Sichere Software-Entwicklung für Java Entwickler

Dominik Schadow
Senior Consultant
Trivadis GmbH
05/09/2012
AGENDA

1. OWASP and the top 10 project
2. A closer look at the current top 10
3. Raise interest in secure programming
Every developer needs secure programming knowledge

- Applications must be protected from the beginning
  - A security fix does not bring back stolen data
  - The problem may be caused by the architecture
    - Not fixable with a couple of simple code changes

- 100% secure software will never exist
  - But we can stop making it that easy for attackers
  - Secure software is not developed accidentally
    - Test web applications for vulnerabilities before deployment

If you don’t do this, the attackers will be happy to do it for you...
Improving the security of (web) application software

- Open Web Application Security Project (OWASP)
  - Not-for-profit worldwide charitable organization since 2001
  - All material available for free

- Top 10
  - Cheat Sheets to avoid most of the top 10 risks
  - Development guides

- **ESAPI** - OWASP Enterprise Security API
- **WebScarab** - analyze applications that communicate using HTTP(S)
- **WebGoat** - deliberately insecure JEE web application to teach web application security
Awareness for developers – the OWASP TOP 10 project

- Lists the 10 most critical web application security risks
  - Focus changed from weaknesses/vulnerabilities to risks in 2010
  - Not a security guide
  - Consider it as a starter

- There are more than 10 risks for web applications
  - Focus on secure development first and train all developers
  - Document secure coding conventions
  - Think about a Software Development Lifecycle (SDLC) later
The Enterprise Security API (ESAPI) addresses the top 10 risks

- Addresses the OWASP Top 10 risks
  - Good Java library, but project is not really active

- Easy to use open source web application security library
  - Collection of security building blocks, not a framework
  - Centralized access to all security related functionality
    - One access point for all security functionality
    - Much easier for developers

- Provides authentication, access control, input validation, output escaping, encryption, random numbers, ...

AGENDA

1. OWASP and the top 10 project
2. A closer look at the current top 10
3. Raise interest in secure programming
Top 10 2010

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

A1 – Injection

- The famous (and least necessary) **SQL injection**
  - Simple to avoid with **prepared statements**
    - Use an OR-Mapper like Hibernate
    - Use Spring JDBCTemplate
    - Dynamic queries may still be misused and made vulnerable
  - Limit database user permissions

- **Other injections** (like LDAP injection, XPath injection)
  - **White list validation** for all user supplied input

*Always validate in front- and backend*
A2 – Cross Site Scripting (XSS)

- Execute code in victim’s browser
  - Steal users’ session, sensitive data
  - Redirect to phishing sites
- Often injected due to missing input validation
  - `<script ...>`
  - `<... onclick="" ...>`
- Different XSS types
  - Stored
  - Reflected
  - DOM based

Basic browser protection – Internet Explorer 8 detects some patterns/ Firefox with NoScript
Server side attacks stored/ reflected, client side DOM based

- **Stored**
  - Injected code stored permanently on target servers
    - Often into a database via forum, guestbook, comment field, ...

- **Reflected**
  - Injected code is reflected off the web server
    - Search results, error messages, or other response which contain (parts of) the input

- **DOM based**
  - Attack payload is executed because of DOM environment modification in the victim’s browser
    - Page itself (HTTP response) does not change, only client side code
A2 – Cross Site Scripting (XSS) (cont’d.)

- Every time an application accepts user input
  - **Validate** all user supplied input with a white list
  - Output **escape** (output encode) all user supplied input

```java
private void escapeOutput() {
    String input = "<script>alert(12345)</script>";

    String safeOutput = ESAPI.encoder().encodeForHTML(input);
    // &lt;script&gt;alert(&#x28;12345&#x29;&lt;/script&gt;);

    safeOutput = ESAPI.encoder().encodeForJavaScript(input);
    // \\x3Cscript\\x3Ealert\\x2812345\\x29\\x3C/script\\x3E

    safeOutput = ESAPI.encoder().encodeForXML(input);
    // &amp;#x3c;script&amp;#x3e;alert&amp;#x28;12345&amp;#x29;&amp;#x3c;/script&amp;#x3e;

    safeOutput = ESAPI.encoder().encodeForXPath(input);
    // &lt;script&gt;alert&lt;#x28;12345&amp;#x29;&lt;/script&gt;
}
```
A2 – Cross Site Scripting (XSS) (cont’d.)

- Prevent scripts from accessing cookie with **http-only**
  - No session cookie theft and other session-based attacks

```xml
<cookie-config>
    <!-- block script access to cookie -->
    <http-only>true</http-only>
    <!-- protect cookie transport -->
    <secure>true</secure>
</cookie-config>
```
A3 – Broken Authentication and Session Management

- One of the most complicated parts to develop
  - Simply: Don’t invent it again, use existing frameworks
    - Apache Shiro [http://shiro.apache.org](http://shiro.apache.org)

- Centralize in one place and reuse code application wide
  - Try to use one library only
  - Know exactly how to use it

But: HTTP is a stateless protocol → credentials (session id) are included in every request
A3 – Broken Authentication and Session Management (cont’d.)

- Protect all connections with authentication data with TLS
  - Session id and credentials must be protected at all times
    - Session id is as valuable as username and password
    - Unprotected connection does expose the session id

- Don’t include session information (like session id) in URLs
  - Shows up in referrer and other logs
  - Included in copied links (send via email, twitter, ...)

- Make sure logoff/timeout completely destroys the session
Servlet specification 3.0 makes secure configuration easier

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <display-name>WebXMLParameters</display-name>
  <session-config>
    <!-- soft session timeout -->
    <session-timeout>30</session-timeout>
  </session-config>
  <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>
</web-app>
```
A4 – Insecure Direct Object References

- Presentation layer access control
  - User notices a direct reference in the URL
    - e.g. a file, account, database record, ...
  - No enforcement of these restrictions on server side

1. User 57894 logs in with username/ password
   URL is https://www.myfakewebsite.com/account?no=57894
2. User experiments with URL no parameter, e.g. 57895
   URL is https://www.myfakewebsite.com/account?no=57895
3. User can view/ change other accounts
Reference map samples with ESAPI

```java
private Set<Object> fileSet;
private File fileA = new File("/temp/dummyA.txt");
private File fileB = new File("/temp/dummyB.txt");
private File fileC = new File("/temp/dummyC.txt");
private File fileD = new File("/temp/dummyD.txt");

public FileService() {
    fileSet = new HashSet<Object>();
    fileSet.add(fileA);
    fileSet.add(fileB);
    fileSet.add(fileC);
    fileSet.add(fileD);
}

public void accessMap() throws AccessControlException {
    IntegerAccessReferenceMap map = new IntegerAccessReferenceMap(fileSet);
    String indRef = map.getIndirectReference(fileB);

    System.out.println("indRef " + indRef);

    String mapRef = indRef; // e.g. accessed via request parameter
    File file = (File) map.getDirectReference(mapRef);

    System.out.println("file " + file.getAbsolutePath());
}

public void accessRandomMap() throws AccessControlException {
    RandomAccessReferenceMap map = new RandomAccessReferenceMap(fileSet);
    String indRef = map.getIndirectReference(fileA);

    System.out.println("indRef " + indRef);

    String mapRef = indRef; // e.g. accessed via request parameter
    File file = (File) map.getDirectReference(mapRef);

    System.out.println("file " + file.getAbsolutePath());
}
```

indRef 3
file C:\temp\dummyC.txt

indRef hUDXFM
file C:\temp\dummyA.txt
A4 – Insecure Direct Object References (cont’d.)

- Replace the direct object references with an **access reference map** (indirect object references)
  - Replace account number with \( no=1, no=2, \ldots \) for current user
  - Mapping reference <-> real object on server **for this user**
    - Map is stored somewhere safe, e.g. session
  - No way for an attacker to break out
    - Using \( no=100 \) results in an error
    - Only resources in this map are accessible

- Useable for files, database records, accounts, ...
  - Use random numbers for more protection
A5 – Cross Site Request Forgery (CSRF)

- Often a vulnerable standard intranet (rarely web) application
  - Not accessible externally
  - Victim’s browser is tricked into issuing commands via XSS
    - Acts as a proxy

- Browser **with authenticated user** must send credentials
  - Attacker causes request to vulnerable application
    - Uses credentials to execute his own request
A5 – Cross Site Request Forgery (CSRF) (cont’d.)

- Calculate a random secret token at beginning of session
  - May not be automatically submitted like session cookie
  - Add this token as hidden field to all forms (and links)
    ```html
    <input name="token" value="abekdi1873843944" type="hidden"/>
    ```
  - Check token before executing selected action

- Configure a low soft session timeout
  - Makes attack more complicated, not impossible
A6 – Security Misconfiguration

- Some other guys job
  - Patches for app-/web-server, databases, operating system, ...
  - App-/web-server/ database configuration, firewall, user rights
    - Turn off unnecessary features, disable ports, services, ...

- Developer’s job
  - Inform admins about project requirements (document them)
  - Configure logging, exception handling
    - No technical errors in frontend
    - Never serve log over web application in a production environment
  - Framework security configuration
    - Security related settings in all used frameworks
    - Security updates, new library versions
A7 – Insecure Cryptographic Storage

- Most of the time, the problem is not the algorithm
  - The data isn’t protected at all
    - Identify and protect all sensitive data in all places
  - The real threats are not identified
    - DB encryption protects data from DBA/ stolen disks, not SQL injection

- Never log any sensitive data **unencrypted**

- Store key(s) and data in different locations
  - Prepare key exchange and revocation
  - Change keys periodically
How do I select a strong algorithm?

- Never invent your own algorithms
- There is more than just the algorithm name
  - Size, padding, mode, and don’t forget the salt
    - Symmetric
      - AES/CBC/PKCS5Padding with 192 bit, Blowfish
    - Asymmetric
      - RSA, DSA with > 1024 bit
    - Hash
      - SHA-256, RIPEMD-160
- Follow the news, replace weak algorithms in next project

if in doubt, choose the stronger key (negative impact on performance)
Encryption does not have to be complicated

```java
/**
 * Symmetric AES (CBC, 128 bits) encryption sample with Apache Shiro.
 */

private void encryptAndDecryptAES() {
    AesCipherService cipher = new AesCipherService();
    byte[] key = cipher.generateNewKey().getEncoded();

    byte[] encrypted = cipher.encrypt("Secure Programming rocks!".getBytes(), key).getBytes();
    System.out.println("Encrypted: " + asHex(encrypted));

    byte[] decrypted = cipher.decrypt(encrypted, key).getBytes();
    System.out.println("Decrypted: " + new String(decrypted));
}
```
A8 – Failure to Restrict URL Access

- Presentation layer access control
  - GUI only shows authorized buttons/ links/ ...
  - User notices his role in the URL and changes it
    - e.g. user, editor, admin, ...
  - No enforcement of these restrictions on server side

1. User 57894 logs in with username/ password
   URL is https://www.myfakewebsite.com/\textit{user}/account
2. User experiments with role part in URL, e.g. admin
   URL is https://www.myfakewebsite.com/\textit{admin}/account
3. User has access to other accounts
A8 – Failure to Restrict URL Access (cont’d.)

- Enforce all restrictions on server side
  - Access for authorized users only
- Think about roles from the beginning
  - Store view files (JSP, JSF, …) in different folders based on their roles
  - Makes role/ filter configuration much easier
- Avoid combining user and admin roles in one application
  - Public application with user role only accessible via internet
  - Separate admin application only accessible in the intranet
A9 – Insufficient Transport Layer Protection

- Identify all routes where sensitive data is broadcasted
- Correct SSL/TLS configuration is difficult
  - Ask an administrator
- 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
Some Secure Sockets Layer and Transport Layer Security basics

- **SSL v2** is insecure and must not be used
  - Disable it

- **SSL v3** and **TLS v1.0** are most common
  - Do not have any major security flaws up to now
  - TLS v1.0 is sometimes referred to as SSL v3.1

- **TLS v1.1** and **TLS v1.2** are the best selection
  - Do not have any security flaws up to now
  - Widely unsupported, choose in case server supports it
    - Older clients will automatically fall back to TLS v1.0

https://www.ssllabs.com
Set the HTTP Strict Transport Security (HSTS) header

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

- HTTP Strict Transport Security is currently an IETF draft
- Application forces browser to only use HTTPS when visiting
  - For specified time, renewed with every response
  - Access is blocked if communication is insecure
    - Invalid certificate results into error page, not a strange certificate warning dialog
- Browser support required, no backwards compatibility issues
  - Supported in Firefox and Chrome

A10 – Unvalidated Redirects and Forwards

- **Redirects** send request to different page
  - Often include user supplied parameters in destination URL
  - **Target:** Phishing and pharming (malware installation)

- **Forwards** send request to new page in same application
  - Sometimes include user supplied parameters in destination URL
  - **Target:** Bypass authentication/ authorization checks
A10 – Unvalidated Redirects and Forwards (cont’d.)

- **Avoid** redirects and forwards wherever possible
- **Don’t allow** user parameters for the target URL

- In case you need parameters in the target URL
  - Use a server side mapping to translate the values shown to the user into valid URL parts
    - That’s the access reference map from before...
  - **Validate the final target URL**
  - Call the access controller
AGENDA

1. OWASP and the top 10 project
2. A closer look at the current top 10
3. Raise interest in secure programming
Use tools to examine/ manipulate your web application (data)

with Firefox QuickProxy https://addons.mozilla.org/de/firefox/addon/quickproxy
The OWASP Broken Web Applications project

- Download the VM
- Run it with NAT virtual machine settings!!!
- Launch your host’s browser with the IP address shown

https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project
One security aware developer is not enough

- Developing with security awareness is a good start
  - Make sure the environment is configured properly
  - Inform administrators about your requirements

- Design security in from the beginning
  - Think about security needs before starting to code
  - Much harder/more expensive to secure an existing application

**Security must be a natural part of the development process**
THANK YOU.

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
Resources

- OWASP  [www.owasp.org](http://www.owasp.org)
  - Cheat Sheets


- ESAPI  [http://esapi.org](http://esapi.org)


- Qualys SSL Labs  [https://www.ssllabs.com](https://www.ssllabs.com)