Tamil CTF

Apr 23, 2022

Demystifying javascript attack vectors

Today's Agenda

1. Introduction to Javascript
2. Javascript Attack Vectors
3. Practical Lab
4. Remediation

Whoami

I am Godson

- Currently Doing My Graduation In BCA.
- CTF player @TamilCTF
- Love Playing with Bugs and Researching

Let's begin!

Ready to start?

What is Javascript

JavaScript is a programming language used both on the client-side and server-side that allows you to make web pages interactive.

1

JavaScript is mainly used for web-based applications and web browsers. But JavaScript is also used beyond the Web in software, servers, and embedded hardware controls.

2

Here are some basic things JavaScript is used for:

- Front-end Development
- Back-end Development
- Mobile apps
- Game Development

Attack Vectors

1. XSS
2. CSRF
3. Dom Clobbering
4. postMessage()
5. Prototype Pollution
6. Deserialization

XSS

https://victim.com?s=<script>alert('XSS')</script>

XSS

Cross-site scripting is a type of security vulnerability that can be found in some web applications. XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users

XSS Types

Reflected

Reflected cross-site scripting (or XSS) arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

Stored

Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

Dom

DOM-based XSS vulnerabilities usually arise when JavaScript takes data from an attacker-controllable source, such as the URL, and passes it to a sink that supports dynamic code execution.

Dom Based XSS

- Dom – Document Object Model

These are High Impact, High Effort tasks.

Why is Dom Used for?

The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

Source

- "Source" is a Javascript property that accepts data
- Ex Sources:
- document.URL
- document.baseURI
- location

Sink

- "Sink" is an unsafe function or DOM object into which the source value is passed.
- Ex Sinks:
- document.write()
- document.location
- eval()

CSP

A great way to mitigate XSS

CSP

CSP – Content Security Policy

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks.

As Header

- Need to configure your webserver to return the 'Content-Security-Policy' HTTP header
- Ex:

HTTP/1.1 200 OK

...

Content-Security-Policy: policy

Meta Tag

- Else, Need to Configure your Server to Return a Meta Element in HTML page
- Ex:

<meta http- equiv="Content-Security-Policy" content="policy">

CSP Policy

Understanding CSP Directives

- default-src: This directive defines the policy for fetching resources by default
- script-src: This directive specifies allowed sources for JavaScript.
- img-src: It defines allowed sources to load images on the web page.
- object-src: It defines allowed sources for the <object>, <embed>, and <applet> elements elements.
- style-src: It defines allowed sources to load Style Sheets on the web page.
- frame-src: This directive restricts URLs to which frames can be called out.
- base-uri: It defines allowed URLs that can be loaded using elements.

Unsafe-inline

Enabled due to Some False Positive

Unsafe-inline

The **unsafe-inline Content Security Policy** (CSP) keyword allows the execution of inline scripts or styles.

Example:

```
Content-Security-Policy:  script-src: 'self' 'unsafe-inline'
```

LAB

Trusted Domains

Load only from Trusted Domains

Trusted Domain

Sometimes, Developer Wants to Load scripts from another domain, so they can write a CSP rules like this

Example:

Content-Security-Policy: script-src: 'self' '**trusted.com**'

Sometimes, this can be Exploited By the JSONP endpoint avaliable in 'trusted.com'

LAB

TamilCTF

Apr 23, 2022

Nonce

The Non-Breakable Security mechanism

Nonce

A nonce is a randomly generated token that should be used only one time.

Example:

Content-Security-Policy: script-src: 'nonce-<**random**>'

The **Random Nonce** is Calculated in the Backend and sent to Browser. Every time the user refresh, the Nonce also be changed.

Nonce Reuse? . . . .

A simple
Mistake may
Exploited

Nonce-Reuse

"The **Random Nonce** is Calculated in the Backend and sent to Browser. Every time the user refresh, the Nonce also be changed."

What If Nonce is Un-Changed?

LAB

Base-uri

Missing of Base-uri

Base-URI

The **base-uri** directive restricts the URLs which can be used in a document's <base> element. If this value is **absent,** then **any URI is allowed.**

Example:

Content-Security-Policy: base-uri <source>;

Missing Base-URI can be Exploited via **HTML Injection**

LAB

Solution

Remediation For XSS.

1. Write strict csp rules
2. Make Sure, User Input in sanitized Before passing into sinks
3. Use Dompurify

CSRF

Make sure, its not a forgery document

CSRF

A CSRF token is a secure random token that is used to prevent CSRF attacks.

Storing CSRF tokens in HTML form can be Exploited.

Stealing CSRF token

```
<form method="POST">
  <input type="hidden" name="_token" value=
  "6JhQN8yVuLg2dCafKw7QvCeonvYFUFuVjNQb0OL6">
  <input type="text" name="username" placeholder="Username">
  <input type="password" name="password" placeholder="Password">
  <input type="submit" name="submit" value="Submit">
</form>
```

A form with a Randomly Generated CSRF token. But the problem is CSRF token is Present inside the Form tag. This Can be Steal with XHR

LAB

Solution

Remediation For CSRF.

1. Get CSRF from Cookies
2. Make Sure to Set HTTP only Flag
3. In Addition to this, Implement a Perfect CORS

TamilCTF

Apr 23, 2022

Dom Clobbering . . . .

A P2 HTML injection

Dom Clobbering

DOM clobbering is a technique in which you inject HTML into a page to manipulate the DOM and ultimately change the behavior of JavaScript on the page.

Dom Clobbering can be Exploited, when there is a Use of **Undefined Variables** in Javascript Sinks

Exploit Dom Clobbering

```
<script>
    window.onload = function(){
        let someObject = window.someObject || {};
        let script = document.createElement('script');
        script.src = someObject.url;
        document.body.appendChild(script);
    };
</script>
```

In Terms of Using an Undefined Variable in Javascript, A Harmless HTML Injection can be Used to Exploit Dom Clobbering

LAB

Solution

Remediation For Dom Clobbering.

1. Don't Use Undefined Variables
2. Avoid these Bad Code pattern
3. Use DOMPurify

TamilCTF

Apr 23, 2022

Prototype Pollution

A Deadly and Underrated Bug

Prototype Pollution

Prototype pollution is a vulnerability where an attacker is able to modify **prototype** of Object

There are 2 types of Prototype Pollution
Client-side and Server Side Prototype Pollution

Client Side

- In Client-Side Prototype Pollution, We can Exploit Client-Side Bugs
- Ex: XSS

Server Side

- In Server Side Prototype Pollution, We can Exploit Server Side Bugs
- Ex: RCE, LFI, RFI

Overview Exploitation

```
> var obj = {'name':'TamilCTF'}
< undefined
> obj.__proto__.isAdmin = true
< true
> var a = {}
< undefined
> a.isAdmin
< true
>
```

The Basic Idea of Prototype Pollution is, Polluting the Present Methods in the Object Prototype or Injecting New Prototype into the Object Prototype to Change the Logic of the Application

LAB

Solution

Remediation For Prototype Pollution.

1. Always use latest library
2. Sanitize User Input Before Passing into Merging Operations

TamilCTF

Arp 23, 2022

Node JS Deserialization

Deserialization, Where the All Languages are Vulnerable...

Serialization

- Serialization is the process of turning some object into a data format that can be restored later. People often serialize objects in order to save them to storage, or to send as part of communications.

Deserialization

- Deserialization is the process of reconstructing a data structure or object from a series of bytes or a string in order to instantiate the object for consumption.

LAB

Solution

Remediation For Node JS Deserialization

1. Don't Use
2. If necessary, Make Sure to Sanitize the Serialized User Input

postMessage()

Origin Check
is Must

PostMessage()

What is postMessage?

- The window.postMessage() method safely enables cross-origin communication between Window objects. e.g., between a page and a pop-up that it spawned, or between a page and an iframe embedded within it.

SOP

What is Same Origin Policy?

The same-origin policy is a critical security mechanism that restricts how a document or script loaded by one origin can interact with a resource from another origin.

Principles of SOP

Principles of SOP

- Protocol Must be Same
- Origin Must be Same
- Port Number Must be Same

Example:

Website = `http://sop.tamilctf.com:80/users.php`

```
protocol: http
origin: https://sop.tamilctf.com
port: 80
```

URL	SOP Violation
`http://lol.tamilctf.com:80/users.json`	YES (origin Mismatch)
`https://sop.tamilctf.com/users.json`	YES (protocol Mismatch)
`http://sop.tamilctf.com:443/users.json`	YES (Port Mismatch)
`http://sop.tamilctf.com:80/bla/users.json`	No (Everything is OK)

LAB

Solution

Remediation For postMessage() Bugs

1. Always Check Origin Before passing input into DOM sinks.

2. Also, Make sure to Sanitize the Source before passed into sinks

Thank you!

Have a great day ahead.

Question?

. . . .