**How to run website: Execute level.py in each of folder and goes to 127.0.0.1:5000**

**Level1:**

A screenshot of a computer

Description automatically generated

Logic: At line 45, message = "Sorry, no results were found for <b>" + query + "</b>." it directly inserts the user's input into the HTML without any sanitization. An attacker can input malicious scripts as the query, and when the page renders, the script will execute. Thus, input such as <script>alert();</script> will easily be exewcuted.

Patch: By using html.escape(), we can sanitize the bad character to prevent execution of javascript. html.escape() is a Python function that transforms special characters in strings into their corresponding HTML escape codes, ensuring they're displayed as plain text rather than being interpreted as actual HTML or JavaScript. When web browsers display the escaped string, users see the original characters, but without the risk of unintended code execution.

CSP2.0:

The directive default-src: 'self' ensures that only resources from the same origin can be loaded. This is further reinforced by the script-src, style-src, img-src, and other directives, which restrict the sources of scripts, styles, images, and other resources to the same origin. When the application constructs the message Sorry, no results were found for <b>" + query + "</b>., even if an attacker attempts to inject malicious scripts or content into the query parameter, the CSP will block any scripts or styles not originating from the application's own domain.

CSP3.0:

A black background with white text

Description automatically generated

By setting the default-src to 'self', it ensures that only content from the same origin is permitted, effectively blocking any external malicious resources. Within the script-src directive, the 'self' value permits the execution of trusted scripts originating from the same domain. The inclusion of 'nonce-random-value' mandates that for a script to execute, it must possess this unique token, thereby ensuring that only explicitly approved scripts run. Any malicious script, which would typically lack this nonce, is instantly blocked. The 'strict-dynamic' feature, introduced in CSP 3.0, further enhances security. It allows scripts that have been dynamically added by nonce-approved scripts to be trusted implicitly, eliminating the need for extensive whitelisting. Additionally, the img-src 'self' directive ensures that images can only be sourced from the same origin, providing a safeguard against potential image-based attacks.

**Level2**

**A screenshot of a computer

Description automatically generated**

Logic:

A computer screen with white text

Description automatically generated

Inner.HTML has already block out the use of <script> tag because it do not execute, thus, the solution of level1 won’t work here. However, we cant still execute alert() without using script such as <img>. By inserting <img src='x' onerror='alert()'>, it will look for image ‘x’ which will fail miserably and then trigger the onerror attribute’s code.

Patch:

A computer screen shot of a program

Description automatically generated

I transitioned from using direct string concatenation with innerHTML to employing the DOM API for creating and manipulating HTML elements. This shift ensures that user-generated content, particularly posts[i].message, is treated as plain text by leveraging the textContent property, effectively neutralizing any embedded scripts. By constructing individual DOM elements for the message, date, and user details, and appending them directly to the container using the appendChild method, the revised approach embeds user content securely, eliminating the risk of executing potential malicious scripts and thereby mitigating XSS vulnerabilities.

A screen shot of a computer screen

Description automatically generated

CSP2.0: By specifying script-src 'self', the CSP ensures that only scripts hosted on the same origin as the website can be executed, effectively blocking any malicious inline scripts or external scripts from untrusted sources. The directive img-src 'self' http://ssl.gstatic.com restricts image sources to the same origin and the specified trusted domain, preventing potential image-based XSS vectors. The style-src 'self' 'unsafe-inline' allows styles from the same origin and inline styles, which might be a necessary compromise for design but can be tightened further.

Csp3.0:



The CSP directive script-src 'strict-dynamic' 'nonce-RANDOM1' ensures that only scripts with the specified nonce value (RANDOM1) can be executed. This prevents attackers from injecting malicious scripts unless they somehow know the nonce, which should be unpredictable and regenerated for each request. The script-src 'strict-dynamic' 'nonce-RANDOM1' directive is particularly interesting. Here, 'strict-dynamic' is a feature of CSP 3.0 that allows scripts specified with a nonce to dynamically load other scripts. This means that if a script with the correct nonce is trusted and loaded, it can further load other scripts without individually whitelisting them. The directive img-src 'self' http://ssl.gstatic.com restricts image sources to the same origin and the specified domain. The line html += "<blockquote>" + posts[i].message + "</blockquote"; is vulnerable to XSS if posts[i].message contains malicious scripts. However, with the CSP in place, even if an attacker injects a <script> tag into posts[i].message, it won't execute unless it has the correct nonce, effectively mitigating the risk of XSS attacks.

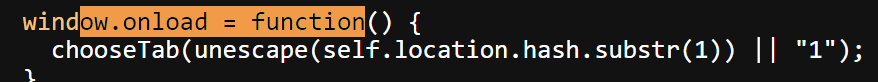
Level3

A screenshot of a computer

Description automatically generated

Logic:

The page's underlying code reveals that it retrieves the URL fragment and sends it to a function named choosetab. When the page loads, it executes:



This function dynamically updates the image tag's source based on the fragment text and then renders the updated tag on the page.

A screen shot of a computer code

Description automatically generated

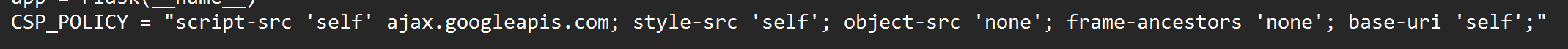
To trick this function, we can prematurely terminate the src attribute using a single quote. After that, we can introduce an onerror attribute with an alert function, similar to the previous level, and use double slashes to comment out the '.jpg' portion. <img src='/static/level3/cloud1' onerror='Alert();//.jpg' />

**Patch:** The main issue was the unsanitized num variable, sourced directly from the URL fragment and used to create dynamic content. This posed a significant XSS risk. To mitigate this, I sanitized the input to ensure num only held valid numbers. Using parseInt, it converted num to an integer, defaulting to 1 if the result wasn't a number or was outside the 1-3 range, given the three images. This ensured only safe values were used, preventing malicious script injections via the URL fragment. Additionally, I eliminated the use of the unescape function, which can decode potentially harmful strings, further reducing the risk of unintended script execution.

A computer screen shot of white text

Description automatically generated

CSP2.0:



By specifying directives like script-src 'self' ajax.googleapis.com, it restricts scripts to only be loaded from the website's own domain or Google's AJAX libraries, blocking external malicious scripts. The style-src 'self' directive ensures only local styles are applied, preventing malicious styling. The object-src 'none' directive completely blocks plugins, which can be vulnerability sources. The frame-ancestors 'none' directive stops the content from being embedded elsewhere, guarding against clickjacking. Lastly, base-uri 'self' ensures only local URLs can be base references.

CSP 3.0:

A black screen with white text

Description automatically generated

The directive default-src 'self' restricts content loading to the same origin, barring potentially harmful external resources. The script-src directive, bolstered by 'strict-dynamic' and a nonce ('nonce-random-value-here'), provides a robust barrier against unauthorized script execution. Specifically, the 'strict-dynamic' allows nonce-approved scripts to dynamically add other scripts, negating the need for an exhaustive whitelist. The nonce, a unique random value regenerated for each page load, ensures execution of only those scripts that carry this exact nonce value. This means that even if an attacker tries to inject malicious content, like 1' onerror='alert();// into the <img> tag's src attribute, the script within the onerror won't execute because it lacks the correct nonce.

Level4

A screenshot of a computer

Description automatically generated

Logic: The value supplied from the index page is directly injected into the function argument as the timer. <img src="/static/loading.gif" onload="startTimer('{{ timer }}');" /> This allows us to potentially tamper with the JavaScript being executed. By inputting the string: 3'\*\*alert());// This can injected into <img src="/static/loading.gif" onload="startTimer('{{ timer }}');" /> is a crafty XSS exploit. The 3' segment prematurely ends the string parameter of startTimer, turning subsequent characters into executable JavaScript. The \*\* is an exponentiation operator, but its real purpose here is to force JavaScript to evaluate its following operand, the alert() function. By trying to compute 3\*\*alert(), JavaScript unwittingly executes alert(), popping up a browser alert dialog.

Patch: The potential XSS vulnerability stems from the direct insertion of the timer variable into the onload attribute of an img tag. To mitigate this risk, I made several modifications. First, instead of embedding the timer value directly into the onload attribute, I stored it in a data-timer attribute of the img tag. Next, I introduced a new JavaScript function, initTimer, which is triggered upon the image's load. This function retrieves the timer value, sanitizes it to ensure it's a valid number, and confines it within a safe range. Only then does it call the original startTimer function with the sanitized value. By adding this layer of validation and indirect handling of the timer value, it address the XSS vulnerability.

A computer screen with text

Description automatically generated

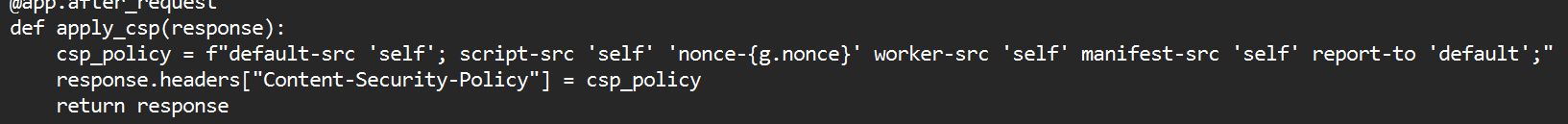
CSP 2.0:

A black background with white text

Description automatically generated

The directive default-src 'self' ensures that, by default, only resources from the same origin as the webpage can be loaded. This blocks attackers from loading malicious assets from external domains. The directive script-src 'self' 'nonce-{g.nonce}' dictates that scripts can be loaded from the same origin ('self'), and any inline script requires a nonce that matches the one generated server-side ('nonce-{g.nonce}'). This prevents attackers from injecting arbitrary inline scripts, as they can't predict the nonce value.

CSP 3.0:



The default-src 'self'; ensures that resources are loaded only from the site's origin. The more nuanced script-src directive, which includes 'self', 'nonce-{g.nonce}', and 'strict-dynamic', has a threefold function. It allows scripts from the same domain, requires inline scripts to possess a matching nonce (a unique token) to execute, and, thanks to CSP3.0's 'strict-dynamic', permits scripts approved by nonce or hash to trust and load other scripts, even if their sources aren't explicitly whitelisted. This dynamic capability increases flexibility while retaining security. By mandating scripts and workers to originate only from trusted sources and allowing trusted scripts to pull in additional resources, this CSP significantly mitigates the risk of XSS attacks.

Level5

A screenshot of a computer

Description automatically generated

Logic: <a href="{{ next }}">Next >></a> The anchor tag <a href="{{ next }}">Next >></a> has its href attribute populated by a template variable {{ next }}. This means that the value of the href attribute can be controlled by whatever value is passed to next. When I set the next parameter's value in url as javascript:alert(), the resulting anchor tag becomes: <a href="javascript:alert()">Next >></a>. When I click on the "Next" link, instead of navigating to another webpage, the browser will execute the JavaScript code specified in the href attribute, which in this case is alert(). This will cause an alert dialog to pop up.

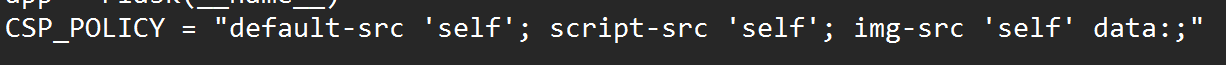
Patch: I introduced a sanitize\_next\_url function in the level.py script. This function parses the provided URL to discern its scheme. If the scheme matches any known malicious patterns, such as javascript:, data:, or vbscript:, the function redirects to a default safe URL, effectively neutralizing the threat. By doing so, even if an attacker attempts to inject a harmful payload, our sanitization logic ensures it won't execute. In the main route function, I applied this sanitization to the next parameter before it's passed to the template.

A screen shot of a computer program

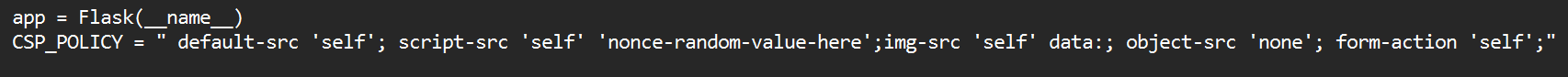
Description automatically generated A screen shot of a computer program

Description automatically generated

CSP2.0:



The CSP directive script-src 'self' ensures that only scripts hosted on the website's own domain can be executed, blocking any external or inline scripts. This is crucial because even if an attacker manages to inject a script into the web page, the browser will refuse to execute it due to the CSP restrictions. The directive img-src 'self' similarly restricts image sources to the same origin.CSP3.0:



By specifying default-src 'self', the CSP restricts all content sources to the domain it originates from, preventing attackers from loading malicious resources from external sources. The script-src directive limits script execution to those from the same origin and scripts with a specific nonce value, barring unexpected inline scripts from executing. The img-src directive ensures images are loaded only from the website itself or valid data URIs, negating malicious image payloads. By setting object-src to 'none', potential vulnerabilities from plugins like Flash are eliminated. Lastly, form-action 'self' ensures that only the site itself can be the recipient of form submissions, safeguarding user data.

Level6

A screenshot of a computer

Description automatically generated

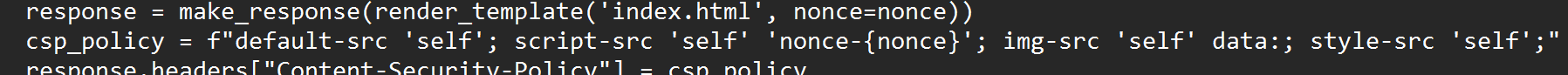
Logic: The intention of this website is to import other resources from its backend, that's why it tries to filter the http/https input to prevent users import malicious javascript files from the internet. However, there are ways to bypass the filter. The data URL is another way of involving files instead of the traditional URL request. The intention of Data URLs is to provide a way to embed small files inline with HTML and avoid establishing additional HTTP connections to request those files. The data URL usually has the form of data:[<mediatype>][;base64],<data>. And the data part supports javascript execution. Thus, we can use URL such as <https://xss-game.appspot.com/level6/frame#data:text/plain,alert('1')> to raise alert.

Patch: First, the JavaScript was externalized, moving it from inline within the HTML to a separate file named main.js. This separation not only makes the code cleaner but also reduces the risk of inline script execution. Next, within the includeGadget function, a whitelist approach was adopted. Instead of blindly loading any script based on the URL fragment, the function now checks the requested script against a predefined list of allowed gadgets (allowedGadgets). If the requested script isn't on this list, it won't be loaded, thereby preventing the loading of unauthorized or malicious scripts. Additionally, the initialization of the script was shifted to the onload event of the <body> tag, ensuring that the script's functions are set up as soon as the page is loaded.

A screen shot of a computer

Description automatically generated

CSP2.0:The ‘default-src 'self' restricts all content sources to the same origin as the document unless otherwise specified. This means that by default, only resources from the website's own domain can be loaded. The script-src 'self' 'sha256-kwaxfc8OCWPP3JdOXI7ylzQ3FN0a8CKgY9FBi90npjo=' directive allows scripts from the website's domain and also permits inline scripts that match the specified SHA-256 hash. This ensures that only known and approved scripts run on the webpage. The img-src 'self' data: directive permits images from the website's domain and also allows inline images using the data: scheme. Lastly, style-src 'self' ensures that only stylesheets from the website's domain are applied.



CSP3.0: The directive default-src 'self' ensures that resources are only loaded from the same origin, blocking any external or inline scripts or styles that haven't been explicitly allowed. The script-src 'self' 'nonce-{nonce}' directive further tightens security by requiring scripts to have a specific nonce (a random token) to execute. This means that even if an attacker can inject a script into the page, it won't run unless it has the correct nonce, which is regenerated on every request. The inclusion of data: in img-src allows for the safe embedding of images using data URIs