



OWASP

Open Web Application  
Security Project

# Covering Your XSS: Attacks in AppLand

Ralph Collum

Email: [Ralph.Collum@owasp.org](mailto:Ralph.Collum@owasp.org)

Twitter: [Optimus\\_\\_Prime](https://twitter.com/Optimus__Prime)

# OWASP Columbia

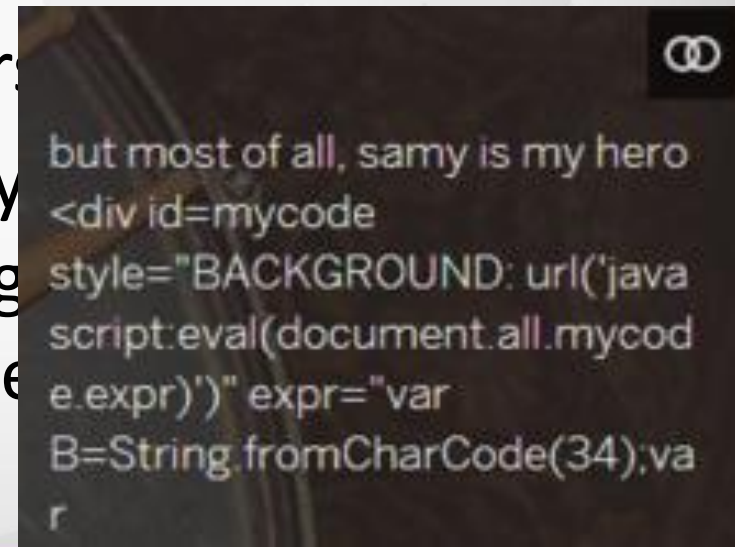
- OWASP is a worldwide free and open community focused on improving the security of application software.
- OWASP Columbia Website -> <https://www.owasp.org/index.php/Columbia>
- Sign-Up for LinkedIn Group -> <https://www.linkedin.com/groups/8342350>
- Slack and Mailing List -> Contact Frank

# Outline

- What is XSS?
- What are the Risks?
- What are the Types of XSS?
- How do I test for XSS?
- Demos/Exploitation Exercises
- What are the Countermeasure?
- Questions and Answers

# Samy Worm

- XSS Worm designed by Samy Kamkar to spread across the social media site Myspace in Oct. 2005.
- The worm would plant a string on the victims profile and send Samy a friend request.
- It impacted over one million users.
- The exploit was made possible by users who put their own JavaScript into blogs. The JavaScript executed on pages where they had been run.

A screenshot of a web page showing a JavaScript exploit payload. The text is displayed in a dark, monospaced font. The payload is a JavaScript code snippet designed to execute on a page where it has been injected. The code includes a comment, a `<div>` tag with a specific ID and style, and a JavaScript expression that evaluates to a variable declaration and assignment.

```
but most of all, samy is my hero  
<div id=mycode  
style="BACKGROUND: url('java  
script:eval(document.all.mycod  
e.expr)')" expr="var  
B=String.fromCharCode(34);va  
r
```

# What is XSS?

- Cross-Site Scripting (referred to as XSS) is a type of web application attack where malicious client-side script is injected into the application output and subsequently executed by the user's browser
- TL;DR: Not filtering out HTML and JavaScript in user input = bad
- The browser believes that the code is part of the site and runs it. It can be used to take over a user's browser in a variety of ways

# What are the Risks?

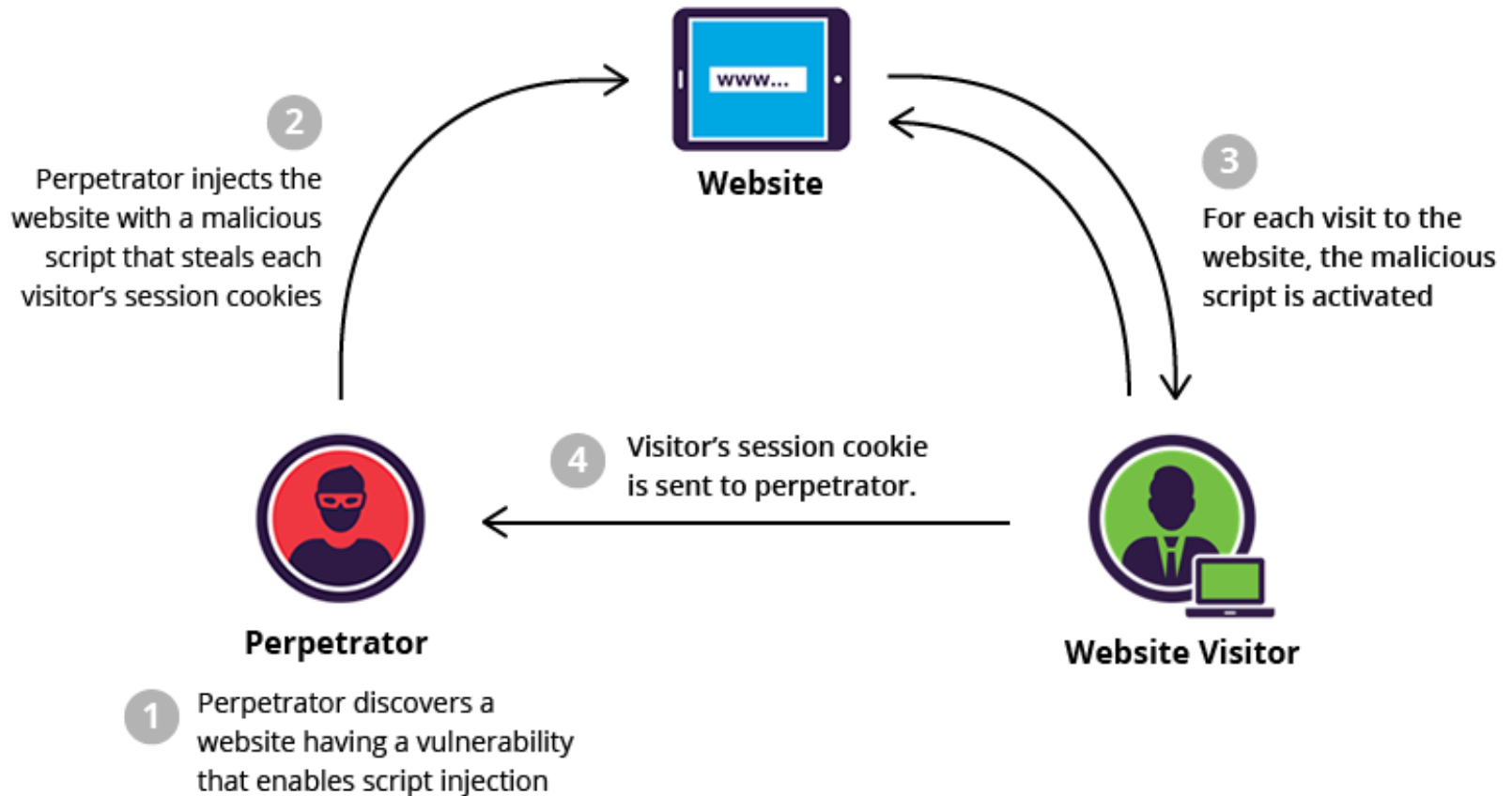
- Malicious Script Execution
- Redirecting to a Malicious Server
- Exploiting User Privileges
- Ads in Hidden IFRAMES and Pop-Ups
- Data Manipulation
- Session Hijacking
- Keylogging and Remote Monitoring
- Credential Theft
- Remote Shell



# Same Origin Policy

- A policy that allows the web browser to run scripts contained in a webpage, but only if both webpages have the same origin.
- An origin is defined as a combination of URI scheme, hostname, and port number.
- This policy is used to prevent a malicious script on one page from obtaining access to data on another webpage.

# How XSS Attacks Work





# Types of XSS Attacks

- Reflected (Non-Persistent – Type 2)
  - Malicious Content is bounced backed to the victim
    - `http://www.example.com/search/?q=<script>alert('Alert')</script>&x=0&y=0`
- Stored (Persistent – Type 1)
  - Malicious Content is stored for everyone on the server
- DOM-Based (Local Persistent – Type 0)
  - Malicious Content gets executed on the client and the server ignores the malicious request

# Discovering XSS

- Testing URLs for XSS
  - Embed XSS payload in target URL
    - `www.contoso.com/?id=[Insert XSS]`
- Testing Search Fields for XSS
  - Insert XSS payload in Search Field
- Testing Comment Fields for XSS
  - Insert XSS payload in Comment Field
- Testing User-Agent Header for XSS
  - Send XSS payload through UserAgent header

# Filtering XSS

- Whitelists
  - Filtering based on “known goods”
  - Better Security
  - Too Complex
- Blacklists
  - Filtering based on “known bads”
  - Easier to Bypass
  - Too Many Updates



# Automated Test for XSS

```
root@kali:~# python xsstrike
Made with <3 by Somdev Sangwan : TeamUltimate

-----
 \_/_/ |_____|_____|_____|_____|_____|
/_ \_ |_____|_____|_____|_____|_____|
Enter "help" to access help manual
-----

[?] Enter the target URL: http://www.liveexpert.ru/search
[-] The URL you entered doesn't seem to use GET Method
[?] Does it use POST method? [Y/n]
[?] Enter post data: author_search_txt=d3v&article_search_txt=Search
[>] Payloads loaded: 17
[>] Striking the parameter(s)
[>] Testing parameter: author_search_btn
[>] Payloads injected: 17 / 17
[-] 'author_search_btn' parameter not vulnerable.
[>] Testing parameter: article_search_txt
[>] Payloads injected: 7 / 17
[+] XSS Vulnerability Found!
[+] Parameter: article_search_txt
[+] Payload: '><svg/onload=alert()'>
```

```
root@kali:~/XsSCan# python XsSCan.py -u liveexpert.ru
XsSCan - Find XSS Made Easier
Author: Sir.4mIR (Telegram: @Sir4mIR)
Telegram Channel: @The404Hacking
Bot Support: @The404Hacking_Bot
Email: The404Hacking.Team@gmail.Com
GitHub: https://github.com/The404Hacking/XsSCan

Usage: XsSCan.py -u website.com (Not www.website.com OR http://www.website.com)
Comprehensive Scan: python XsSCan.py -u website.com -e
Verbose logging: python XsSCan.py -u website.com -v
Cookies: python XsSCan.py -u website.com -c name=val name=val

Description: XsSCan is a python tool for finding Cross Site Scripting vulnerabilities in websites. This tool is the first of its kind. Instead of just checking one page as most of the tools do, this tool traverses the website and find all the links and subdomains first. After that, it starts scanning each and every input on each and every page that it found while its traversal. It uses small yet effective payloads to search for XSS vulnerabilities. XSS in many high profile websites and educational institutes has been found by using this very tool.

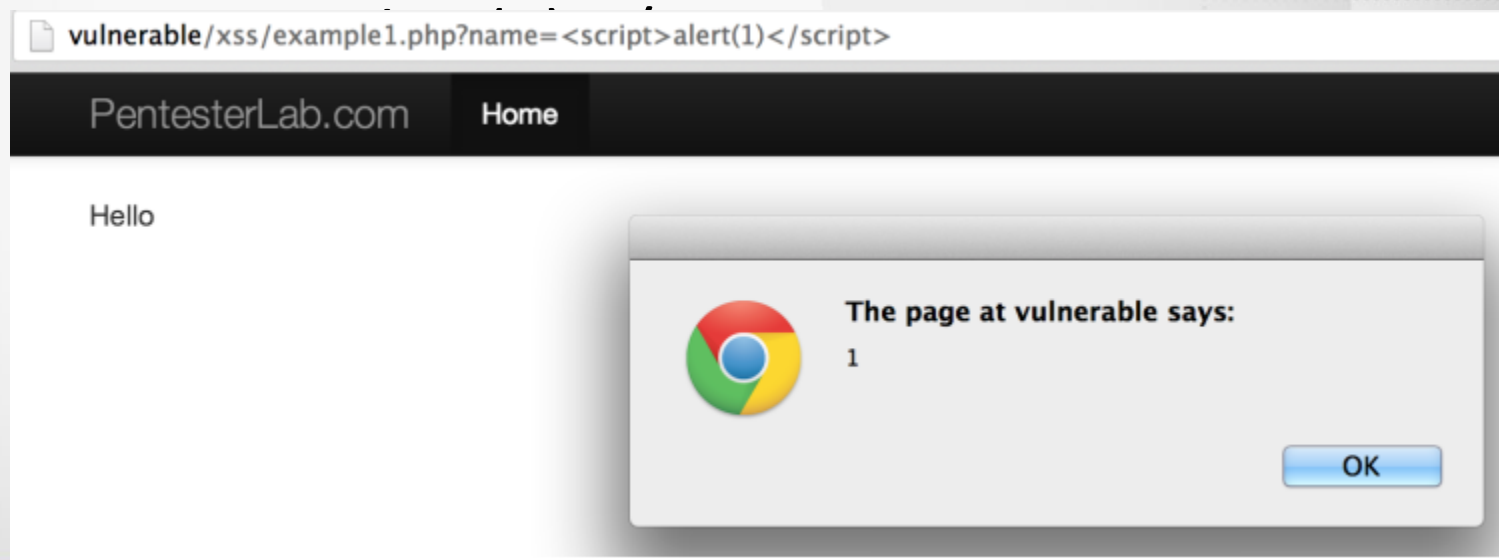
[22:52:50] Doing a short traversal.
[22:52:53] Finding all the links of the website http://www.liveexpert.ru
[22:52:53] Number of links to test are: 38
[22:52:53] Started finding XSS
[22:53:07] Link: http://www.liveexpert.ru/help/support, Payload: <svg "ons>, Element: name
[22:53:12] Link: http://www.liveexpert.ru/login/forgot, Payload: <svg "ons>, Element: email
root@kali:~/XsSCan#
```



# Exploring/Exploiting XSS

- To trigger a pop-up, you can simply use the following payload: `alert(1)`.

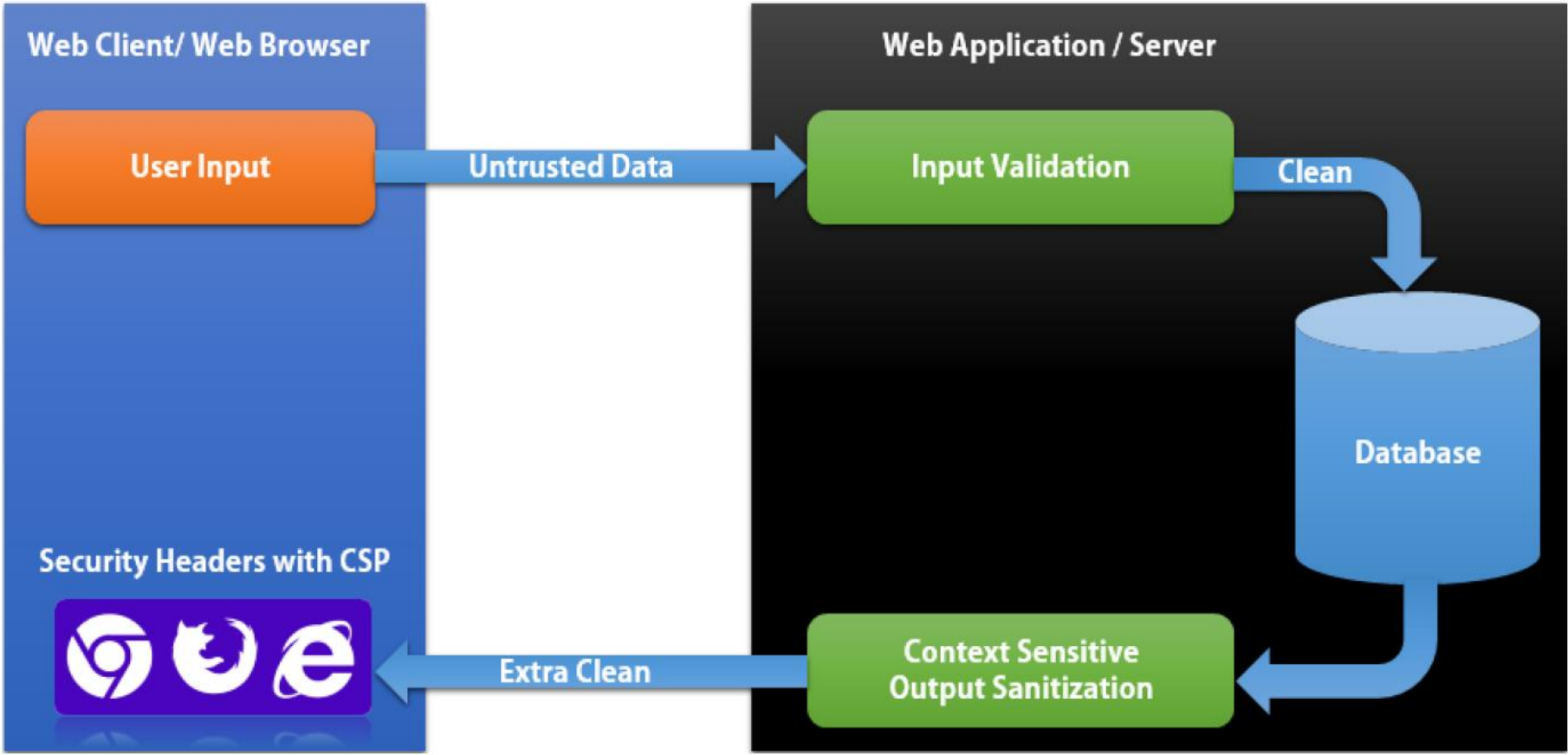
If you are injecting inside HTML code, you will need to tell the browser that this is JavaScript code. You can use the `<script>` tag to do that:



# Demo Time



# Countermeasure for XSS





# Decoding and Parsing Order

## Parsing Order in Browser



HTML Parser >> CSS Parser >> JavaScript Parser

## Decoding Order in Browser



HTML Decoding >> URL Decoding >> JavaScript Decoding





# HTTP Response Headers

HTTP Response Headers	Description
X-XSS-Protection: 1; mode=block	This header will enable the browser's built-in Anti-XSS filter.
X-Frame-Options: deny	This header will deny the page from being loaded into a frame.
X-Content-Type-Options: nosniff	This header will prevent the browser from doing MIME-type sniffing
Content-Security-Policy: default-src 'self'	This header enforces policies on loading objects and executing it from URLs or contexts.
Set-Cookie: key=value; HttpOnly	The Set-Cookie header with the HttpOnly flag will restrict JavaScript from accessing your cookies.
Content-Type: type/subtype; charset=utf-8	Always set the appropriate Content Type and Charset. (plaintext = text/html)

# Control XSS

- Almost all client-side script injection comes down to the following characters:

CONNECT. LEARN. GROW.  
< > ( ) { } [ ] " ' ; / \

- There are various ways to take care of these characters, but it is too context-dependent to give a one-size-fits-all answer
- The shortest answer is, make sure you're only getting characters you expect when a user enters any kind of information - make sure you **never** display a user-entered string without properly encoding it

# Questions & Answers

