
    Account account = (Account) map.getDirectReference(mapRef);
    System.out.println("No " + account.getNo());
}
```

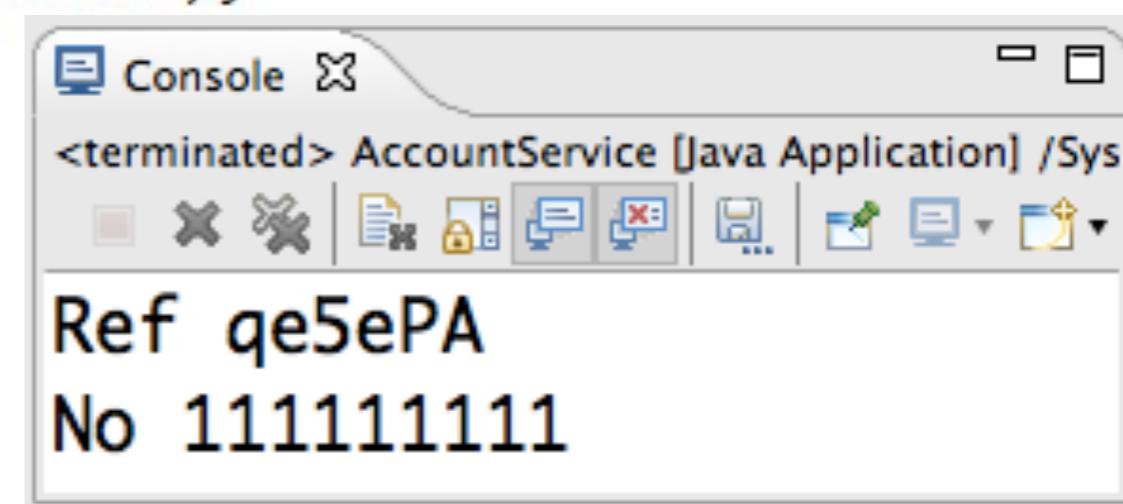


```
private Set<Object> accounts;
private Account accountA = new Account(111111111);
private Account accountB = new Account(222222222);
private Account accountC = new Account(333333333);
private Account accountD = new Account(444444444);
```

```
public AccountService() {
    accounts = new HashSet<Object>();
    accounts.add(accountA);
    // add all accounts
}
```

```
public void accessRandomMap() throws AccessControlException {
    RandomAccessReferenceMap map = new RandomAccessReferenceMap(accounts);
    String indRef = map.getIndirectReference(accountA);
    System.out.println("Ref " + indRef);

    String mapRef = indRef; // e.g. accessed via request parameter
    Account account = (Account) map.getDirectReference(mapRef);
    System.out.println("No " + account.getNo());
}
```



(5) Cross Site Request Forgery (CSRF)

- Browser **with authenticated user** must send credentials
 - Attacker causes request from his site to vulnerable application
 - e.g. with an invisible or <iframe>
 - User's credentials are used to execute attacker's request
- Often a vulnerable (standard) intranet application
 - Not accessible externally
 - Victim's browser inside corporate network is tricked into issuing commands



?

<SCRIPT>



Calculate and use a random secret token

- **Calculate a random secret token** at beginning of session
 - Calculate per request for higher security needs
 - Value not automatically submitted like session cookie
- **Add** this token as hidden field to **all** (critical) **forms**

```
<input type="hidden" name="csrfToken"  
value="928ce83948da9389eb9384019c38de8c"/>
```

- **Check** token before executing selected action



Develop your own secure form type for easy usage

- Create own form like ***SecureForm***
 - Adds token automatically
 - Easy (re)usable by developers
- Standard unprotected form still available

Configure session timeout in web.xml

```
<session-config>
    <session-timeout>60</session-timeout>
</session-config>
```



(9) Insufficient Transport Layer Protection

Correct SSL/TLS configuration is difficult

- ▶ Web-/application server administrator
 - Identify all routes where sensitive data is broadcasted
 - Protect all *(or nothing)*
 - Don't mix protected with unprotected content
 - Secure the input form for log-in credentials
 - Secure the session cookie
- less vulnerable
for Man-in-
the-Middle
attacks*



Use the HTTP Strict Transport Security (HSTS) IETF draft

```
HttpServletResponse response ...;  
response.setHeader("Strict-Transport-Security",  
    "max-age=8640000; includeSubdomains");
```

- Application forces browser to only use HTTPS when visiting
 - For specified time, renewed with every response
- Access blocked if communication is insecure
 - Invalid certificate --> error page (not strange warning dialog)
- Browser support required  
 - No backwards compatibility issues



Security is every developer's job

Developing with security awareness is a good start

- ▶ Every developer must know at least security basics
- ▶ One (senior) developer per team with deep security knowledge

Design security in from the beginning

- ▶ Think about security requirements before starting to code
- ▶ Much harder/ more expensive to secure an existing application

Security must be a natural part
of your development process

Resources

- OWASP www.owasp.org
- OWASP WebScarab https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project with Firefox QuickProxy <https://addons.mozilla.org/de/firefox/addon/quickproxy>
- ESAPI <http://esapi.org>
- Java Secure Coding Guidelines
<http://www.oracle.com/technetwork/java/seccodeguide-139067.html>
- Qualys SSL Labs <https://www.ssllabs.com>
- Preventing CSRF with JSF 2.0
<http://blog.eisele.net/2011/02/preventing-csrf-with-jsf-20.html>
- HTTP Strict Transport Security Header
<http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec>



Secure coding...

Visit us at our booth for
more (security) questions...

Trivadis GmbH
Dominik Schadow

Industriestrasse 4
D-70565 Stuttgart

Phone +49-711-903 63 230
Fax +49-711-903 63 259

dominik.schadow@trivadis.com
www.trivadis.com

BASEL

BERN

LAUSANNE

ZÜRICH

DÜSSELDORF

FRANKFURT A.M.

FREIBURG I.BR.

HAMBURG

MÜNCHEN

STUTTGART

WIEN

