Let's Write Secure Drupal Code!

(Application Vulnerabilities & Fixes)

Shreyal Mandot

Born and raised in Ratlam, Madhya Pradesh, India
Bachelor's and Master's from Pune
Technical Lead (L3)

@Shreyal999

What are we going to Cover ?

- What is Application Vulnerabilities
- Lets see **The OWASP Top 10 application vulnerabilities 2021.**
- Undestanding the Vulnerabilities.
- Points where we missed out to check in Drupal.
- How to fix the Vunlerabilities while writing code in Drupal 8.

What is Application Vulnerabilities ?

“An **application vulnerability** is a system flaw or weakness in an **application** that could be exploited to compromise the security of the **application**…”

What is OWASP ?

“The Open Web Application Security Project is an online community that produces freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security.…”


DEMO TIME

OWASP Top Web Application Security Risks

- Injection.
- Broken Authentication and Session Management
- Insecure Direct Object Reference
- autocomplete enabled
- Unprotected Cookie

OWASP Top 10 Web Application Security Risks

- Insecure deserialization
- Incorrect Exception Handling
- cross-origin resource sharing (CORS).
- Insecure Randomness.
- Cross-Site Scripting (XSS).

OWASP Top 10 Web Application Security Risks

- Cacheable HTTPS response.
- Content Security Policy (CSP).
- Session Timeout.
- Crawlable Links.

SQL & Header Injection

- **SQL Injection** : Executing SQL of the attacker.
- **Header Injection** : HTTP response header injection vulnerabilities arise when user-supplied data is copied into a response header in an unsafe way.

SQL Injection

```
<?php

$results = db_query(“SELECT uid, name and email from

{users} WHERE name LIKE ‘%%$user_search%%’”);

?>
```

Fixing SQL Injection

- Use Always Use Drupal Database API.
- Use Placeholders to pass variables to query.
- Filter Parameters.
- db_like().

Header Injection

Impact : An attacker can inject new HTTP headers and also, by injecting an empty line, break out of the headers into the message body and write arbitrary content into the application's response.

Fixing Header Injection

Added header in nginx add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always; add_header X-Content-Type-Options "nosniff" always;

Broken Authentication & Session Management

Broken Authentication allow attackers to compromise passwords, keys or session tokens, even going so far as to exploit other implementation flaws to assume users' identities temporarily or permanently.

Fixing Broken Authentication & Session Management

Unprotected Cookie

Description : When a cookie is set with the HttpOnly flag, it instructs the browser that the cookie can only be accessed by the server and not by client-side scripts.

Unprotected Cookie

Impact : If the HttpOnly flag is not set, then sensitive information stored in the cookie may be exposed to unintended user. Also if the cookie is an authentication cookie, then not setting the HttpOnly flag may allow a malicious user to steal authentication data (e.g., a session ID) and assume the identity of the user.

Fixing Unprotected Cookie

```
---·a/vendor/symfony/http-foundation/Request.php
+++·b/vendor/symfony/http-foundation/Request.php
@@·-14,6·+14,7·@@·namespace·Symfony\Component\HttpFoundation;
 use·Symfony\Component\HttpFoundation\Exception\ConflictingHeadersException;
 use·Symfony\Component\HttpFoundation\Exception\SuspiciousOperationException;
 use·Symfony\Component\HttpFoundation\Session\SessionInterface;
+use·Drupal\Core\Site\Settings;

 /**
  *·Request·represents·an·HTTP·request.
@@·-1246,6·+1247,12·@@·class·Request
         }

         $https·=·$this->server->get('HTTPS');
+        //·Custom·Patch·to·make·drupal·save·secure·cookie·as·drupal·is·not·able·to·set·https
+        $is_secure·=·Settings::get('is_elc_secure',·FALSE);
+        if($is_secure)·{
+            return·TRUE;
+        }
+        //·Custom·Code·Ends

         return·!empty($https)·&&·'off'·!==·strtolower($https);
     }
```

CROSS ORIGIN RESOURCE SHARING

There are a number of HTTP headers related to CORS, but the following three response headers are the most important for security :

CROSS PRIGIN RESOURCE SHARING

Access-Control: Allow-Origin specifies which domains can access a domain's resources. For instance, if requester.com want to access provider.com's resources, then developers can use this header to securely grant requester.com access to provider.com's resources.

CROSS PRIGIN RESOURCE SHARING

Access-Control : **Allow-Credentials** specifies whether or not the browser will send cookies with the request. Cookies will only be sent if the Allow-credentials header is set to true.

CROSS PRIGIN RESOURCE SHARING

Access-Control : **Allow-Methods** specifies which HTTP request methods (GET, PUT, DELETE, etc.) can be used to access resources. This header lets developers further enhance security by specifying what methods are valid when requester.com requests access to Provider.com’s resources.

CROSS SITE SCRIPTING (XSS)

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

Fixing CROSS SITE SCRIPTING (XSS)

- ‘add_header X-XSS-Protection "1; mode=block";’ in the nignx file
- Header set X-XSS-Protection "1; mode=block" in .htaccess. For apache.

Cacheable HTTPS response

Browsers can store a local cached copy of content received from web servers. Some browsers, including Internet Explorer, cache content accessed via HTTPS. If sensitive information in application responses is stored in the local cache, then this may be retrieved by other users who have access to the same computer at a future time.

Fixing Cacheable HTTPS response

Added header add_header Cache-Control "no-store, no-cache, must-revalidate post-check=0, pre-check=0";

AND also use Javascript to remove the last visited node using history function.