Penetration Test Report

Jan Sneeuw

July 20, 2025

Abstract

This report is a summary of the most severe vulnerabilities discovered during ethical penetration tests carried out on the domain `bikeseek.org`. This domain is owned by an acquaintance who gave explicit permission to run various tests on his site while still in development. The sole constraint of our agreement was not to run any unrestricted and unsupervised DoS-type attacks. Apart from basic web-hacking strategies, most vulnerabilities below are discovered by using tools such as Burp Suite and the Zed Attack Proxy program.

0.1 Good Practices

Before discussing vulnerabilities, we argue that the domain is well-organised and follows a modern standard, so we expect very few vulnerabilities to begin with. The website incorporates a 2FA process to login a user via their Google account, instead of natively using login forms with PHP, MySQL, etc. in the backend. Through normal practice on the site, one finds that the only publicly visible URLs are `bikeseek.org` and `bikeseek.org/my_results`, where one is redirected to after uploading an image. As is described below in more detail, even with more advanced tools like ZAP we don't find any further subdomains, so there is not much of an attack surface for, say, IDOR vulnerabilities. As is standard practice, the command `curl -I http://bikeseek.org` throws a 301 and redirects to the HTTPS page `https://bikeseek.org`.

1 Discovered Vulnerabilities

1.1 Upload Path Leak

The standard format for uploads is `.jpg` and `.jpeg`. However, if we create an arbitrary file `textfile.txt` and rename it to `textfile.txt.jpg` and try to upload it, the website throws the following error:

```
An error occurred during search: cannot identify image file '/home/ec2-user/BikeSearch/src/
    uploads/user_3_query_09c63e039b7448efad76d697662b5eb6_textfile.txt.jpg'
```

First, notice that this reveals part of the folder-structure of the web-server, namely we see all the folders in the path `/home/ec2-user/BikeSearch/src/uploads/`. Moreover, we find that we are logged in as `user_3`, which indicates that at any point, if we log in with a fresh Google account we are able to deduce the number of users of the website at any given point in time. The folder `ec2-user` hints at the website running on an AWS EC2 instance, which will be confirmed later. Lastly, we see the unique MD5 hash `09c63e039b7448efad76d697662b5eb6`, which changes for every upload, even when uploading the exact same file twice, meaning that whatever string is hashed likely involves the current time or date of upload, possibly in addition to other sub-strings such as image name, user ID, user name etc.

1.2 First Attempt at XSS Using Burp Suite

We set the browser proxy to route traffic through Burp Suite and proceeded to upload an image file via the application's regular upload functionality. In the intercepted request within Burp's proxy interceptor, we attempted to manually modify the image's filename — for instance, replacing it with:

`"><script>alert(1);</script>.jpg`

The line `<img src="/uploads/user_3_query_HASH_image.jpg">` is included in the HTML of the resulting subdirectory `/my_results`, hence the goal was to embed the line

`<img src="/uploads/user_3_query_HASH_"><script>alert(1);</script>.jpg">`

in the backend, thereby triggering a reflected XSS attack.
However, the issue with this vector lies in its leading `"` character, which prematurely terminates the `filename="image.jpg"` attribute in the multipart form-data request when we replace `image` with the payload. As a result, the structure of the HTTP request becomes invalid, and the intended injection does not propagate into the rendered page as planned.

1.3 Successful XSS Using Burp Suite

An alternative approach yielded a successful stored XSS attack on the domain. We give a full explanation how to obtain this result:

1. Proxy the browser to route any traffic through Burp Suite.
2. Upload a vector graphic image with file extension `.svg` to the website and intercept the POST request in Burp Suite.
3. Change the POST content, i.e. the block `<?xml ... </svg>`, to the following snippet, containing a classical Javascript payload:

```
<?xml version="1.0" standalone="no"?>
<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
  <script type="text/javascript">
    alert(1);
  </script>
</svg>
```

4. Forward the requests and return to the website. The response is a visible error message printed on screen. In our case (we uploaded `bike.svg`), it reads

```
An error occurred during search: cannot identify image file '/home/ec2-user/
BikeSearch/src/uploads/user_3_query_7a0586d404254425a0e7e5c43050a4c7_bike.svg'
```

5. Thus, it seems as though the test-image `bike.svg` has failed to be uploaded to the database. However, if we copy the last part of the file path, i.e. `user_3_query...0a4c7_bike.svg`, navigate to "View Last Search" and open an arbitrary image in a new tab, we see that its URL is of the form

```
https://bikeseek.org/upload/user_3_query_HASH_image.jpg
```

6. Now replace the last part of this path file by the copied unique identifier of `bike.svg` from step 5. The URL `https://bikeseek.org/upload/user_3_query_7a0586d404254425a0e7e5c43050a4c7_bike.svg` gives the desired XSS attack in our case, as we obtain the expected alert in the browser.

   Some observations that were made about this vulnerability:

   - It was necessary to upload an `svg` file, as only these image types support XML, thus enabling us to smuggle in some Javascript code.
   - Despite the unique identifier including the string 'user_3_query', the XSS exploit is not unique to us as a user. It was tested with the owner's account that whenever they are logged into the domain they can still access the URL from step 6 above and get the alert in the browser.
   - This attack is categorized as a *stored* XSS vulnerability, since our payload is permanently uploaded in the owner's database. This was checked with the owner. Nonetheless, the security impact is not extremely high for a few reasons:
     - Any images that users upload are not reused in the backend to search for fitting bikes using ML-methods. Thus, it is impossible for another user to be presented with our malicious file `bike.svg` on their session.
     - We cannot extend our XSS attack to steal cookies, since the domain has the setting `httponly` set to `true`, and thus we cannot successfully implement a payload like `alert(document.cookie)` in the snippet.
   - One possible way to further exploit this vulnerability is to redirect to a new webpage upon clicking on the malicious URL. This is done with the function

     ```
     eval(location.href = "https://www.malicious-website.com/")
     ```

     instead of the function `alert(1);` from before.

1.4 Unsecured Cookies

By inspecting the website we obtain a cookie for our session. An example of this is:

```
.eJwlzjkOAjEMAMC_pKZwHJ98BsWJLZCodkWF-
 DtI1NPMu93qyPPerjWfZ17a7bHbtYmBKRPtGGomHTcLGIIZTRQAYVbotJe4YnHGxlBfc6ZPz0Ax8
dCfj0AomVzayULHiCpbqegVEjQJtyxy7uU8GKnngJTVfpHXmcd_M9rnC00eLlk.aF2zbQ.
 XdhBWfBvuyUkyMnvwGuhcHST7rA
```

Notice that the cookie is given in base 64. With `cyberchef.org` we can translate this using the "From Base64" and "Zlib inflate" recipes to plain text, in which case it will read the following:

```
{"_fresh":false,
"_id":"68087544db3788612d560820884a26006557014dc6972f5e
    bd2b79caae9a9eb26869b74dc3b20f6a5f7148b733bff8ce72
    9fb6b4a42d6c4951f9535241e30e6c",
"_user_id":"3"}
```

Once again,we confirm that we are logged in as `user_3` and additionally we are presented with a likely SHA-512 hash, which we tested to be different from session to session, and is again most likely the hash of a string containing user ID, date, etc.

1.5 Uploading Images with Embedded Information

I minor vulnerability, similar in spirit to the uploaded `svg`-files but less dangerous, is that it is possible to alter legitimate `jpg`-files using software like exiftools to embed messages into images. A proof of concept has been uploaded to the website, but without any further implications.

2 Zed Attack Proxy

An automated attack against `bikeseek.org` was launched and resulted in a total of 31 alerts. Of these, 4 were categorized as *medium* severity, 8 as *low* and 19 as *informational*. Below we give information about the four most severe risks, taken from the report generated by the ZAP itself:

1. **Content Security Policy (CSP) Header Not Set**: Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

2. **Cross-Domain Misconfiguration** (Bootstrap in the backend, `https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css`): Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server. The CORS misconfiguration on the web server permits cross-domain read requests from arbitrary third party domains, using unauthenticated APIs on this domain. Web browser implementations do not permit arbitrary third parties to read the response from authenticated APIs, however. This reduces the risk somewhat. This misconfiguration could be used by an attacker to access data that is available in an unauthenticated manner, but which uses some other form of security, such as IP address white-listing.

3. **Missing Anti-clickjacking Header**: The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options. Using the Burp Suite 'Clickbandit' function a proof of concept `clickjacked.html` was created, showing that, principally, click-jacking is possible.

4. **Vulnerable JS Library**: The library `https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js` is used in the backend. The identified library bootstrap, version 4.5.2 is vulnerable, see also CVE-2024-6531. More precisely, an anchor element `<a>`, when used for carousel navigation with a data-slide attribute, can contain an `href` attribute value that is not subject to proper content sanitization. Improper extraction of the intended target carousel's #id from the `href` attribute can lead to use cases where the click event's `preventDefault()` is not applied and the `href` is evaluated and executed. As a result, restrictions are not applied to the data that is evaluated, which can lead to potential XSS vulnerabilities.

The remaining alerts constitute warnings such as 'Cookie Without Secure Flag', 'Server Leaks Version Information via "Server" HTTP Response Header Field', 'Timestamp Disclosure - Unix' and mainly 'Tech detected', but this is hardly an issue, as more easily accessible web-developer tools such as Wappalyzer perform the same task within a browser.

3 Additional Tools and Vulnerabilities

3.1 Nmap

One of the most commonly used tools in penetration testing to scan for hosts and open ports is Nmap. We used `nmap bikeseek.org` to obtain the following output:

```
Nmap scan report for bikeseek.org (54.175.64.247)
Host is up (0.000096s latency).
rDNS record for 54.175.64.247: ec2-54-175-64-247.compute-1.amazonaws.com
Not shown: 705 filtered tcp ports (net-unreach), 287 filtered tcp ports (no-response)
PORT    STATE  SERVICE
22/tcp  open   ssh
80/tcp  open   http
111/tcp closed rpcbind
135/tcp closed msrpc
139/tcp closed netbios-ssn
161/tcp closed snmp
443/tcp open   https
445/tcp closed microsoft-ds

Nmap done: 1 IP address (1 host up) scanned in 23.62 seconds
```

This result also supports our earlier claim that the web server is an EC2-instance on the Amazon AWS cloud, and hence we can assume that (even though we did not test this explicitly) the site is well-defended against DoS-type attacks. However, we also see that the SSH port 22 is open, which is slightly uncommon. Entering `nc bikeseek.org 22`, we get the output `SSH-2.0-OpenSSH_8.7`, i.e. the server is running on OpenSSH version 8.7. In particular, this means that, technically, the site is susceptible to CVE-2024-6387, even though exploits of this type are uncommon in practice, and have almost exclusively been successful in lab-conditions.

3.2 DoS Attack

under supervision of the owner of the domain, we started a very small instance of a DoS-attack. To this end, we simply uploaded a legitimate jpg-file to the website, intercepted the POST request in Burp Suite, sent it over to the repeater, duplicated the tab 100 times and sent the request via the function "Send group in sequence (single connection). As expected, the server had no troubles with such a small amount of requests and threw no errors. It was observed on the owner's AWS monitoring page that the CPU usage climbed from roughly 2% to 4% and all other parameters easily stayed within bounds. No further, more impactful, DoS-attacks have been launched since.

3.3 Brute-Force Image-Traversal

We were able to successfully reverse engineer an image entry from a friend's database — specifically, the bike associated with product ID `3080448350` on `kleinanzeigen.de`. To achieve this, we systematically browsed through various bike listings on Kleinanzeigen and manually tested a sequence of product IDs directly within the application's URL structure until we identified one that returned a valid match. Notably, this particular image was not among those previously displayed within the `my_results` subdirectory, indicating that it represents new, previously unlisted information within the application's dataset.

3.4 Imprint

As per German law, any website catering products or services to the public is obliged to include an imprint. So far, this is not done on the website, which is a big oversight. However, seeing as the site is still in development, this is not a major issue.

4 Mitigation

We conclude this report by providing a list of recommendations to the owner on how to best mitigate any vulnerabilities that were discovered.

1. To prevent further XSS attacks, directly deny the ability to upload `svg`-files. Alternatively, include CSP (Content Security Policy) headers, as recommended by ZAP, to mitigate any XSS risks that we might not have discovered. As per the ZAP report: "Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header."

2. Cross-Domain Misconfiguration, taken from the ZAP report: "Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance). Configure the 'Access-Control-Allow-Origin' HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allow the web browser to enforce the Same Origin Policy (SOP) in a more restrictive manner."

3. Missing click-jacking header: "Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app. If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's 'frame-ancestors' directive."

4. Vulnerable JS Library: Upgrade to the latest version of the affected library. Since `bikeseek.org` does not make use of carousel navigation, this mitigation is almost redundant, but nonetheless good practice.

5. Imprint: Include a basic imprint as issued by German law, at least as soon as the site is actively being used by customers.