

Cross-Site Scripting Attack (XSS)

Outline

- Evolution of Web Applications and Various Attack Vectors
- The Cross-Site Scripting attack
- Reflected XSS
- Persistent XSS
- Damage done by XSS attacks
- Basic protection mechanisms
- Countermeasures

Introduction

- The dynamic **web applications** are quite complex in nature.
- Using **web applications** becomes more and more **popular** on a daily basis,
- This motivates the **hackers** to commit cyber-crimes such as **cross-site scripting**.
- This connectivity has raised a major security threat since attacker will be able to **access personal and sensitive information**.

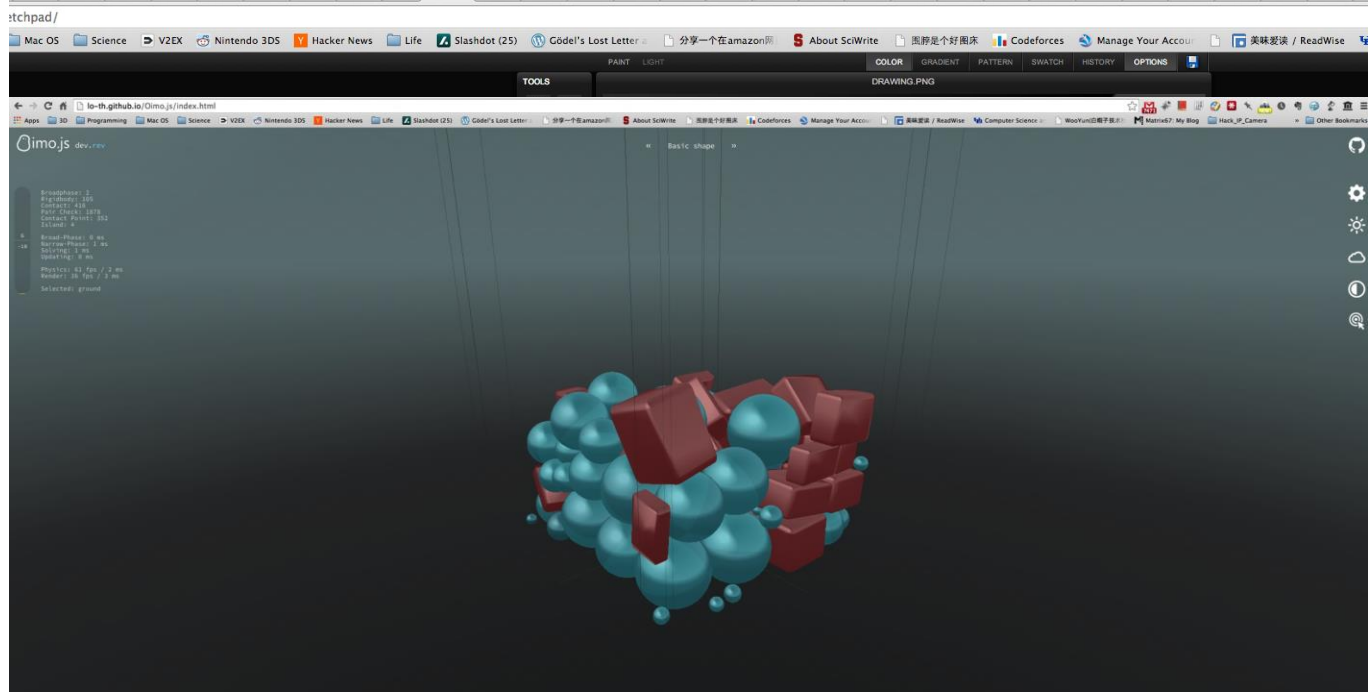
The Evolution of Web Applications

- In the early days of the Internet, the World Wide Web (WWW) consisted only of websites.
 - essentially information repositories containing static documents.



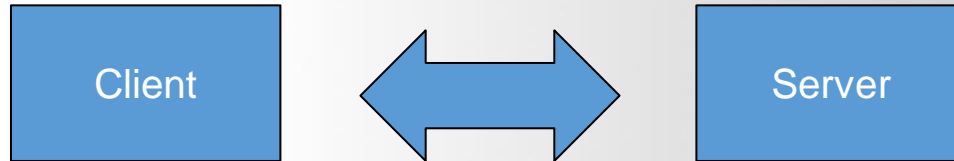
The Evolution of Web Applications

- Today, the majority of sites on the web are in fact applications.
 - Highly functional
 - Rely on two-way flow of information between the server and browser.



THE WEB IS SIMPLE

- Hyper Text Transfer Protocol (HTTP)
- Designed to allow remote document retrieval
- Simple client server model:

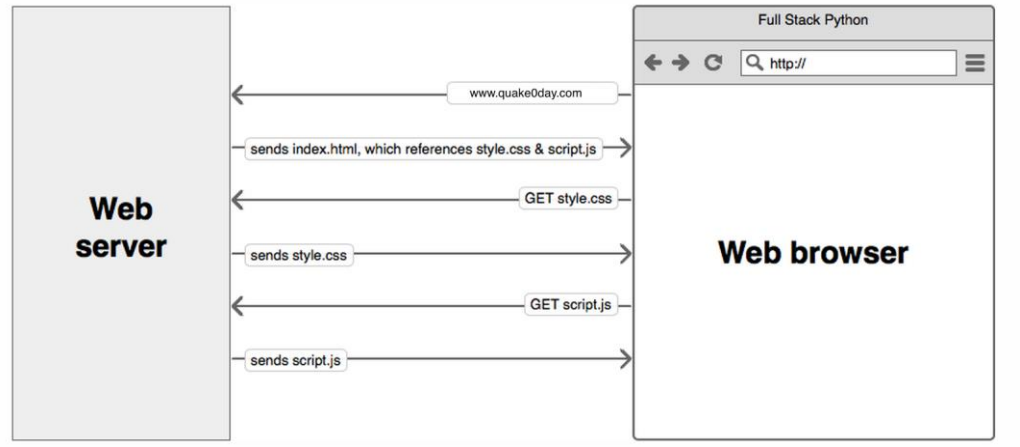


- Web application security is massively complex.
- Constant evolving field
 - WebGL, HTML5, CSS3, AJAX...



Typical Web Application Stack

- Browser (client)
- HTTP over TCP/IP
- Server
 - Operating system
 - Web Server
 - Scripting Language
 - Database or persistence layer



Just the client

- Many different clients, all implementing differently (Chrome, Firefox, Microsoft Edge, Safari, Opera, etc...)
- The breakdown of the client-server divide
 - The functional boundaries between client and server responsibilities were quickly eroded

JavaScript

1. JavaScript allows for client side programming (responsive user interface (UI))
2. Plug-in's allow for store data locally (jStorage)
3. AJAX allows display multiple HTML sources in one page

What's Worse

- In the browser world, the separation between high-level data objects (documents), user-level code (applications) is virtually nonexistent.
- Firewalls become irrelevant as everything flows over port 80 (http), 443 (https)
- Web is becoming the default content and application deliver mechanism

Now moving on to true 'attacks'....

Injection Attacks

- **SQL Injection:**
- **Command Injection:** The user being able to inject code into a command line

```
<?php
$retval = exec('echo "$line" >> logfile.txt');
?>
```

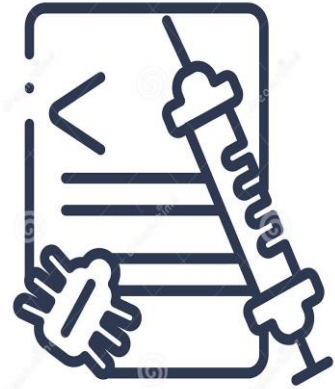
becomes

```
<?php
$retval = exec('echo ""; rm -rf *; echo "" | >> logfile.txt');
?>
```

Code Injection: User being able to directly inject code.

Code Injection Attack

- Inject code into an application.
- Injected code is then interpreted by the application, changing the way a program executes.



Code injection

ICON

The Cross-Site Scripting Attack

- Code Injection Attack executed on Client Side of a Web Application
- Inject Malicious Code through a Web Browser to do something by the web application that it is not suppose to do.

What is Cross-Site Scripting?

The **three conditions** for Cross-Site Scripting:

1. A **Web application** accepts user input
 - Well, which Web application doesn't?
2. The **Input** is used **to create dynamic content**
 - Again, which Web application doesn't?
3. The **Input** is **Insufficiently Validated**
 - Most Web applications don't validate sufficiently!

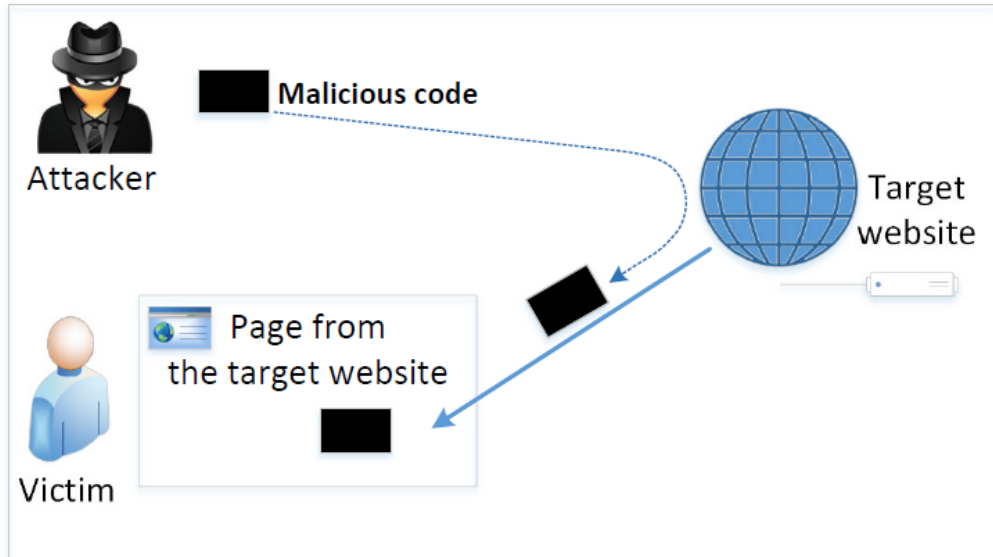
What is Cross-Site Scripting?

- Cross-Site Scripting („XSS“ or „CSS“)
- The **Players**:
 - An **Attacker**
 - Anonymous Internet User
 - Malicious Internal User
 - A company's **Web server** (i.e. Web application)
 - External (e.g.: Shop, Information, CRM, Supplier)
 - Internal (e.g.: Employees Self Service Portal)
 - A **Client**
 - Any type of customer
 - Anonymous user accessing the Web-Server

What is Cross-Site Scripting?

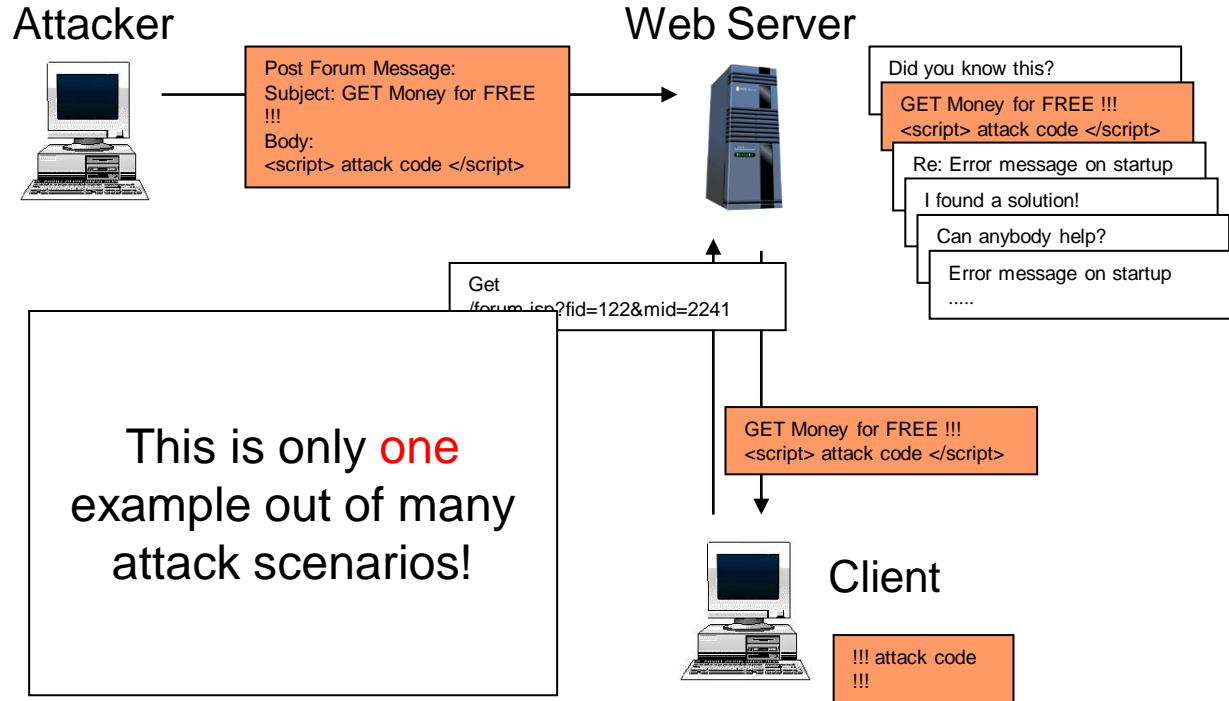
- **Scripting:** Web Browsers can execute commands
 - Embedded in HTML page
 - Supports different languages (JavaScript, VBScript, ActiveX, etc.)
 - Most prominent: JavaScript
- “**Cross-Site**” means: **Foreign script** sent via server to client
 - Attacker „makes“ Web-Server deliver malicious script code
 - Malicious script is executed in Client’s Web Browser
- **Attack:**
 - Steal Access Credentials, Denial-of-Service, Modify Web pages
 - Execute any command at the client machine

The Cross-Site Scripting Attack

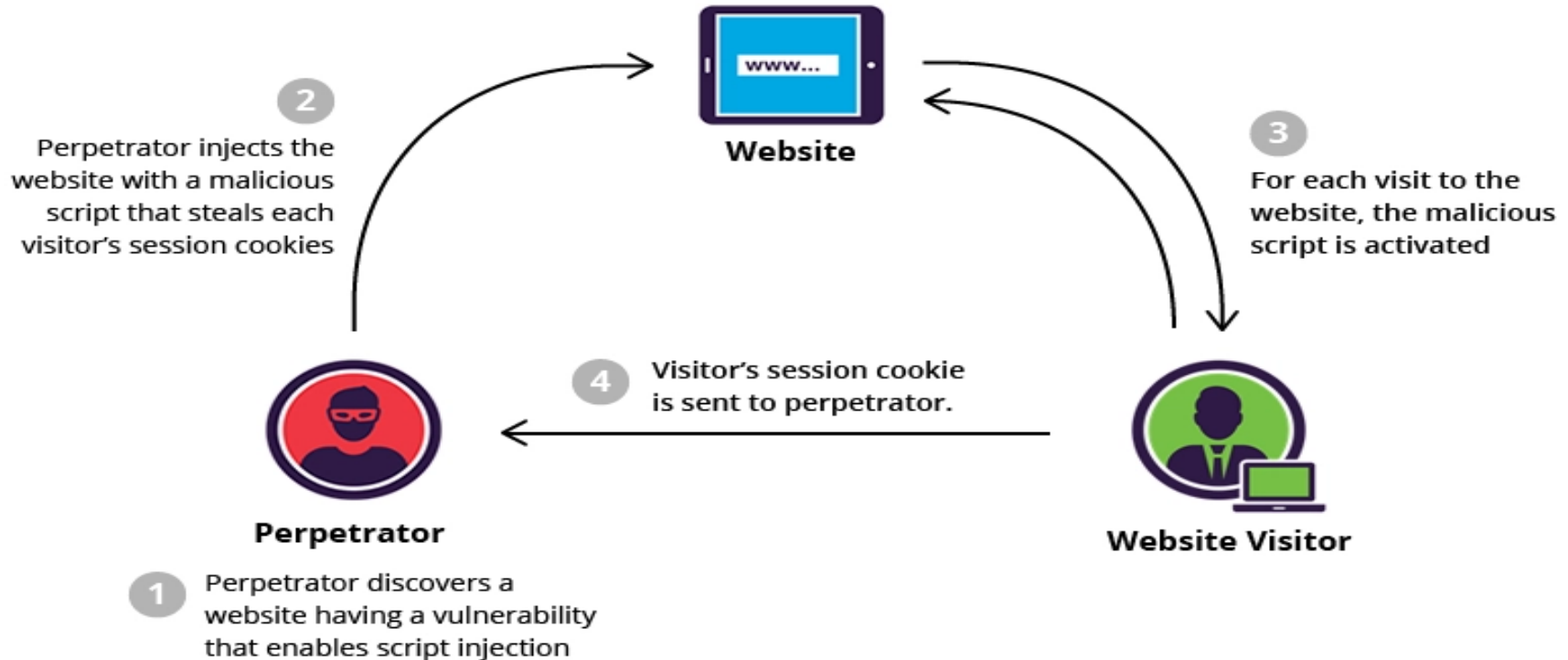


- In XSS, an attacker injects his/her malicious code to the victim's browser via the target website.
- When code comes from a website, it is considered as trusted with respect to the website, so it can access and change the content on the pages, read cookies belonging to the website and sending out requests on behalf of the user.
- Basically, code can do whatever the user can do inside the session.

XSS-Attack: General Overview



XSS-Attack: General Overview



XSS-Attack: A New Threat?



CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests

Original release date: February 2, 2000

Last revised: February 3, 2000

A web site may inadvertently include malicious HTML tags or script in a dynamically generated page based on unvalidated input from untrustworthy sources. This can be a problem when a web server does not adequately ensure that generated pages are properly encoded to prevent unintended execution of scripts, and when input is not validated to prevent malicious HTML from being presented to the user.

- XSS is an old problem
- Nevertheless:
 - Many Web applications are affected

What's the source of the problem?

- Insufficient input/output checking!
- Problem as old as programming languages

Who is affected by XSS?

- XSS attack's first target is the Client
 - Client trusts server (Does not expect attack)
 - Browser executes malicious script
- But second target = Company running the Server
 - Loss of public image (Blame)
 - Loss of customer trust
 - Loss of money

Impact of XSS-Attacks

Access to authentication credentials for Web application

- **Cookies, Username and Password**

- XSS is not a harmless flaw !

- Normal users

- Access to personal data (Credit card, Bank Account)

- Access to business data (Bid details, construction details)

- Misuse account (order expensive goods)

- High privileged users

- Control over Web application

- Control/Access: Web server machine

- Control/Access: Backend / Database systems

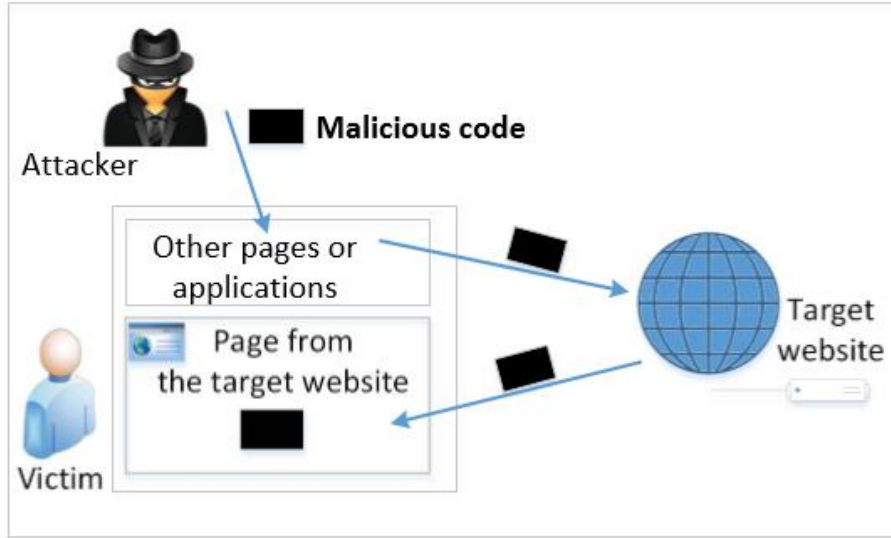
Impact of XSS-Attacks

- Denial-of-Service
 - Crash Users`Browser, Pop-Up-Flodding, Redirection
- Access to Users` machine
 - Use ActiveX objects to control machine
 - Upload local data to attacker`s machine
- Spoil public image of company
 - Load main frame content from „other“ locations
 - Redirect to dialer download

Types of XSS Attacks

- **Non-persistent (Reflected) XSS Attack**
- **(DOM Based XSS Attack)**
- **Persistent (Stored) XSS Attack**

Non-persistent (Reflected) XSS Attack

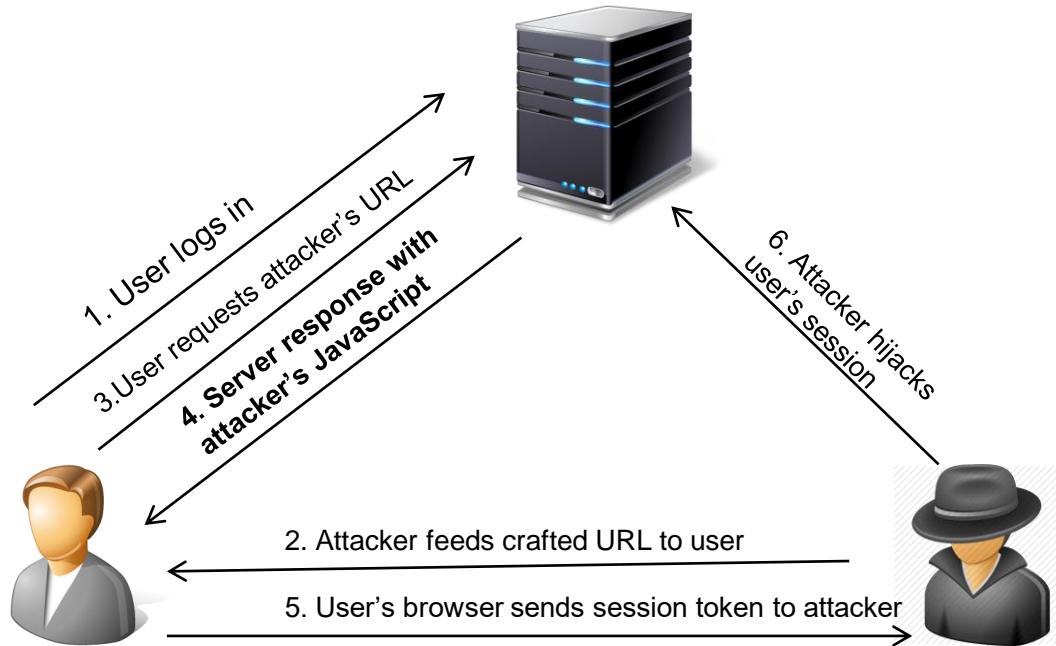


If a website with a reflective behaviour takes user inputs, then :

- Attackers can put JavaScript code in the input, so when the input is reflected back, the JavaScript code will be injected into the web page from the website.

Non-persistent (Reflected) XSS Attack

Directly **echoing back content** from the user.

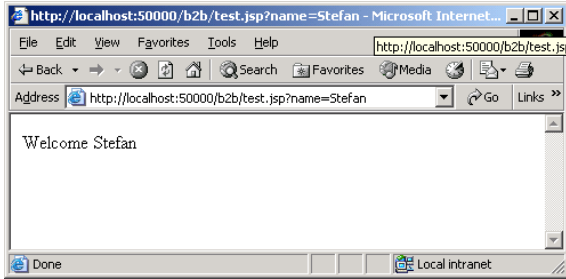


Non-persistent (Reflected) XSS Attack

- Assume a vulnerable service on website :
<http://www.example.com/search?input=word>, where word is provided by the users.
- Now the attacker sends the following URL to the victim and tricks him to click the link:
[http://www.example.com/search?input=<script>alert\("attack"\);</script>](http://www.example.com/search?input=<script>alert("attack");</script>)
- Once the victim clicks on this link, an HTTP GET request will be sent to the www.example.com web server, which returns a page containing the search result, with the original input in the page. The input here is a JavaScript code which runs and gives a pop-up message on the victim's browser.

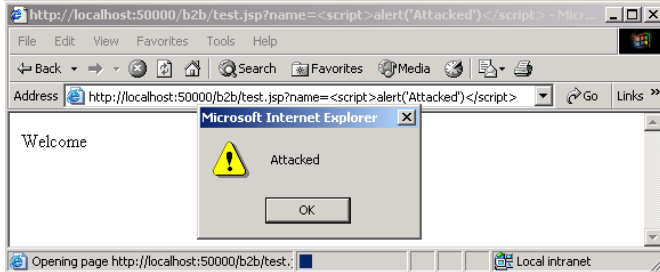
Non-persistent (Reflected) XSS Attack

http://myserver.com/test.jsp?name=Stefan



```
<HTML>
<Body>
Welcome Stefan
</Body>
</HTML>
```

http://myserver.com/welcome.jsp?name=<script>alert("Attacked")</script>



```
<HTML>
<Body>
Welcome
<script>alert("Attacked")</script>
</Body>
</HTML>
```

Non-persistent (Reflected) XSS Attack

The Security Hole:

```
<p>Thank you for your submission: <?= $_POST['first_name'] ?></p>
```

The Attack:

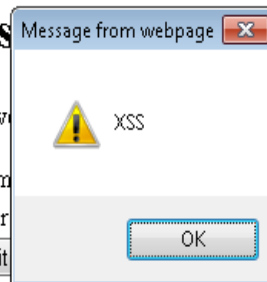
First Name:

Submit

This type of simple XSS bug accounts for approximately 75% of the XSS vulnerabilities that exists in real-world web apps.

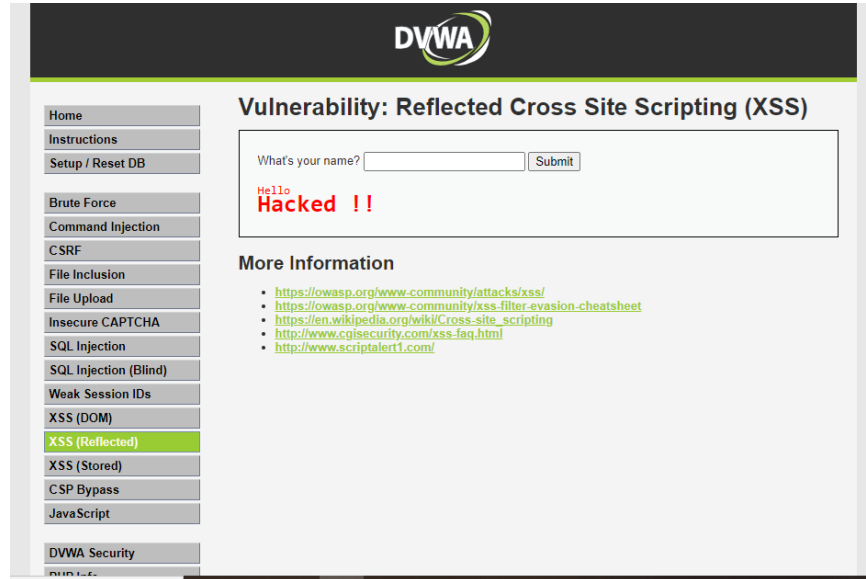
It is called reflected XSS because exploiting the vulnerability involves crafting a request containing embedded JavaScript that is reflected to **any user who makes the requests**

Sess

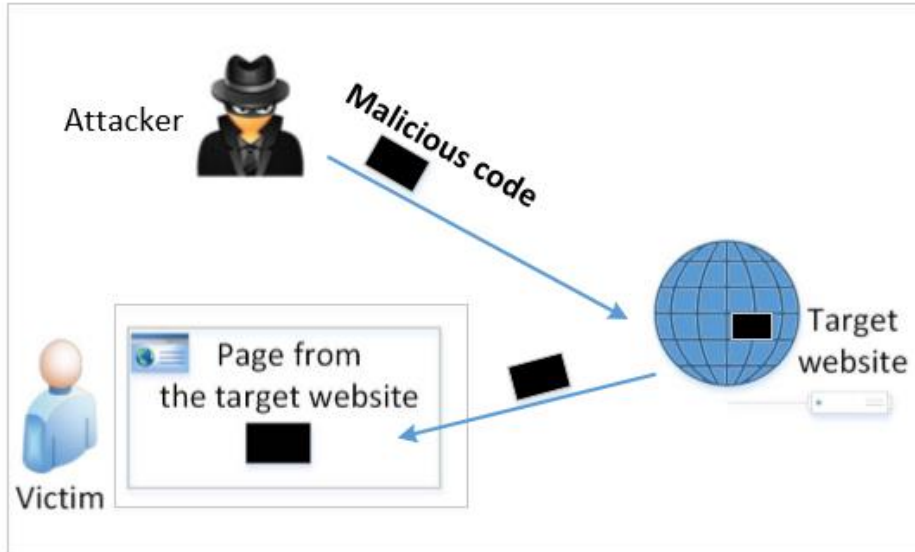


Non-persistent (Reflected) XSS Attack

- Practical Demonstration.
- http://localhost/dvwa/vulnerabilities/xss_r/



Persistent (Stored) XSS Attack



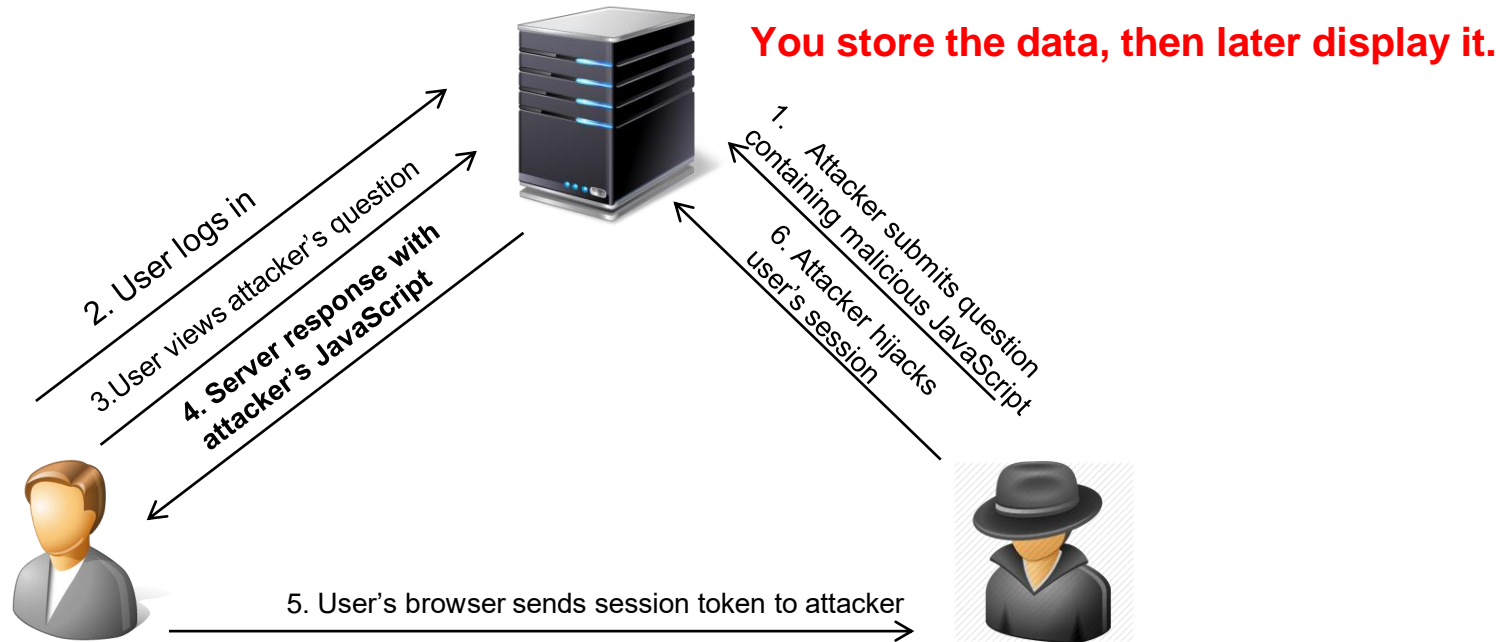
- Attackers directly send their data to a target website/server which stores the data in a persistent storage.
- If the website later sends the stored data to other users, it creates a channel between the users and the attackers.

Example : User profile in a social network is a channel as it is set by one user and viewed by another.

Persistent (Stored) XSS Attack

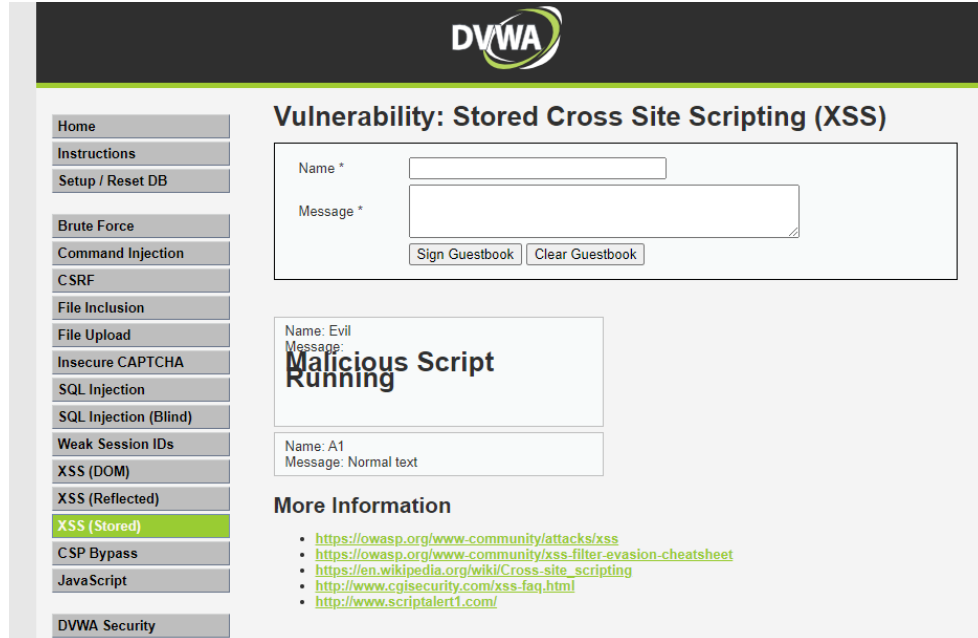
- These channels are supposed to be data channels.
- But data provided by users can contain HTML markups and JavaScript code.
- If the input is not sanitized properly by the website, it is sent to other users' browsers through the channel and gets executed by the browsers.
- **Browsers consider it like any other code** coming from the website. Therefore, the code is given the same privileges as that from the website.

Persistent (Stored) XSS Attack



Persistent (Stored) XSS Attack

- Practical Demonstration.
- http://localhost/dvwa/vulnerabilities/xss_s/



The screenshot shows the DVWA interface. The top header has the DVWA logo. The left sidebar contains a list of vulnerability categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), **XSS (Stored)**, CSP Bypass, JavaScript, and DVWA Security. The main content area is titled 'Vulnerability: Stored Cross Site Scripting (XSS)'. It features a form with 'Name *' and 'Message *' input fields, and 'Sign Guestbook' and 'Clear Guestbook' buttons. Below the form, there is a list of previous submissions. The first submission shows 'Name: Evil' and 'Message: Malicious Script Running'. The second submission shows 'Name: A1' and 'Message: Normal text'. At the bottom, there is a 'More Information' section with a list of links: <https://owasp.org/www-community/attacks/xss>, <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>, https://en.wikipedia.org/wiki/Cross-site_scripting, <http://www.cgisecurity.com/xss-faq.html>, and <http://www.scriptalert1.com/>.

Damage Caused by XSS

Web defacing:

- JavaScript code can use **DOM APIs** to access the DOM nodes inside the hosting page.
- Therefore, the injected JavaScript code can make arbitrary changes to the page.
- Example: JavaScript code can **change a news article page** to something fake or change some pictures on the page.

Damage Caused by XSS

Spoofting requests:

- The injected JavaScript code can send HTTP requests to the server on behalf of the user. (Discussed in later slides)

Stealing information:

- The injected JavaScript code can also steal victim's private data including the session cookies, personal data displayed on the web page, data stored locally by the web application.

Preventing XSS means Preventing...

- Subversion of separation of clients
 - Attacker can access affected clients' data
 - Industrial espionage
- Identity theft
 - Attacker can impersonate affected client
- Illegal access
 - Attacker can act as administrator
 - Attacker can modify security settings

Countermeasures: the Filter Approach

- Removes code from user inputs.
- It is difficult to implement as there are many ways to embed code other than `<script>` tag.
- Use of open-source libraries that can filter out JavaScript code.
- Example : jsoup

Countermeasures: the Filter Approach

Input Validation

But what is to consider “Input”?

Typical HTTP Request

POST /thepage.jsp?var1=page1.html HTTP/1.1

Accept: */*

Referer: http://www.myweb.com/index.html

Accept-Language: en-us,de;q=0.5

Accept-Encoding: gzip, deflate

Content-Type: application/x-www-url-encoded

Content-Lenght: 59

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Host: www.myweb.com

Connection: Keep-Alive

uid=fred&password=secret&pagestyle=default.css&action=login

This all is input:

Requsted Resource

GET and POST Parameters

Referer and User Agent

HTTP Method

What to Consider Input?

- Not only field values with user supplied input
- Should be treated as Input:
 - All field values: Even hidden fields
 - All HTTP header fields: Referer
 - And even the HTTP method descriptor

What if you request the following from your Web Server?

```
<script>alert("Hello")</script> / HTTP/1.0
```

- Input is any piece of data sent from the client!
 - That is the whole client request

How to perform Input Validation

- Check if the input is what you expect
 - Do not try to check for "bad input"
- Black list testing is no solution
 - Black lists are never complete!
- White list testing is better
 - Only what you expect will pass
 - (correct) Regular expressions

Countermeasures: The Encoding Approach

- Replaces HTML markups with alternate representations.
- If data containing JavaScript code is encoded before being sent to the browsers, the embedded JavaScript code will be displayed by browsers, not executed by them.
- Converts `<script> alert('XSS') </script>` to `<script>alert('XSS')`

HTML Encoding may help ...

- HTML encoding of all input when put into output pages
- There are fields where this is not possible
 - When constructing URLs from input (e.g. redirections)
 - Meta refresh, HREF, SRC,
- There are fields where this is not sufficient
 - When generating Javascript from input
 - Or when used in script enabled HTML Tag attributes

```
Htmlencode("javascript:alert(`Hello`)") = javascript:alert(`Hello`)
```

Cookie Options mitigate the impact

Complicate attacks on Cookies

- "httpOnly" Cookies
 - Prevent disclosure of cookie via DOM access
 - IE only currently
 - use with care, compatibility problems may occur
 - But: cookies are sent in each HTTP requests
 - E.G. Trace-Method can be used to disclose cookie
 - Passwords still may be stolen via XSS
- "secure" Cookies
 - Cookies are only sent over SSL

Countermeasures: Other Approachs

PHP module HTMLawed:

Highly customizable PHP script to sanitize HTML against XSS attacks.

PHP function htmlspecialchars:

Encode data provided by users, s.t., JavaScript code in user's inputs will be interpreted by browsers only as strings and not as code.

Defeating XSS using Content Security Policy

- Fundamental Problem: mixing data and code (code is inlined)
- Solution: Force data and code to be separated: (1) Don't allow the inline approach. (2) Only allow the link approach.

```
<script>  
  ... JavaScript code ...  
</script>
```

①

```
<button onclick="this.innerHTML=Date()">The time is?</button>
```

②

```
<script src="myscript.js"> </script>
```

③

```
<script src="http://example.com/myscript.js"></script>
```

④

Summary

- Cross-Site Scripting is extremely dangerous
 - Identity theft, Impersonation
- Cause: Missing or in-sufficient input validation
- XSS-Prevention Best Practices
 - Implement XSS-Prevention in application
 - Do not assume input values are benign
 - Do not trust client side validation
 - Check and validate all input before processing
 - Do not echo any input value without validation
 - Use one conceptual solution in all applications