web coding security

Huỳnh Hải Âu
Công ty ISePRO

Đơn vị tổ chức:

Đơn vị tài trợ:

Contents

- SQL Injection
- XSS
- File upload

SQL Injection

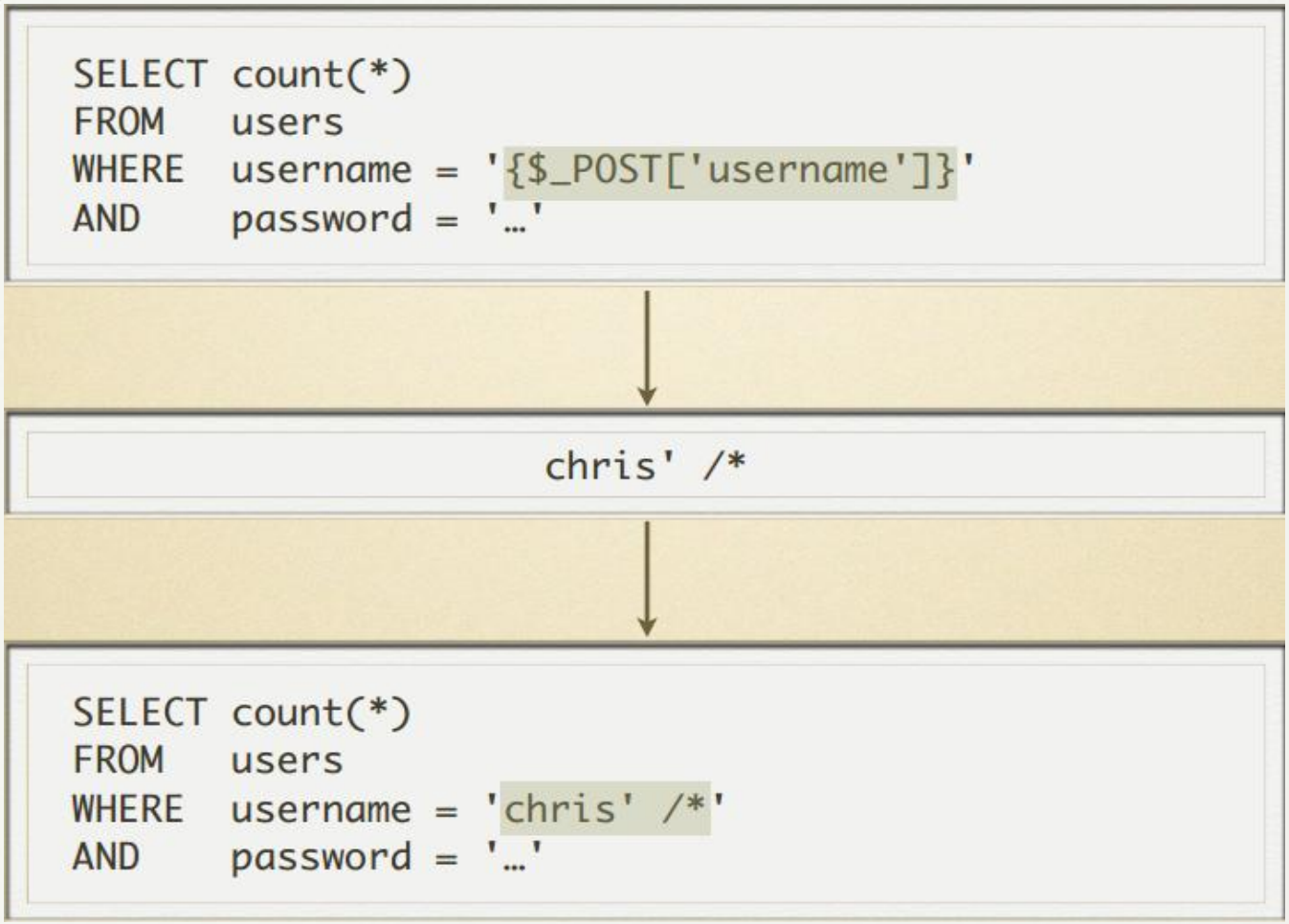
- Introduction
- Bad codes
- Preventing solutions

Conference &Training
Cần Thơ
SECURITY BOOTCAMP
2013
www.lactien.com

What is SQL Injection?

SQL injection is a technique used to take advantage of non-validated input vulnerabilities to pass SQL commands through a Web application for execution by a backend database

SQL injection is a basic attack used to either gain unauthorized access to a database or to retrieve information directly from the database


SELECT count(*)
FROM users
WHERE username = '{$_POST['username']}'
AND password = '…'
chris' /*
SELECT count(*)
FROM users
WHERE username = 'chris' /*'
AND password = '…'

- Types: 3 types
  - Union base
    - Inject a union query to extract data from database
  - Error base
    - Inject SQL characters, queries to make the application query fail and raise errors
    - Use sql errors to extract data from database
  - Blind SQLi
    - Inject sql characters to ask the database true or false questions and determines the answer based on the applications response

- Bad code 1:

```
if(isset($_POST['user']) && isset($_POST['password']))
{
	$u = $_POST['user'];
	$p = $_POST['password'];
	$u = preg_replace('/union|select|from|where|and|or/i','',$u);
	$p = preg_replace('/union|select|from|where|and|or/i','',$p);
	$q = mysql_query("select * from user where username='$u' and
password='$p'");
	if($r=mysql_fetch_assoc($q))
	{
		echo "<br>Log in successfully !";
		echo "<br>Hello $r[username]";
	}
	else
		echo "<br>Invalid login !";
}
```

– Bad code 2:

```
if(isset($_POST['user']) && isset($_POST['password']))
{
	$u = $_POST['user'];
	$p = $_POST['password'];
	$u = preg_replace( “/’ /i” ,” ” ,$u);
	$p = preg_replace( “/’ /i” ,” ” ,$p);
	$q = mysql_query("select * from user where username='$u' and
password='$p'");
	if($r=mysql_fetch_assoc($q))
	{
		echo "<br>Log in successfully !";
		echo "<br>Hello $r[username]";
	}
	else
		echo "<br>Invalid login !";
}
```

– Bad code 3:

```
if(isset($_GET['id']))
{
	$id = $_GET['id'];
	$id = mysql_real_escape_string($id);
	$q = mysql_query("select * from user where id=$id");
	if($r=mysql_fetch_assoc($q))
	{
		echo "<br>Profile: $r[username]";
		echo "<br>Age: $r[Age]";
		echo "<br>Phone: $r[Phone]";
		echo "<br>Mail: $r[Mail]";
		echo "<br>Address: $r[Address]";
	}
	else
		echo "<br>Invalid id !";
}
```

- Bad code 4:

```
if(isset($_POST['user']) && isset($_POST['password']))
{
	$u = mysql_real_escape_string($_POST['user']);
	$p = mysql_real_escape_string($_POST['password']);

	$p = hash("whirlpool", $p, true);
	$q = mysql_query("select * from user where username='$u' and password='$p'");

	if($r=mysql_fetch_assoc($q))
	{
		echo "<br>Log in successfully !";
		echo "<br>Hello $r[username]";
	}
	else
		Echo "<br>Invalid login !";
}
```

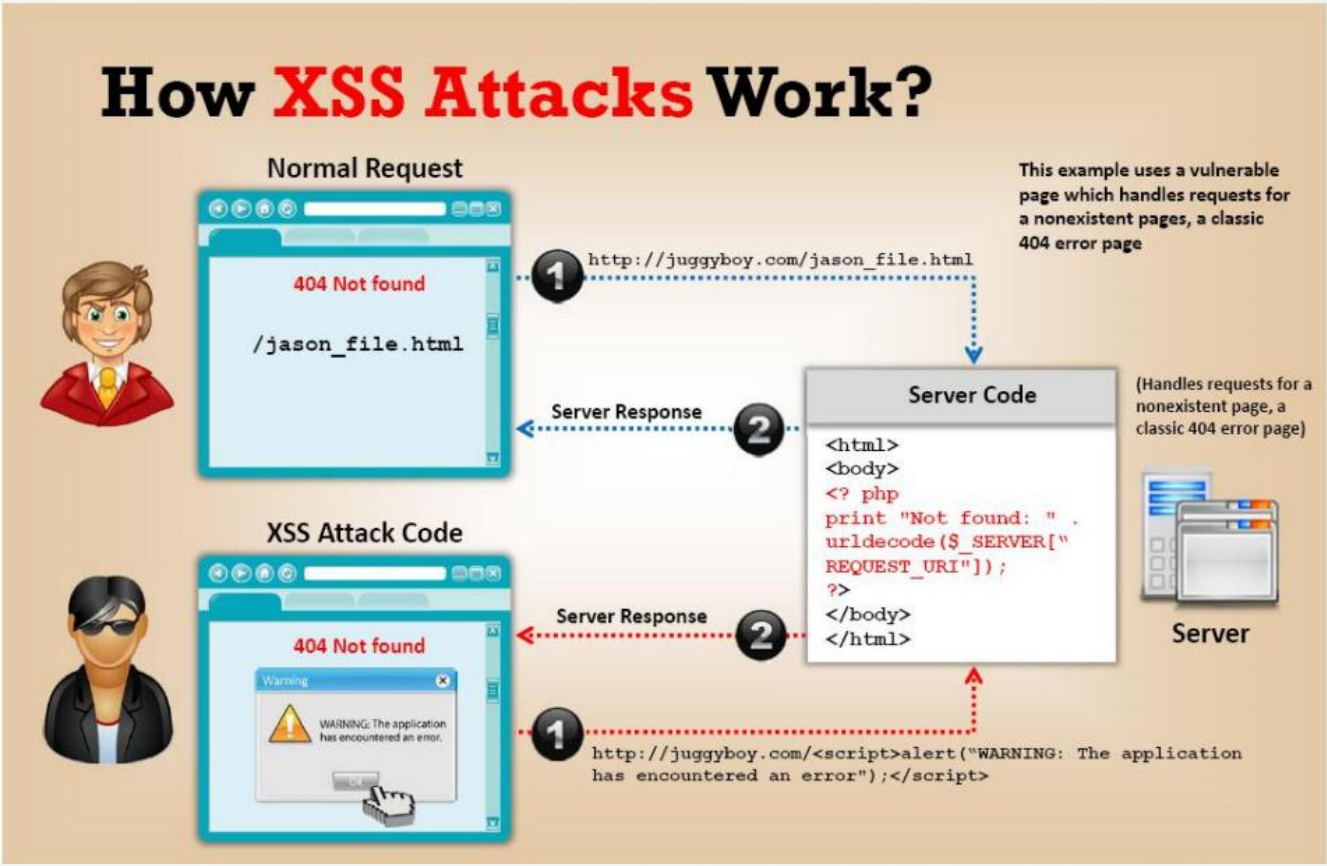
- Preventing solutions:
  - Prepare statement
    - aka parameterized statement
    - Template of sql query structure
      - INSERT INTO PRODUCT (name, price) VALUES (?, ?)
    - Separation of control flow & data flow

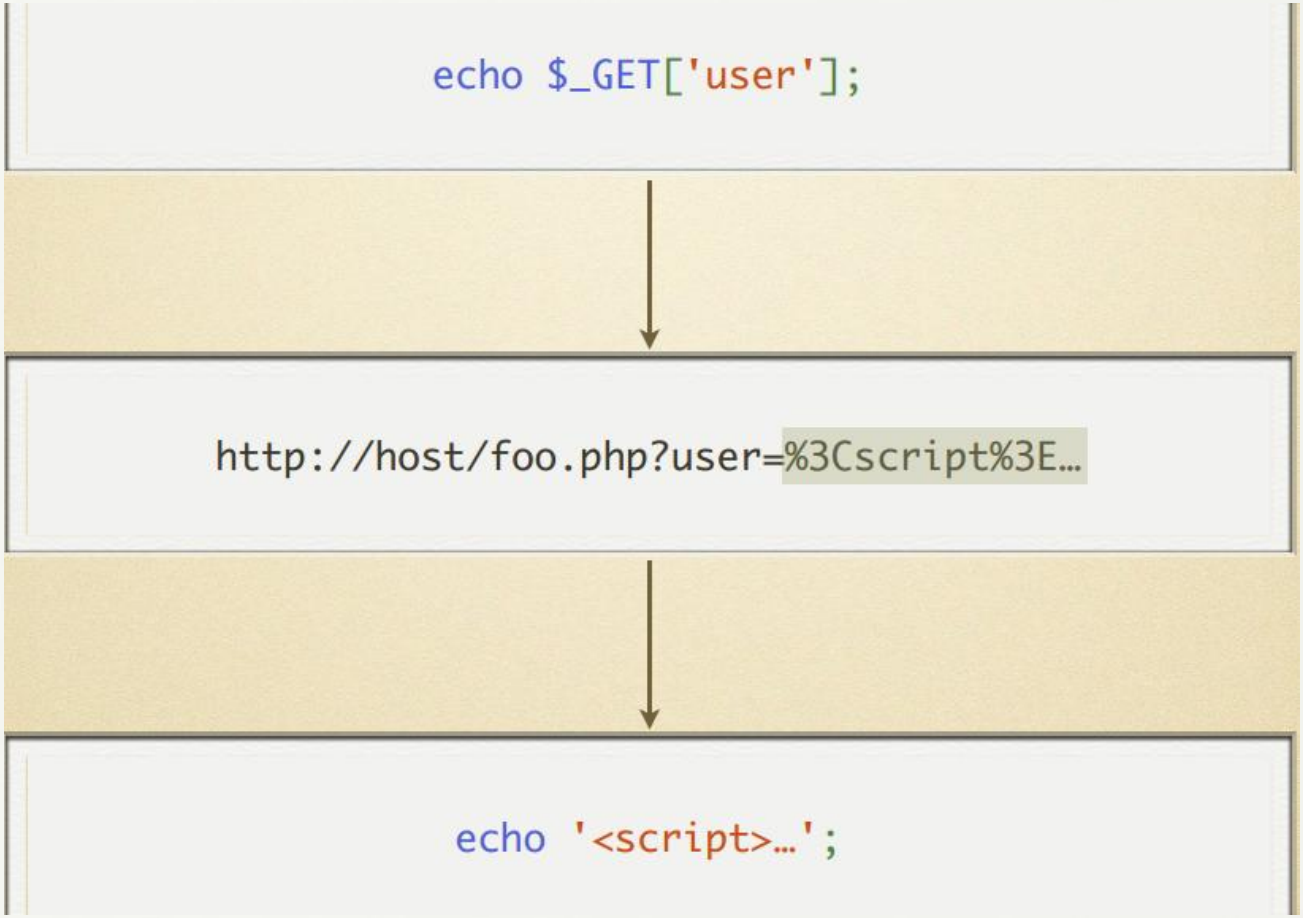

```
if(isset($_POST['user']) && isset($_POST['password']))
{
        $u = $_POST['user'];
        $p = $_POST['password'];

        $q = $sql->prepare("select * from user where username=? and password=?");
        $q->bind_param("ss", $u, $p);
        $q->execute();
        $res = $q->get_result();

        if($r=$res->fetch_assoc())
        {
                echo "<br>Log in successfully !";
                echo "<br>Hello $r[username]";
        }
        else
                echo "<br>Invalid login !";
        $q->close();
}
```

Cross Site Scripting (XSS)

- Introduction
- Bad codes
- Preventing solutions


Cross-Site Scripting (XSS) Attacks
Cross-site scripting ('XSS' or 'CSS') attacks exploit vulnerabilities in dynamically generated Web pages, which enables malicious attackers to inject client-side script into web pages viewed by other users
It occur when invalidated input data is included in dynamic content that is sent to a user's web browser for rendering
Attackers inject malicious JavaScript, VBScript, ActiveX, HTML, or Flash for execution on a victim's system by hiding it within legitimate requests
Malicious script execution
Redirecting to a malicious server
Exploiting user privileges
Ads in hidden IFRAMES and pop-ups
Data manipulation
Session hijacking
Brute force password cracking
Data theft
Intranet probing
Keylogging and remote monitoring


How XSS Attacks Work?
Normal Request
404 Not found
/jason_file.html
1
http://juggyboy.com/jason_file.html
This example uses a vulnerable page which handles requests for a nonexistent pages, a classic 404 error page
Server Response
2
Server Code
<html>
<body>
<? php
print "Not found: " .
urldecode($_SERVER["
REQUEST_URI"]);
?>
</body>
</html>
(Handles requests for a nonexistent page, a classic 404 error page)
Server
XSS Attack Code
404 Not found
Warning
WARNING: The application has encountered an error.
Server Response
2
1
http://juggyboy.com/<script>alert("WARNING: The application has encountered an error");</script>


echo $_GET['user'];
http://host/foo.php?user=%3Cscript%3E…
echo '<script>…';

- TYPES:
  1. Non-Persistent
  2. Persistent

- Non-Persistent:

  In this type of XSS vulnerability an attacker is able to execute his own code into a webpage but no changes can be done in that website.

- Persistent:

  In this case attacker stores his executable script in the vulnerable website database which is being executed every time webpage is showing the data.

- Common targets are:
  - Comments
  - Chat messages
  - E-mail messages
  - Wall posts, etc.

- Bad codes
  - Bad code 1:

```
if (isset($_GET['color']))
{
    $color = $_GET['color'];
    $color = htmlspecialchars($color);
    echo "<body bgcolor='". $color."'></body>";
}
```

- Bad code 2:

```
if (isset($_GET['color']))
{
    $color = $_GET['color'];
    $color = htmlspecialchars($color, ENT_QUOTES);
    echo "<body bgcolor=$color></body>";
}
```

- Preventing solutions:
  - Input validation
  - **Output encoding**

- Input validation
  - whitelist of acceptable inputs
  - Consider potential input properties: length, type, range of acceptable values, syntax


```
function validate($input)
{
        if(!is_string($input))
                die( “input must be string !” );
        if(strlen($input) > 10)
                die( “input length must lower than 10” );
        if(!pregmatch( “/^city/” ,$input))
                die( “input must begin with city word” );
        $whitelist={ “red” , “green” , “blue” };
        if(!in_array($input, $whitelist))
                die( “bad input” );
}
```

- Output encoding
  - Sanitizing all values before outputing to browser
  - Output encoding functions:
    - htmlentities: convert all applicable characters to HTML entities


```
function encoding($output)
{
        return htmlenties($output);
}
$safe_value = encoding($value);
echo $safe_value;
```

File Upload Attack

- Introduction
- Bad codes
- Preventing solutions

- Allow attacker to upload malicious files to server
- Most of time, it's web shell to take control over web server
- Risk:
  - Web-shell upload
  - Website deface
  - XSS
  - Phishing
  - Malware upload
  - …

- Bad codes
  - Bad code 1:

```
if (isset($_POST['submit']))
{
  if($_FILES['userfile']['type'] != "image/gif")
  {
          echo "Sorry, we only allow uploading GIF images";
          exit;
  }

  $uploaddir = 'uploads/';
  $uploadfile= $uploaddir.basename($_FILES['userfile']['name']);

  if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile))
          echo "File is valid, and was successfully uploaded.₩n";
  else
          echo "File uploading failed.₩n";
}
```

– Bad code 2:

```
if (isset($_POST['submit']))
{
  $imageinfo = getimagesize($_FILES['userfile']['tmp_name']);

  if($imageinfo['mime'] != 'image/gif' && $imageinfo['mime'] != 'image/jpeg')
  {
        echo "Sorry, we only accept GIF and JPEG images₩n";
        exit;
  }
  $uploaddir = 'uploads/';
  $uploadfile= $uploaddir.basename($_FILES['userfile']['name']);

  if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile))
        echo "File is valid, and was successfully uploaded.₩n";
  else
        echo "File uploading failed.₩n";
}
```

– Bad code 3:

```
if (isset($_POST['submit']))
{
  $uploaddir =  'D:/uploads/';
  $uploadfile= $uploaddir.basename($_FILES['userfile']['name']);

  if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile))
  {
        echo "File is valid, and was successfully uploaded.₩n";
        echo "<IMG SRC='" . $uploadfile . "'>";
  }
  else
        echo "File uploading failed.₩n";
}
```

- Preventing solutions
  - Keep uploaded files where they cannot be directly accessed by the users via a direct URL
    - Outside of webroot
    - Or configure web server to deny access to upload directory
  - Use system-generated file names instead of the names supplied by users when storing files


```
if (isset($_POST['submit']))
{
  $uploaddir =  'D:/uploads/';
  $new_file_name = rand(1,1000);
  $uploadfile = $uploaddir . $new_file_name;

  if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile))
  {
        echo "File is valid, and was successfully uploaded.₩n";
        echo "<IMG SRC='" . $uploadfile . "'>";
  }
  else
        echo "File uploading failed.₩n";
}
```

The end

Thank you !