Introduction to

Web Application Security

Application security

- Creating a software component is easy (framework, component re-use, library, etc.)
- But securing this component is hard
  ➡ you must consider everything the attacker has in mind

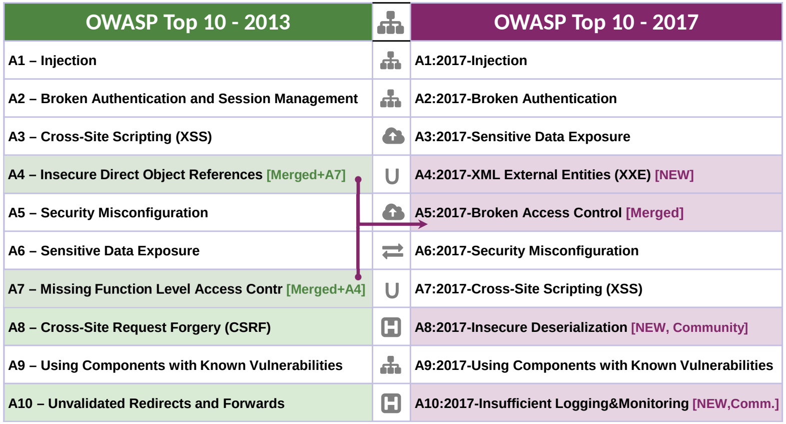
- Large attack surface
  - Application layer ⬅
  - System/server layer (kernel weakness, key-logger, etc.)
  - Network layer (sniffing, etc.)
  - User layer (phishing, etc.)

Web application security

- Web application are multi-tiered architecture
  ➡ security flaws may appear at many levels
- Despite that, they are associated to strong security requirements
  - Authentication
  - Authorization
  - Confidentiality
  - Integrity
  - Non-repudiation

https://www.owasp.org

OWASP Top Ten



OWASP Top 10 - 2013	🖧	OWASP Top 10 - 2017
A1 – Injection	🖧	A1:2017-Injection
A2 – Broken Authentication and Session Management	🖧	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	☁	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	☁	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	⇄	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	🄷	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	🖧	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	🄷	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

Injection

The man enters in ______ cinema.

Injection

The man enters in ______ cinema.

the ✓

Injection

The man enters in ______ cinema.

the ✓

brings ✗

Injection

The man enters in ______ cinema.

the ✓

brings ✗

the main ✓

Injection

The man enters in ______ cinema.

the ✓

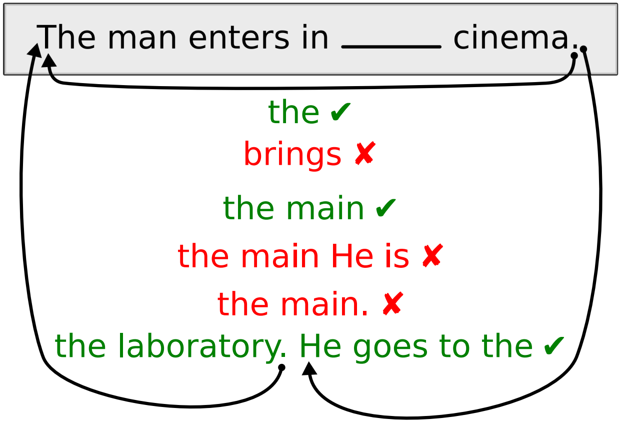
brings ✗

the main ✓

the main He is ✗

the main. ✗

Injection


The man enters in ______ cinema.

the ✓

brings ✗

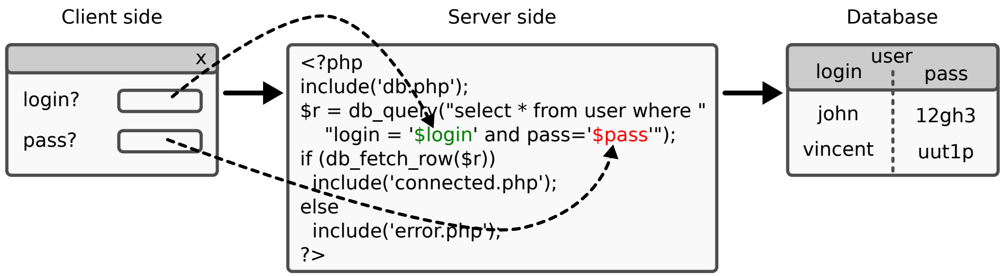
the main ✓

the main He is ✗

the main. ✗

the laboratory. He goes to the ✓

SQL Injection



```
<?php
include('db.php');
$r = db_query("select * from user where "
    "login = '$login' and pass='$pass'");
if (db_fetch_row($r))
  include('connected.php');
else
  include('error.php');
?>
```

user	
login	pass
john	12gh3
vincent	uut1p

Client inputs	Request	Result
random ; 123	select * from user where login = 'random' and pass='123'	error
john ; 12gh3	select * from user where login = 'john' and pass='12gh3'	connected
yes ; x' or '1'='1	select * from user where login = 'yes' and pass='x' or '1'='1'	connected!!!

SQL Injection

- Consider this vulnerable code:

```
http://library.com/dispatcher?action=list&id=1234567
$id = $_GET["id"];
$res = query("select * from books where id='" + $id + "'");
```

- How to list all books and users?

- Server constructs a SQL request with user data, on the fly
- The badly crafted user data change the semantic of the request
- Loss of confidentiality, authentication of integrity

SQL Injection

- Security measures:
  - Parameterized SQL statements
  - Stored procedures
  - Escape user-input
  - Whitelisting
  - Privilege principle
- Beware, the ultimate solution does not exist

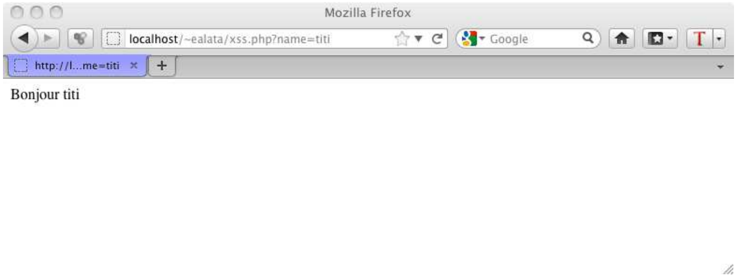
```
$res = query("select * from sensors_" + $_GET["num"]);
```

XSS Injection

- Cross-Site Scripting

```
<html><body>
<?php   if ($_GET["name"]) { print("Hi " . $_GET["name"]);
        } else             { print("Who are you?"); } ?>
</body></html>
```

http://server/xss.php?name=titi



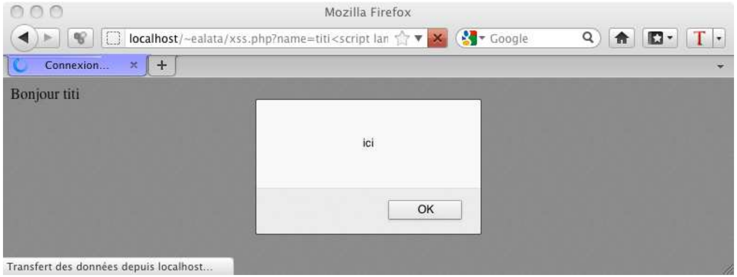
Bonjour titi

XSS Injection

- Cross-Site Scripting

```
<html><body>
<?php   if ($_GET["name"]) { print("Hi " . $_GET["name"]);
        } else             { print("Who are you?"); } ?>
</body></html>
```

http://server/xss.php?name=titi<script>alert('ici');</script>



Bonjour titi

ici

Transfert des données depuis localhost...

XSS Injection

- Attacker manages to modify the semantic of the server response
- Usually, the modified response includes a malicious script
- Script has access to all DOM structure
- Objectives: retrieve the cookie, get page content, etc.
- Reflected XSS / Stored XSS

- Server may inhibit the meaning of special characters
  <script> ➡ &#60;script&#62;
- HTTP header: Content-Security-Policy, X-XSS-Protection, etc.

CSRF

- Cross-Site Request Forgery: close to XSS attack
- Force an authenticated user to execute a request
- The server cannot distinguish beetwen a legitimate request and the forced request

- Alice uses a browser
- She is connected to its bank account and to its mail account
- She receives a mail from Eve with this content:

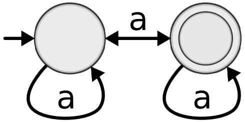
```
<img src="http://bank.com/transfer.do?to=eve&amount=100000"/>
```

CSRF

- The REFERER header identifies the site behind the request
- Make request distinguishable
  - Add a random token inside the legitimate webpage
  - Change this token value at every access of the webpage
  - Every critical user request must contain this token
  - Third party website (maybe malicious) cannot guess this value
- Add challenge/response (like captcha) on critical functions

ReDoS

- Regular Expression Denial of Service
- Takes benefit of algorithmic complexity of regexp matching



- Input `ab` ⇨ 2 paths
- Input `aab` ⇨ 4 paths
- Input `aaab` ⇨ 8 paths

- Choose the right library!

Unvalidated Input

- User can tamper with any part of his HTTP request
- Attacker can downalod a page and change its content

```
<form method="POST" action="buy.php">
  <input type="hidden" name="price" value="10.00">
  <input name="number" value="1">
  <input type="submit">
</form>
```

- Client-side validation is bad from a security point of view
- Never trust client side data and inputs

Others vulnerabilities

- Broken Authentication
- Http Parameter Pollution
- OS-Command
- Directory traversal
- ...