Agenda

Day 1

- OAuth Attacks
- BOLA
- Excessive Data Exposure
- Hacking JWTs

Day 2

- Security Misconfiguration
- Hacking GraphQL
- Mass Assignment
- CORS

API Security Bootcamp
Hands-On OWASP Top 10 for APIs

Dr. Sunny Wear

2023

Security Misconfiguration

Missing or Misconfigured Headers specific to APIs, Unpatched Systems, Misconfigurations

Security misconfiguration is commonly a result of insecure default configurations, incomplete or ad-hoc configurations, open cloud storage, misconfigured HTTP headers, unnecessary HTTP methods, permissive Cross-Origin resource sharing (CORS), and verbose error messages containing sensitive information.

What is API Security Misconfiguration?

GOTCHA: Taking phishing to a whole new level

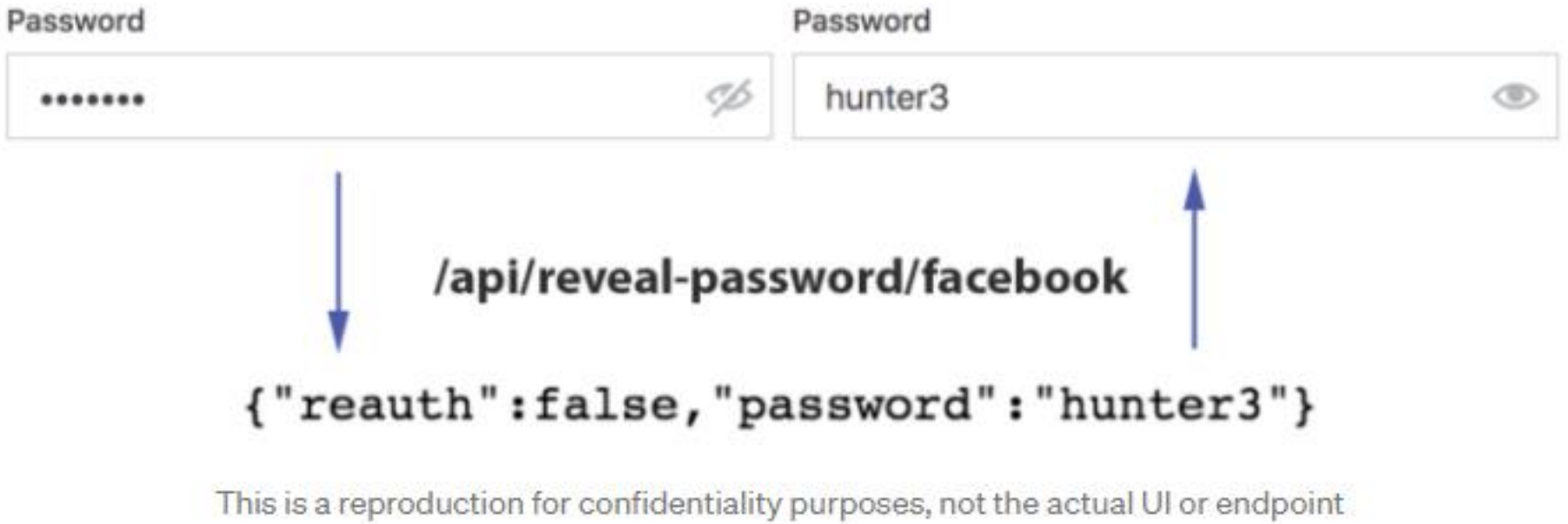
Inti De Ceukelaire Follow

Jun 28, 2019 · 4 min read

Why X-FRAME-OPTIONS matters on API endpoints

- **Clickjacking + Excessive Data Exposure + Lack of nosniff protection + a TON of creativity = PWNED!**

Why X-FRAME-OPTIONS matters!



GOTCHA: Taking Phishing to a whole new level

- API returning passwords:
  - The password manager's UI did not show plaintext passwords right away.
  - In order to view the password, you had to press a button that would trigger an XHR request to the endpoint ***/api/reveal-password/APP_ID***, which would reveal the password to the user."

GOTCHA: Taking Phishing to a whole new level

- However, **the site did not have X-FRAME-OPTIONS protection**, therefore, a user of the application could display the password of the JSON response within an iframe. "I noticed that the AJAX call did not have any X-FRAME-OPTIONS headers set and started thinking of possible security implications. Technically, this would allow users to display a user's password within an iframe:"

GOTCHA: Taking Phishing to a whole new level

No way to read that response as an attacker because of CORS. "Due to CORS, other websites **can not extract the password from the iframe**. So, unless we could find a way to visually extract the password from the iframe, this would be a dead-end."

However, this attacker created a very small iframe, large enough to fit one character of the password string into the iframe and he positioned it in such a way that the user would see only one character of that sensitive string.

GOTCHA: Taking Phishing to a whole new level

- And he did this a couple of times with a couple of characters next to each other and pretended it was a CAPTHCA. So, he applied some CSS filtering to make the password look like a real CAPTHCA with fuzzy, hard to read letters and ask the user to read the CAPTHCA to prove they were not a robot.
- What they did not realize is each user was entering their own password into the form field. **This happened because that API endpoint was allowing the rendering of the data in the browser.**

Clickjacking Protection

Best practice for APIs to protect against Clickjacking

1. X-Frame-Options: DENY
2. Content-Security-Policy: frame-ancestors ‘none’

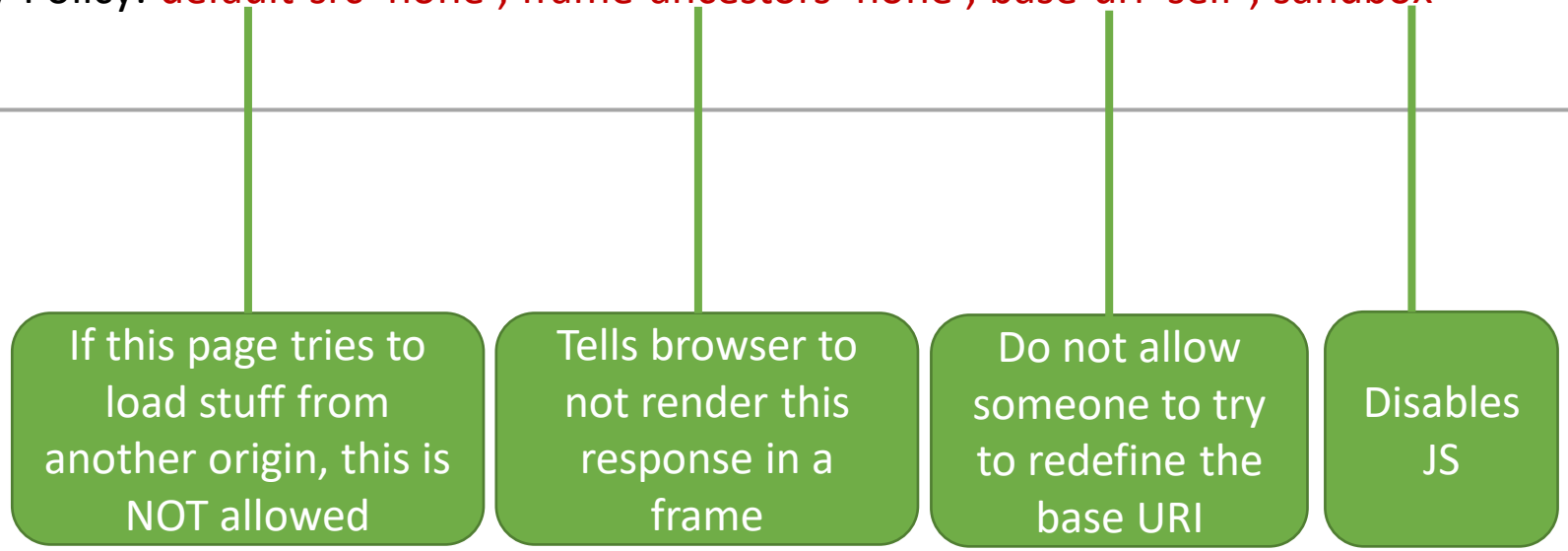
Content Security Policy for APIs

CSP provides the following:

1. Defines where resources are loaded (e.g., URL to remote JS file)
   a. Resources can be JavaScript, Images, CSS, Fonts, AJAX requests, Frames, HTML5 Media
2. Controls interactions with third parties
3. Common control to protect against cross-site attacks

Best practice for APIs for configuring CSP:

```
1  Content-Security-Policy: default-src 'none'; frame-ancestors 'none'; base-uri 'self'; sandbox
```



Best Practice Recommendation

Overview of best practice header configuration for APIs

```
1  Strict-Transport-Security: max-age 31536000; includeSubDomains; preload
2  X-Content-Type-Options: nosniff
3  Referrer-Policy: no-referrer
4  X-Frame-Options: DENY
5  Content-Security-Policy: default-src ‘none’; frame-ancestors ‘none’; base-uri ‘self’; sandbox
```

Exercise 5-1: CSP Bypass

Cross-site Scripting section, last lab:

Lab: Reflected XSS protected by CSP, with CSP bypass

EXPERT

LAB | Solved

This lab uses CSP and contains a reflected XSS vulnerability.

To solve the lab, perform a cross-site scripting attack that bypasses the CSP and calls the `alert` function.

Please note that the intended solution to this lab is only possible in Chrome.

Bypassing SameSite Lax restrictions with newly issued cookies

- No attribute defaults to Lax
- Cookies with Lax SameSite restrictions aren't normally sent in any cross-site POST requests, but there are some exceptions.
- To avoid breaking single sign-on (SSO) mechanisms, it doesn't actually enforce these restrictions for the first 120 seconds on top-level POST requests. As a result, there is a two-minute window in which users may be susceptible to cross-site attacks.
- This two-minute window does not apply to cookies that were explicitly set with the SameSite=Lax attribute.

Exercise 5-2: SameSite Bypass

Web Security Academy -> CSRF -> Bypassing SameSite cookie restrictions -> Lab

Lab: SameSite Lax bypass via cookie refresh

This lab's change email function is vulnerable to CSRF. To solve the lab, perform a CSRF attack that changes the victim's email address. You should use the provided exploit server to host your attack.

The lab supports OAuth-based login. You can log in via your social media account with the following credentials:

```
wiener:peter
```