Announcements

• Last week’s guest lecture: Slides on Ed board (please don’t reshare)
• Lab 2:
  – Sign-up form out
  – Actual lab out soon (today or tomorrow)
  – Deadline: November 29
    • Trying to avoid Thanksgiving, but don’t wait to start!
• No Lab 3 this quarter
• Homework 3 will include some “hands-on” components
• Final Project checkpoint #1: Due Friday
• Friday is a holiday! (Veterans’ Day)
Review: Dynamic Web Application

GET / HTTP/1.1

HTTP/1.1 200 OK

Browser

Web server

index.php

Database server
Review: Cross-Site Scripting (aka XSS)

```
naive.com/hello.php?name=User
```

Welcome, dear *User*

```
```

Welcome, dear

CSE 484 - Fall 2022
Preventing Cross-Site Scripting

• Any user input and client-side data must be preprocessed before it is used inside HTML
• Remove / encode HTML special characters
  – Use a good escaping library
    • OWASP ESAPI (Enterprise Security API)
    • Microsoft’s AntiXSS
  – In PHP, htmlspecialchars(string) will replace all special characters with their HTML codes
    • ‘ becomes &#039;  “ becomes &quot;  & becomes &amp;
  – In ASP.NET, Server.HtmlEncode(string)
Evading Ad Hoc XSS Filters

• Preventing injection of scripts into HTML is hard! Use standard APIs
  – Blocking “<” and “>” is not enough
  – Event handlers, stylesheets, encoded inputs (%3C), etc.
  – phpBB allowed simple HTML tags like <b>
    <b c=""> onmouseover="script" x="<b">Hello<b>

• Beware of filter evasion tricks (XSS Cheat Sheet)
  – If filter allows quoting (of <script>, etc.), beware of malformed quoting:
    <IMG """"> <SCRIPT>alert("XSS")</SCRIPT>""
  – Long UTF-8 encoding
  – Scripts are not only in <script>:
    <iframe src='https://bank.com/login' onload='steal()'>
MySpace Worm (1)

• Users can post HTML on their MySpace pages
• MySpace does not allow scripts in users’ HTML
  – No <script>, <body>, onclick, <a href=javascript://>
• ... but does allow <div> tags for CSS.
  – <div style="background:url(‘javascript:alert(1)’)">
• But MySpace will strip out “javascript”
  – Use “java<NEWLINE>script” instead
• But MySpace will strip out quotes
  – Convert from decimal instead:
    alert('double quote: ' + String.fromCharCode(34))
MySpace Worm (2)

Resulting code:

<http://www.myspace.com/>

https://samy.pl/myspace.tech.html
MySpace Worm (3)

• “There were a few other complications and things to get around. This was not by any means a straight forward process, and none of this was meant to cause any damage or [make anyone angry]. This was in the interest of..interest. It was interesting and fun!”

• Started on “samy” MySpace page

• Everybody who visits an infected page, becomes infected and adds “samy” as a friend and hero

• 5 hours later “samy” has 1,005,831 friends
  – Was adding 1,000 friends per second at its peak
Another Common Web App Vulnerability: SQL Injection
Typical Login Prompt
Typical Query Generation Code

```php
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key " .
    "WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

What if `user` is a malicious string that changes the meaning of the query?
User Input Becomes Part of Query

Web browser (Client) → Web server
Enter Username & Password

Web server → DB
SELECT passwd
FROM USERS
WHERE uname IS ‘$user’
Normal Login

Web browser (Client) -> Enter Username & Password
Web server
DB

SELECT passwd FROM USERS WHERE uname IS ‘franzi’
Malicious User Input
SQL Injection Attack

Web browser (Client) → Enter Username & Password

Web server

SELECT passwd FROM USERS
WHERE uname IS ''); DROP TABLE USERS; --

DB

Eliminates all user accounts
SQL Injection: Basic Idea

1. **Post malicious form**

2. **Unintended query**

3. **Receive data from DB**

- This is an **input validation vulnerability**
  - Unsanitized user input in SQL query to back-end database changes the meaning of query
- **Special case of command injection**

Victim server

Victim SQL DB
Authentication with Backend DB

set UserFound = execute(
    "SELECT * FROM UserTable WHERE
    username=' ' & form("user") & ' ' AND
    password=' ' & form("pwd") & ' ');

User supplies username and password, this SQL query checks if user/password combination is in the database

If not UserFound.EOF
    Authentication correct
else Fail

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database

(*) remember to hash passwords for real authentication scheme
Using SQL Injection to Log In

• User gives username: ’ OR 1=1 --

• Web server executes query

  set UserFound=execute(
  SELECT * FROM UserTable WHERE
  username=‘ ’ OR 1=1 -- … );

  Always true!  Everything after -- is ignored!

• Now all records match the query, so the result is not empty
  ⇒ correct “authentication”!

CSE 484 - Fall 2022
“Blind SQL Injection”
https://owasp.org/www-community/attacks/Blind_SQL_Injection

• SQL injection attack where attacker asks database series of true or false questions

• Used when
  – the database does not output data to the web page
  – the web shows generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

• SQL Injection vulnerability more difficult to exploit, but not impossible.
Preventing SQL Injection

• Validate all inputs
  – Filter out any character that has special meaning
    • Apostrophes, semicolons, percent, hyphens, underscores, ...
    • Use escape characters to prevent special characters from becoming part of the query code
      – E.g.: escape(O’Connor) = O\’Connor
  – Check the data type (e.g., input must be an integer)

• Same issue as with XSS: is there anything accidentally not checked / escaped?
Prepared Statements

PreparedStatement ps =
   db.prepareStatement("SELECT pizza, toppings, quantity, order_day "
   + "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParameter("month")));
ResultSet res = ps.executeQuery();

- **Bind variables**: placeholders guaranteed to be data (not code)
- Query is parsed without data parameters
- Bind variables are typed (int, string, ...)

http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html
Core Issue: Data-As-Code

• XSS

• SQL Injection

• (Like buffer overflows)