OWASP TOP 10

2010

Intro

- Web App security isn't limited to 10 Vuln
- I choose 10 because these are the main ones (according to OWASP)
- Notes based on OWASP's recommendation
- Risk based approach
- Less attack -- more patch
- Aim: Converting Insecure app to Secure App

Top 10 security risks

1. Code Injection (server side)
2. XSS (client side)
3. Broken Authentication and Session
4. Insecure Direct Object References
5. CSRF
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Unrestricted URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

1 - Code Injection

- Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query.

- The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.

Type of Injection

- SQL
- OS command
- LDAP
- …
- etc

Example

```
String query = "SELECT account_balance FROM user_data WHERE user_name = "
  + request.getParameter("customerName");

try {
        Statement statement = connection.createStatement( … );
        ResultSet results = statement.executeQuery( query );
}
```

```
$command = 'type ' . $_POST['username'];

exec($command, $res);


for ($i = 0; $i < sizeof($res); $i++)

        echo $res[$i].'<br>';
```

2 - Cross Site Scripting (XSS)

- XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping.

- XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Type

- Reflected
  - Temporary
- Stored
  - Permanent

Example

```
<% String eid = request.getParameter("eid"); %>
...
Employee ID: <%= eid %>
```

```
<%...
 Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery("select * from emp where id="+eid);
 if (rs != null) {
  rs.next();
  String name = rs.getString("name");
%>

Employee Name: <%= name %>
```

Common context

- HTML
- Attributes
- URL parameter
- CSS
- Javascript

Context	Code Sample
HTML Body	UNTRUSTED DATA
Safe HTML Attributes	<input type="text" name="fname" value="UNTRUSTED DATA">
GET Parameter	clickme
Untrusted URL in a SRC or HREF attribute	clickme <iframe src="UNTRUSTED URL" />
CSS Value	Selection
JavaScript Variable	<script>var currentValue='UNTRUSTED DATA';</script> <script>someFunction('UNTRUSTED DATA');</script>

3 - Broken Authentication and Session

- Application functions related to authentication and session management are often not implemented correctly.

- Allow attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities.

Example

- 1: Putting session IDs in the URL:

```
http://example.com/sale/saleitems;
jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV
?dest=Hawaii
```

An authenticated user e-mails the above link to his friend. When his friends use the link, they will use his session and credit card.

Example (cont.)

- 2: Application's timeouts aren't set properly.

User, instead of selecting "logout", simply closes the browser tab and walks away. Attacker uses the same browser an hour later, and that browser is still authenticated.

Example (cont.)

- 3: User passwords are not encrypted .

Insider or external attacker gains access to the system’s password database, learns everyone's password.

4 - Insecure Direct Object References

- A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key.

- Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

Why it happens

- Reference to internal application implementation is exposed

- Use of predictable sequence

- Example:

```
http://example.com/app/accountInfo?acct=notmyacct
```

```
http://www.example.com/application?file=1
```

5 - Cross Site Request Forgery (CSRF)

- A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application.

- This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

Example

- Example request by an authenticated user:

```
http://example.com/app/transferFunds?
amount=1500&destinationAccount=4673243243
```

By embedding the following code inside a website, the transfer will occur if an authenticated user of example.com visits the page

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#" width="0"
height="0" />
```

Why/How this works

- The is no difference (to the server) where the code originated from
- The code is triggered with a valid session

6 - Security Misconfiguration

- Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. This includes keeping all software up to date, including all code libraries used by the application.

- Effect could range from single information disclosure to remote code execution

Example

1. Using a framework version that is known to be vulnerable to certain attack

Apache **Struts** : List of security **vulnerabilities**
www.cvedetails.com/**vulnerability**-list/...id.../Apache-**Struts**.html
Security **vulnerabilities** of Apache **Struts** : List of all related cve security **vulnerabilities**. CVSS Scores, **vulnerability** details and links to full cve details and **...**

Apache **Struts** version 2.0.2 : Security **vulnerabilities**
www.cvedetails.com/**vulnerability**.../Apache-**Struts**-2.0.2.html
Security **vulnerabilities** of Apache **Struts** version 2.0.2 List of cve security **vulnerabilities** related to this exact version. You can filter results by cvss scores, years **...**

Apache **Struts** version 1.2.8 : Security **vulnerabilities**
www.cvedetails.com/**vulnerability**.../Apache-**Struts**-1.2.8.html
Security **vulnerabilities** of Apache **Struts** version 1.2.8 List of cve security **vulnerabilities** related to this exact version. You can filter results by cvss scores, years **...**

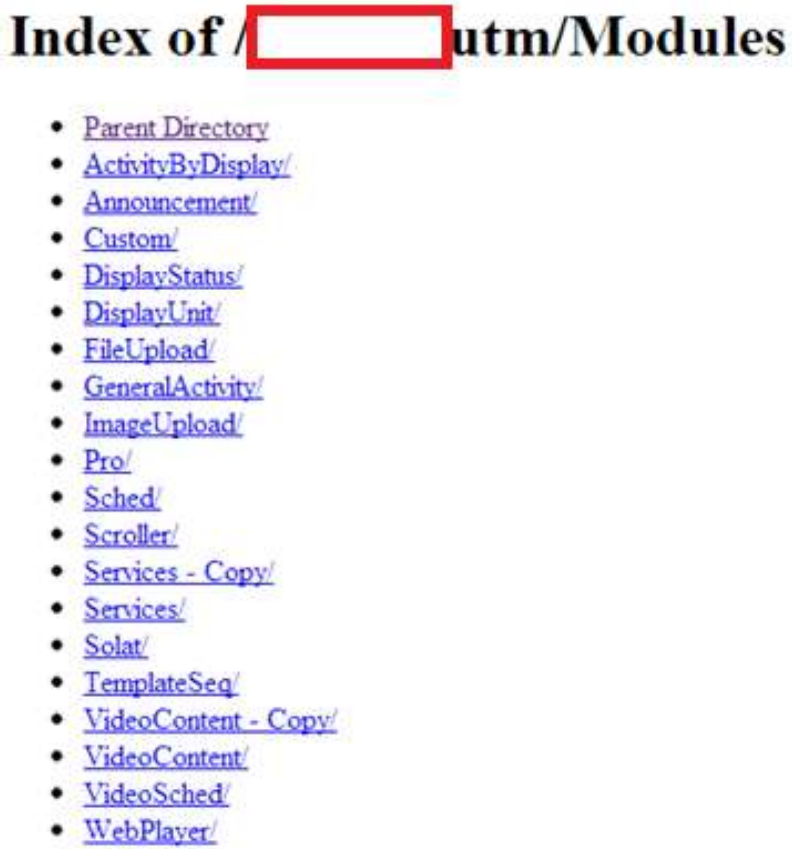
Apache **Struts** version 1.2.7 : Security **vulnerabilities**
www.cvedetails.com/**vulnerability**.../Apache-**Struts**-1.2.7.html
Security **vulnerabilities** of Apache **Struts** version 1.2.7 List of cve security **vulnerabilities** related to this exact version. You can filter results by cvss scores, years **...**

Example (cont.)

2. Having App server admin control running with default settings.

Example (cont.)

3. Directory listing not disabled



Example (cont.)

4. Errors shown to user

JSP Processing Error

HTTP Error Code: 404

Error Message:

```
JSPG0036E: Failed to find resource /cardcenter/common/interstitial.jsp
```

Root Cause:

```
java.io.FileNotFoundException: JSPG0036E: Failed to find resource /cardcenter/common/interstitial.jsp
        at com.ibm.ws.jsp.webcontainerext.AbstractJSPExtensionProcessor.findWrapper(AbstractJSPExtensionProcessor.java:322)
        at com.ibm.ws.jsp.webcontainerext.AbstractJSPExtensionProcessor.handleRequest(AbstractJSPExtensionProcessor.java:284)
        at com.ibm.ws.webcontainer.webapp.WebApp.handleRequest(WebApp.java:3548)
        at com.ibm.ws.webcontainer.webapp.WebGroup.handleRequest(WebGroup.java:269)
        at com.ibm.ws.webcontainer.WebContainer.handleRequest(WebContainer.java:818)
        at com.ibm.ws.wswebcontainer.WebContainer.handleRequest(WebContainer.java:1478)
        at com.ibm.ws.webcontainer.channel.WCChannelLink.ready(WCChannelLink.java:126)
        at com.ibm.ws.http.channel.inbound.impl.HttpInboundLink.handleDiscrimination(HttpInboundLink.java:458)
        at com.ibm.ws.http.channel.inbound.impl.HttpInboundLink.handleNewInformation(HttpInboundLink.java:387)
        at com.ibm.ws.http.channel.inbound.impl.HttpICLReadCallback.complete(HttpICLReadCallback.java:102)
        at com.ibm.ws.tcp.channel.impl.AioReadCompletionListener.futureCompleted(AioReadCompletionListener.java:165)
        at com.ibm.io.async.AbstractAsyncFuture.invokeCallback(AbstractAsyncFuture.java:217)
        at com.ibm.io.async.AsyncChannelFuture.fireCompletionActions(AsyncChannelFuture.java:161)
        at com.ibm.io.async.AsyncFuture.completed(AsyncFuture.java:136)
        at com.ibm.io.async.ResultHandler.complete(ResultHandler.java:196)
        at com.ibm.io.async.ResultHandler.runEventProcessingLoop(ResultHandler.java:792)
        at com.ibm.io.async.ResultHandler$2.run(ResultHandler.java:881)
        at com.ibm.ws.util.ThreadPool$Worker.run(ThreadPool.java:1497)
```

7 - Insecure Cryptographic Storage

- Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing.

- Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.

Problem

- Unsalted password hash
- Using base64 to encode sensitive parameter
- Using weak algorithms for encryption/hash
- Hard coding encryption/decryption key
- Using self-created hash/encryption algorithm

Example

- Password hash found in DB dump:

```
,'fc646ab58bc3535f15cebaf9caa144e6',
```

- This is an unsalted MD5 password. A quick Google will yield that the password.

md5(fadhil) = fc646ab58bc3535f15cebaf9caa144e6

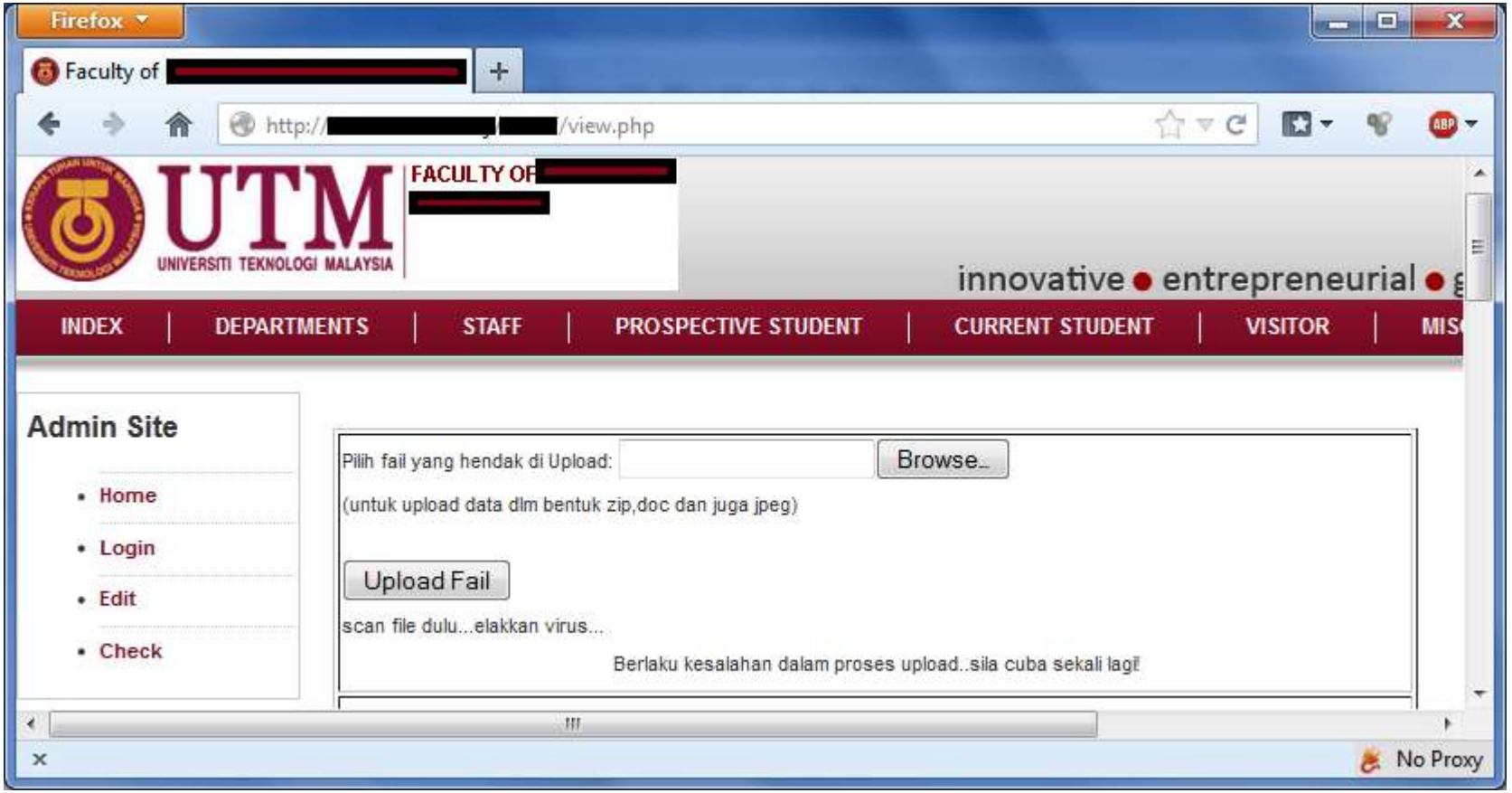
8 - Unrestricted URL Access

- Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks each time these pages are accessed.

- Attackers will be able to forge URLs to access these hidden pages anyway.

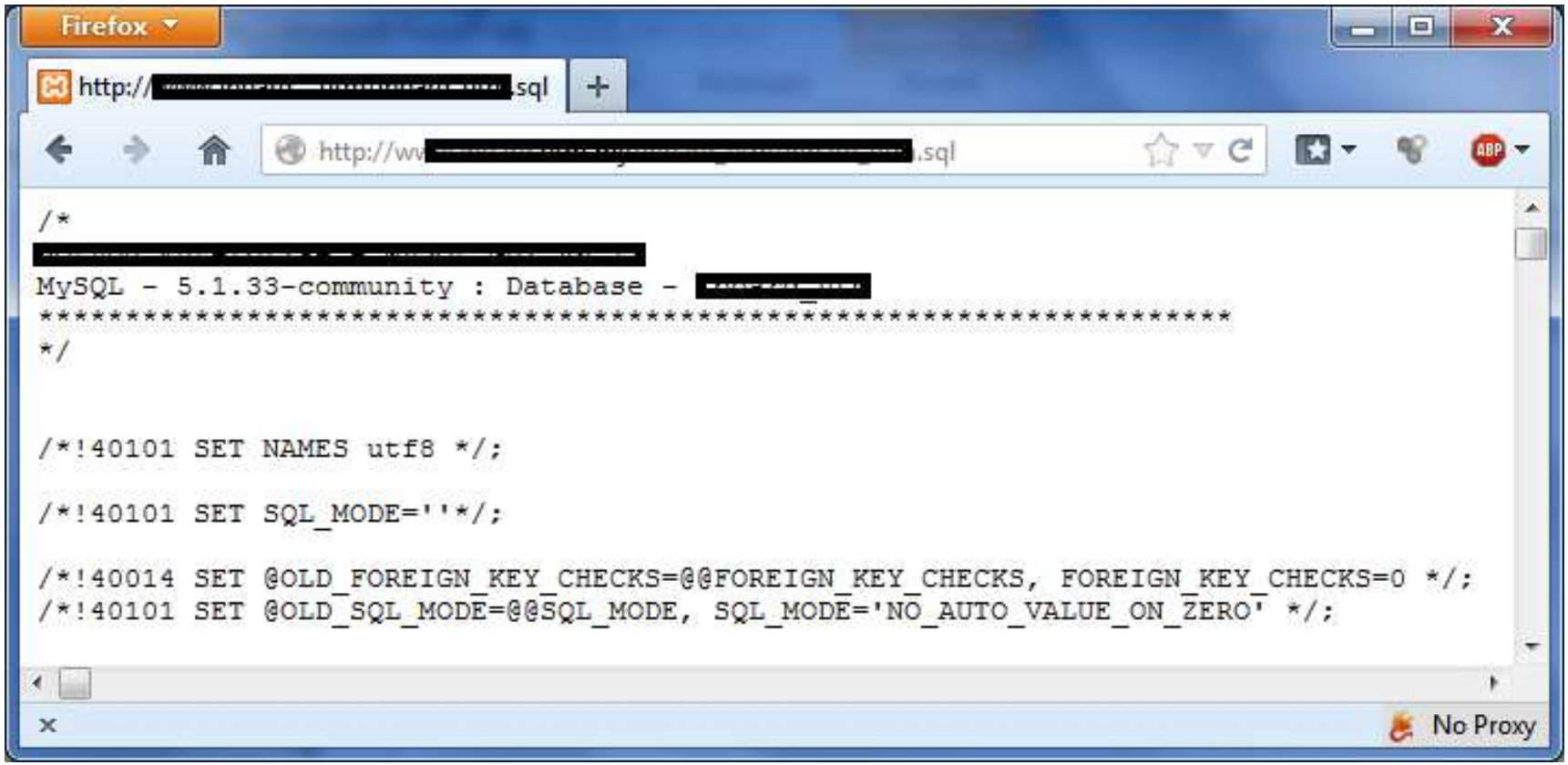
Example

- Access to admin page by guessing URLs



Example (cont.)

- Access/Download of SQL Dump file



Why it happens

- No proper session and permission check
- Predictable URL pattern
- Having the assumption that people will never know the URL because it is not shown on normal user page

9 - Insufficient Transport Layer Protection

- Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic.

- When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly.

Example

- Login page didn't use SSL
- Use of Self-signed / invalid / improperly configured certificated/server.

Why it is bad ?

- Login page with no SSL can leak sensitive information
- Use of self-signed/invalid certificate defeats the purpose of having a cert since the user isn't able to tell if his/her traffic is being intercepted

10 - Unvalidated Redirects and Forwards

- Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages.

- Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Examples

- example.com/go.php?url=
  example.com/ie/ie40/download/?
- example.com/search?q=user+search+keywords&url
  =
- example.com/coupon.jsp?code=ABCDEF&url=
  example.com/cs.html?url=
- proxy.example.com/?url=
- example.com/login?url=
- example.com/cgi-bin/redirect.cgi?

How it is exploited

- Example:
  - http://bank.com/go.php?url=

- Attack:
  - http://bank.com/go.php?url=**http://hack.er/login**

Why it is bad ?

- It facilitates phishing attacks against your site (gave it a more convincing look)

- Enables attackers to lure visitors into downloading malicious file