CSE 484 / CSE M 584: Computer Security and Privacy

Web Security:
Web Application Security [continued]

Spring 2015

Franziska (Franzi) Roesner
franzi@cs.washington.edu

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

• Homework #2 due on Friday (5pm)
• Lab #2 out Friday
  – Fill out the Catalyst form with group names and passwords (don’t reuse a password!!!!) by Thursday 5pm
• Guest lecture (Ben Livshits, MSR) on Friday
• Section this week
  – Intro to Lab #2
  – Clickjacking attacks
Cross-Site Scripting (XSS) [continued]
Reminder: XSS – Quick Demo

```php
setcookie("SECRET_COOKIE", "12345");
header("X-XSS-Protection: 0");
?>
<html><body><br><br>
<form action="vulnerable.php" method="get">
Name: <input type="text" name="name" size="80">
<input type="submit" value="submit"></form>
<br><br><br>
<div id="greeting">
```php
$name = $_GET['name'];
if($name) { echo "Welcome " . $_GET['name'];}
?>
</div></body></html>
```

Need to explicitly disable XSS protection – newer browsers try to help web developers avoid these vulnerabilities!
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

User victim

Attack server

Server victim
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

- **User victim**: request content
- **Server victim**: store bad stuff
- **Attack server**: inject malicious script

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

Users view or download content
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;
  – 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>

  <b c="">
  onmouseover="script" x="">
  Hello
  
• 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
<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 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
Command Injection and SQL Injection
Recall: PHP

- Server scripting language with C-like syntax
- Can intermingle static HTML and code
  ```html
  <input value=.GetTypeValue(); ?>>
  ```
- 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

\texttt{system("cp \texttt{temp.dat} \$name.dat")}

User calls

http://victim.com/copy.php?name="a; \texttt{rm *}"  

copy.php executes

\texttt{system("cp \texttt{temp.dat} a; \texttt{rm *\texttt{.dat}}");}
SQL

- Widely used database query language
- Fetch a set of records
  
  ```
  SELECT * FROM Person WHERE Username= 'Franzi'
  ```
- Add data to the table
  
  ```
  INSERT INTO Key (Username, Key) VALUES ( 'Franzi', 3611BBFF)
  ```
- Modify data
  
  ```
  UPDATE Keys SET Key=FA33452D WHERE PersonID=5
  ```
- Query syntax (mostly) independent of vendor
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?
Typical Login Prompt
User Input Becomes Part of Query

```
SELECT passwd
FROM USERS
WHERE uname IS '${user}'
```

Web browser (Client)

Web server

DB

Enter Username & Password

SELECT passwd FROM USERS WHERE uname IS '${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

```sql
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

- 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

Attacker → **post malicious form** → Victim server → **unintended query** → 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
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”!
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)

• Whitelist permitted characters
  – Blacklisting “bad” characters doesn’t work
    • Forget to filter out some characters
    • Could prevent valid input (e.g., last name O’Brien)
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

View catalog

www.e_buy.com

Select item

www.e_buy.com/shopping.cfm?pID=269

Check out

www.e_buy.com/checkout.cfm?pID=269&item1=102030405

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

- Unstable, frequently changing URLs
- Vulnerable to eavesdropping
- There is no guarantee that URL is private
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.”

<FORM METHOD=POST
ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">

  Black Leather purse with leather straps<BR>
  <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

Bargain shopping!

Fix: MAC client-side data, or, more likely, keep on server.
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
(Other) Options for Storing Session Tokens

• Embedded in URL links
  – https://site.com/checkout?SessionToken=kh7y3b

• Browser cookie
  – Set-Cookie: SessionToken=fduhye63sfdb

• Store in a hidden form field
  – <input type="hidden" name="sessionid" value="kh7y3b"/>

• Window.name DOM property
Issues

• Embedded in URL link
  — Token leaks out via HTTP Referer header

• Browser cookie
  — Browser sends it with every request, even if request not initiated by the user (cross-site request forgery)
  — Leaked in HTTP requests (use HTTPS!)

• Hidden form field
  — Short sessions only

• DOM property
  — Not private, does not work if user connects from another window, short sessions only