CSE 484 / CSE M 584: Computer Security and Privacy

XSS attacks

Fall 2016

Ada (Adam) Lerner

lerner@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
OWASP Top 10 Web Vulnerabilities

1. Injection
2. Broken Authentication & Session Management
3. Cross-Site Scripting
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery
9. Using Known Vulnerable Components
10. Unvalidated Redirects and Forwards

http://www.owasp.org
CSRF

• “Confused Deputy” – the browser acts with Alice’s privileges (cookies) even when directed to make requests by an attacker

• Defenses:
  – Form synchronization tokens
  – Referer header checking
Cross-Site Scripting (XSS)
XSS

• I have a friend with a really hard to pronounce name.

PHP: Hypertext Processor

PHP:

prints out: `<?php echo $name; ?>>`
XSS

• “Reflected” XSS – vulnerable service echoes user input directly from input (e.g., from query string in URL)
  – example.com?name=<img src=...

• “Stored” XSS – vulnerable service echoes user input stored in database
  – E.g., Make a social media post that includes a <script> tag, and when other people read your post...
Defenses: Cross-Site Scripting (XSS)

- 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)
With appropriate defenses

naive.com/hello.cgi?
name=Bob

Welcome, dear Bob

naive.com/hello.cgi?

With filters in place

• `<html> Welcome, dear Bob </html>`

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</b>
Evading XSS Filters

• 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
    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

http://namb.la/popular/tech.html
Command Injection and SQL Injection
Command Injection in PHP

http://victim.com,copy.php?name=username

copy.php includes

system("cp temp.dat $name.dat")
Command Injection in PHP


copy.php includes

system("cp temp.dat $name.dat")

What if username = “/etc/shadow”?
Command Injection in PHP


copy.php includes

```php
system("cp temp.dat $name.dat")
```

Attacker uses name “a; rm*”

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

copy.php executes

```php
system("cp temp.dat a; rm *.dat");
```
• Widely used database query language
• Fetch a set of records
  \texttt{SELECT * FROM Person WHERE Username= 'lerner'}
• Add data to the table
  \texttt{INSERT INTO Key (Username, Key) VALUES ('lerner', 3611BBFF)}
• Modify data
  \texttt{UPDATE Keys SET Key=FA33452D WHERE PersonID=5}
• Query syntax (mostly) independent of vendor
Naïve 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

![User Login - Microsoft Internet Explorer](image)

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

Enter Username & Password

Web server

SELECT passwd FROM USERS WHERE uname IS ‘$user’

Web browser (Client)

DB
Normal Login

Web browser (Client) -> Enter Username & Password

Web server

SELECT passwd FROM USERS WHERE uname IS ‘franzi’

DB
Malicious User Input

![Image of a login form with malicious input]

```plaintext
Enter User Name: '; DROP TABLE USERS; --
Enter Password: ********
```
SQL Injection Attack

Enter Username & Password

Web server

SELECT passwd
FROM USERS
WHERE uname IS ‘’ ; DROP TABLE USERS; -- ’

Web browser (Client)

DB

Eliminates all user accounts
Exploits of a Mom

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

1. **Attacker** post malicious form
2. **Victim server** unintended query
3. **Victim SQL DB** 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 form 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

```java
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 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 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<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.