Apr 23, 2022

Demystifying javascript attack °°°° vectors

Tamil CTF

Today's Agenda

0 0 0 0

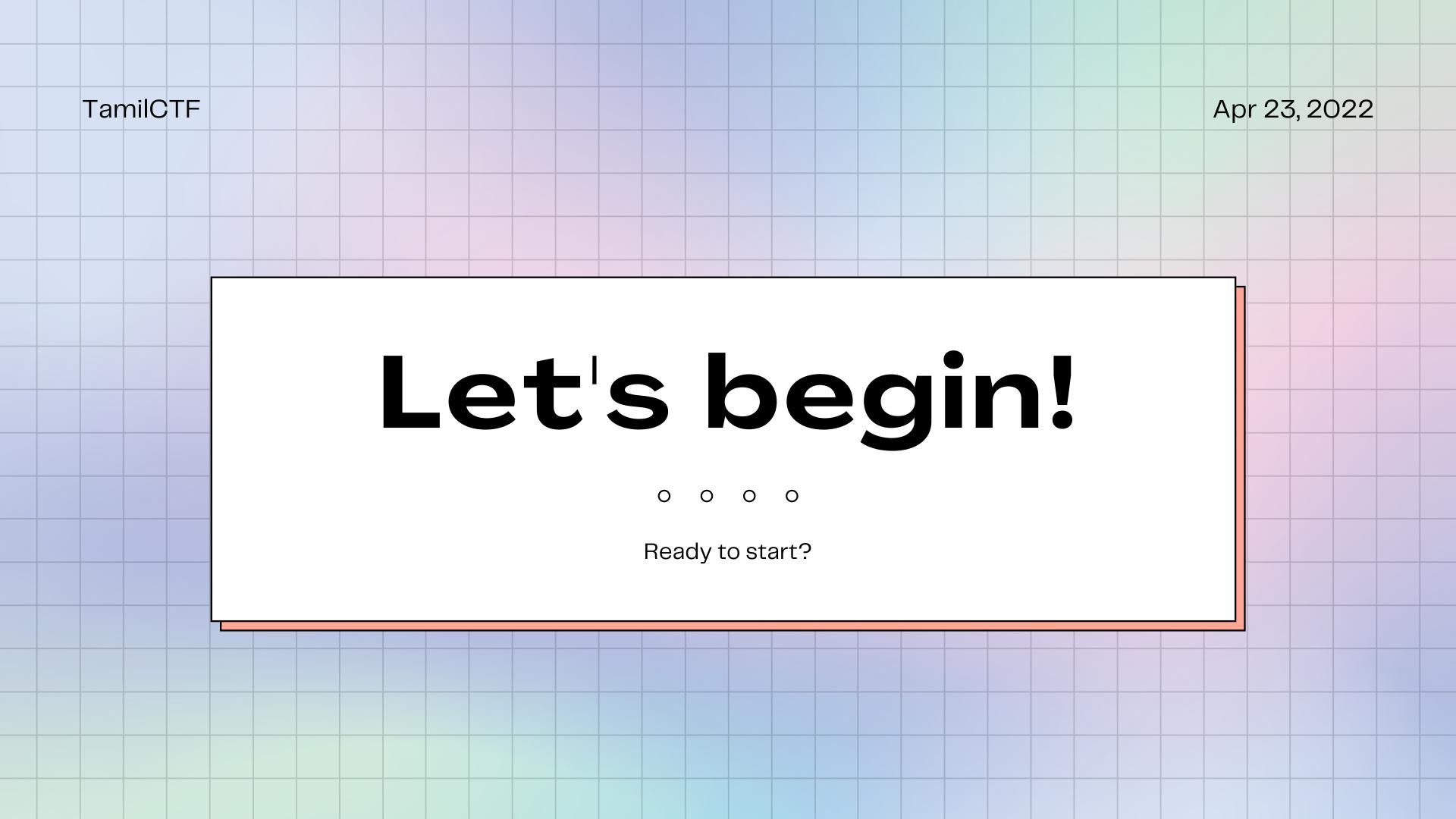
- Introduction to Javascript
- 2 Javascript Attack Vectors
- 3 Practical Lab
- 4. Remediation

Whoami

0

I am Godson

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



What is Javascript

0 0 0 0

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

Here are some basic things JavaScript is used for:

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

TamilCTF

Attack Vectors

0 0 0 0

1 xss

postMessage()

2 CSRF

5 Prototype Pollution

- 3 Dom Clobbering
- 6 Deserialization

XSS

0 0 0 0

o o o 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

0 0 0 0

O O 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

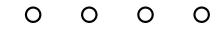
Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

\bigcirc \bigcirc \bigcirc

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

0 0 0 0

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

Sink

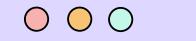
0 0 0 0

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



CSP

0 0 0 0



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

- 0 0 0 0
- Need to configure your webserver to return the 'Content-Security-Policy' HTTP header
- Ex:

HTTP/1.1 200 OK

--

Content-Security-Policy: policy

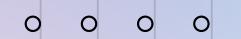
Meta Tag

0 0 0 0

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

<meta http- equiv="ContentSecurity-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



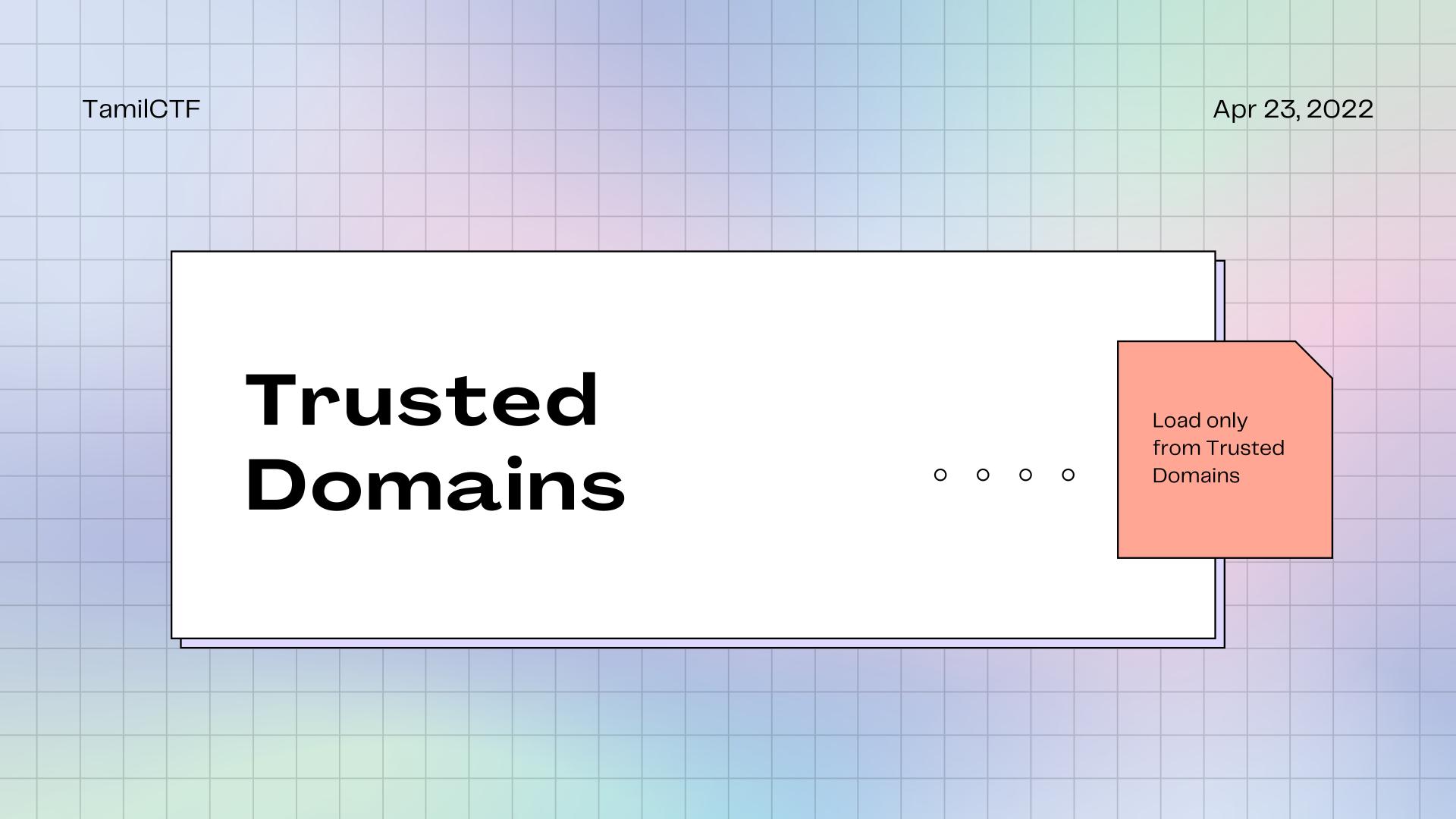
The **unsafe-inline Content Security Policy** (CSP) keyword allows the execution of inline scripts or styles.

0

Example:

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





Trusted Domain

0 0 0 0



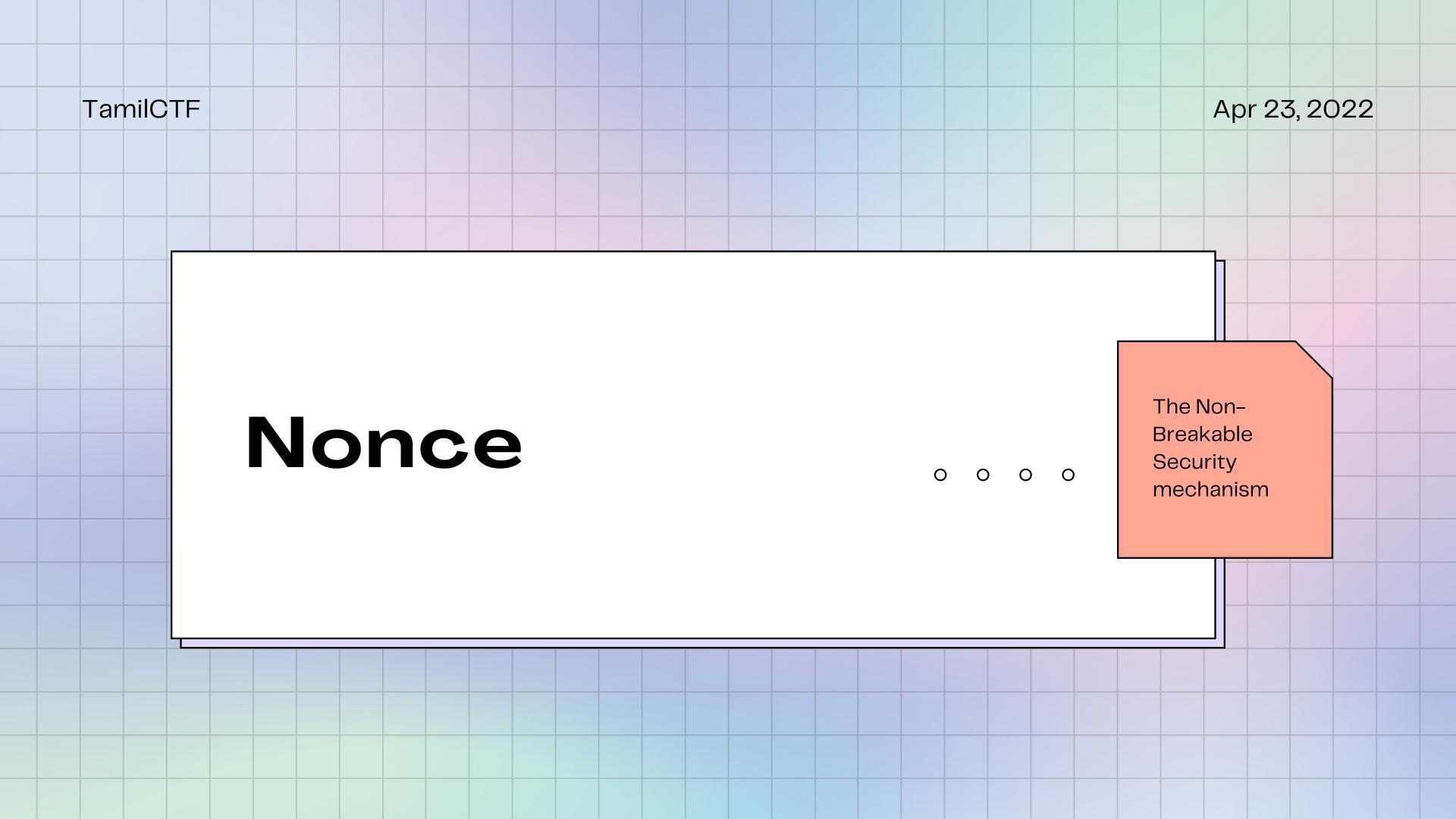
Sometimes, Developer Wants to Load scripts from another domain, so they can write a CSP rules like this

Example:

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

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





Nonce

0 0 0 0



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.



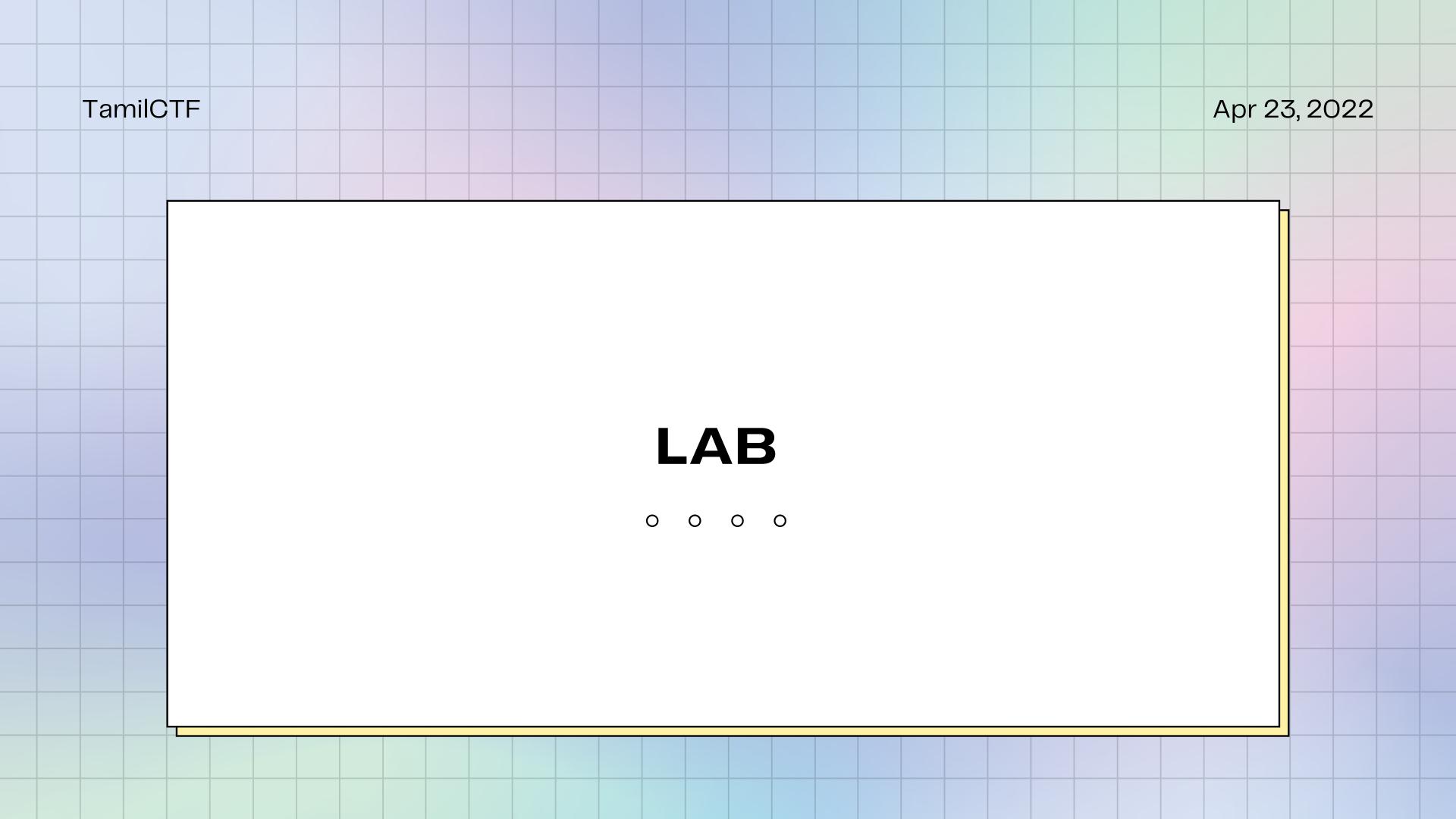
Nonce-Reuse

0 0 0 0



"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?





Base-URI

0 0 0 0



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



Solution

0 0 0 0

Remediation For XSS.

Write strict csp rules

Make Sure, User Input in sanitized Before passing into sinks

3 Use Dompurify



CSRF

0 0 0 0

 \bigcirc \bigcirc \bigcirc

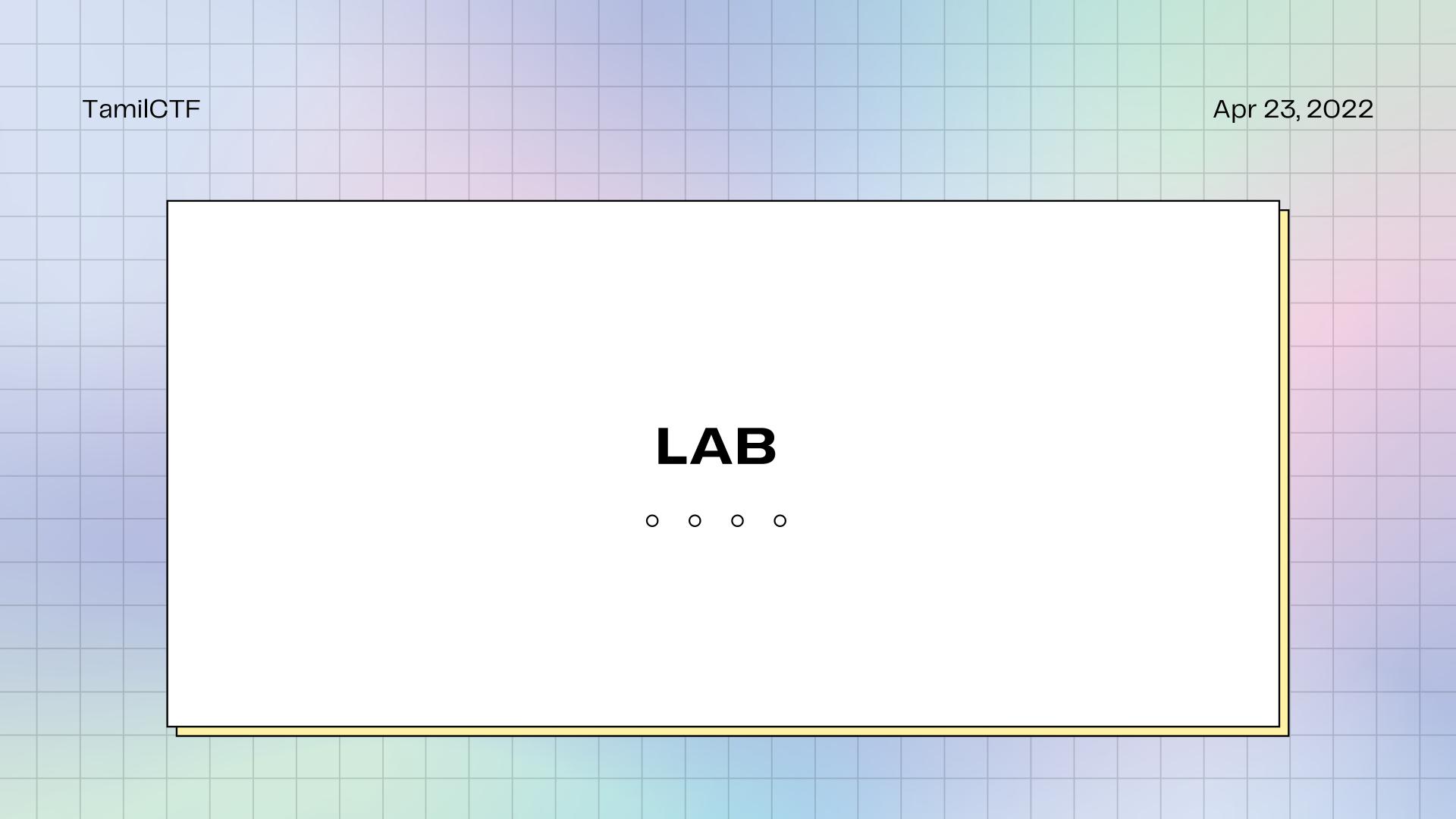
A CSRF token is a secure random token that is used to prevent CSRF attacks.

Storing CSRF tokens in HTML form can be Exploited.



Stealing CSRF token

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



Solution

0 0 0 0

Remediation For CSRF.

Get CSRF from Cookies

Make Sure to Set HTTP only Flag

In Addition to this, Implement a Perfect CORS



Dom Clobbering

0 0 0 0



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

0 0 0 0

```
<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



Solution

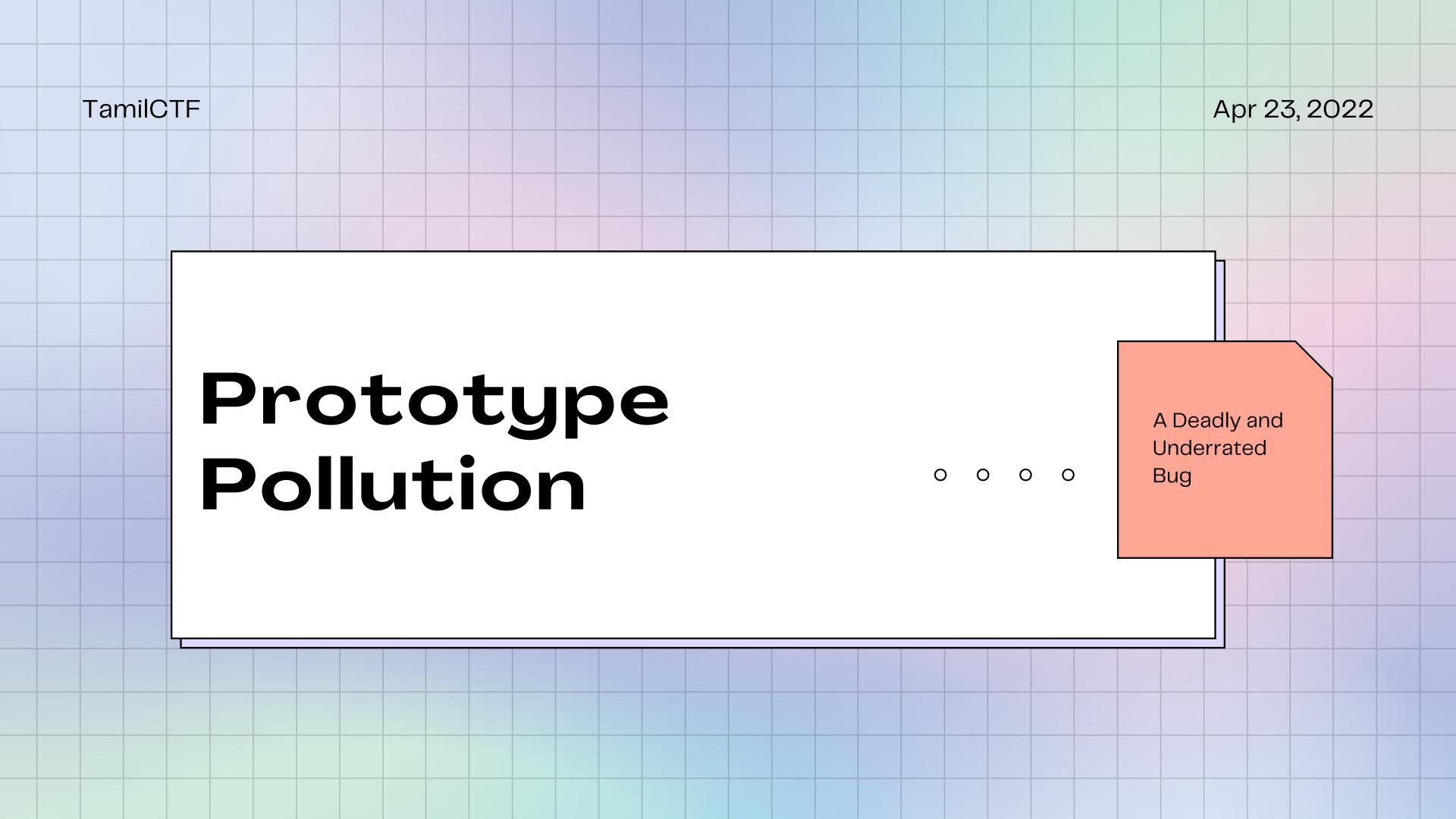
0 0 0 0

Remediation For Dom Clobbering.

Don't Use Undefined Variables

2 Avoid these Bad Code pattern

3 Use DOMPurify



Prototype Pollution

0 0 0 0



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

0 0 0 0

- 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

```
0 0 0
> var obj = {'name':'TamilCTF'}

    undefined

                                               The Basic Idea of Prototype
  obj. proto .isAdmin = true
                                                 Pollution is, Polluting the
true
                                              Present Methods in the Object
> var a = {}
                                               Prototype or Injecting New
undefined
                                                Prototype into the Object
                                             Prototype to Change the Logic
> a.isAdmin
                                                    of the Application
< true</pre>
```



Solution

0 0 0 0

Remediation For Prototype Pollution.

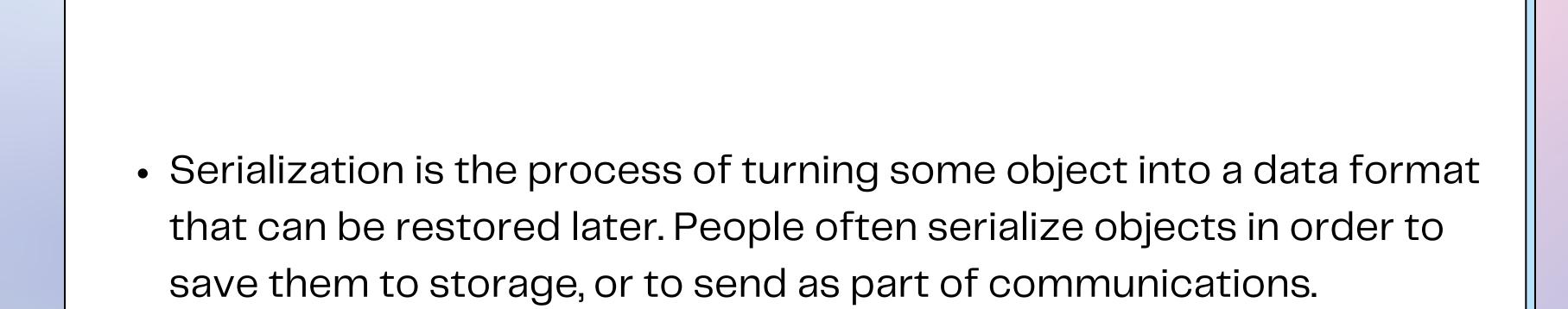
Always use latest library

Sanitize User Input Before Passing into Merging Operations



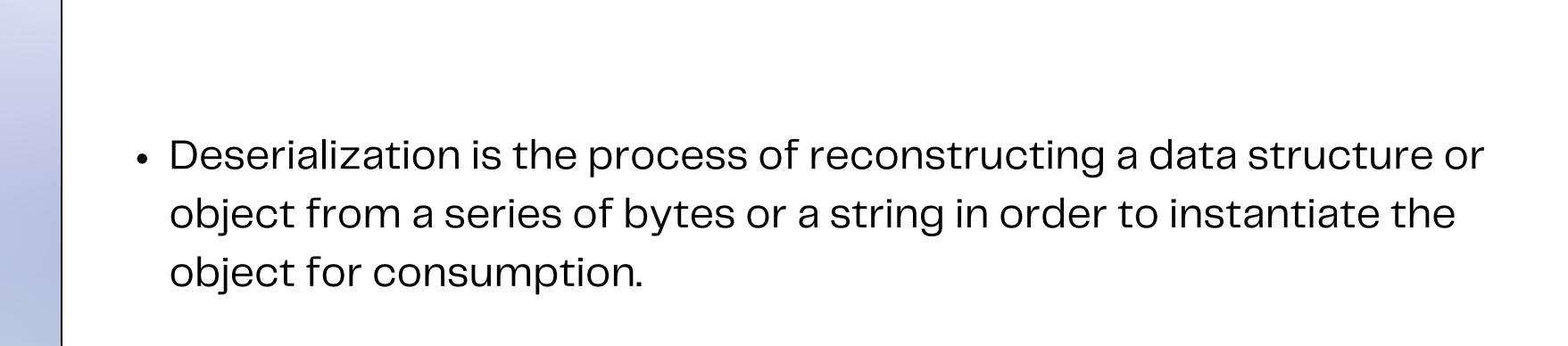
Serialization

0 0 0



Deserialization

0 0 0





Solution

0 0 0 0

Remediation For Node JS Deserialization

Don't Use

If necessary, Make Sure to Sanitize the Serialized User Input



PostMessage()

0 0 0 0



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

0 0 0 0



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

0 0 0 0



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)



Solution

0 0 0 0

Remediation For postMessage() Bugs

Always Check Origin Before passing input into DOM sinks.

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



