7 Overlooked recon techniques to find more vulnerabilities

BY BLACKBIRD-EU · JANUARY 13, 2025 · LAST UPDATED ON MARCH 6, 2025

Reconnaissance is an important phase in bug bounty and in pentesting in general. As every target is unique and as we often do not have access to the code base, we'd need to come up with unique methods to gather useful and accurate data about our target to help us find vulnerabilities.

In this article, we will be covering 7 overlooked reconnaissance techniques that you can apply to your bug bounty target to gather more useful data and find more security vulnerabilities!

Let's dive in!

What is recon?

Recon (short for reconnaissance) is the act of gathering (useful) information to help with detection and exploitation phases later on in your security tests. Reconnaissance or information gathering also forms the first (and most important phase) of any pentest.

Bug bounty hunters who spend time performing recon are almost always rewarded well for their efforts as they often come across forgotten assets or hosts that haven't received the same security attention as the main application, making them more prone to security vulnerabilities.

As every target is unique, we will need to try several different methods to accurately gather as much useful data about our target as possible to help us find more vulnerabilities.

Below we outline 7 different techniques that are often overlooked but are still effective in finding new API endpoints, app routes and input parameters. Let's take a look at each one of them in detail.

Overlooked recon techniques

Targeted wordlists

Bruteforcing (or fuzzing) is an integral part of discovering unreferenced and hidden hosts, API endpoints, app routes or input parameters. However, generic wordlists are not always accurate and can miss out on finding critical endpoints.

A more recommended approach is using targeted wordlists that are crafted based on your target. Tools like CeWL can help you generate custom wordlists based on product and API documentation or help resources.

You can later use this custom wordlist to find more undocumented and unreferenced API endpoints that you can test for security vulnerabilities.

Virtual host (VHost) enumeration

Most hosts and servers deployed today serve multiple applications. For example, multiple (sub)domains may point to a single IP address or host that has a running reverse proxy server (such as Nginx). This proxy server will determine based on the host header what application to serve.

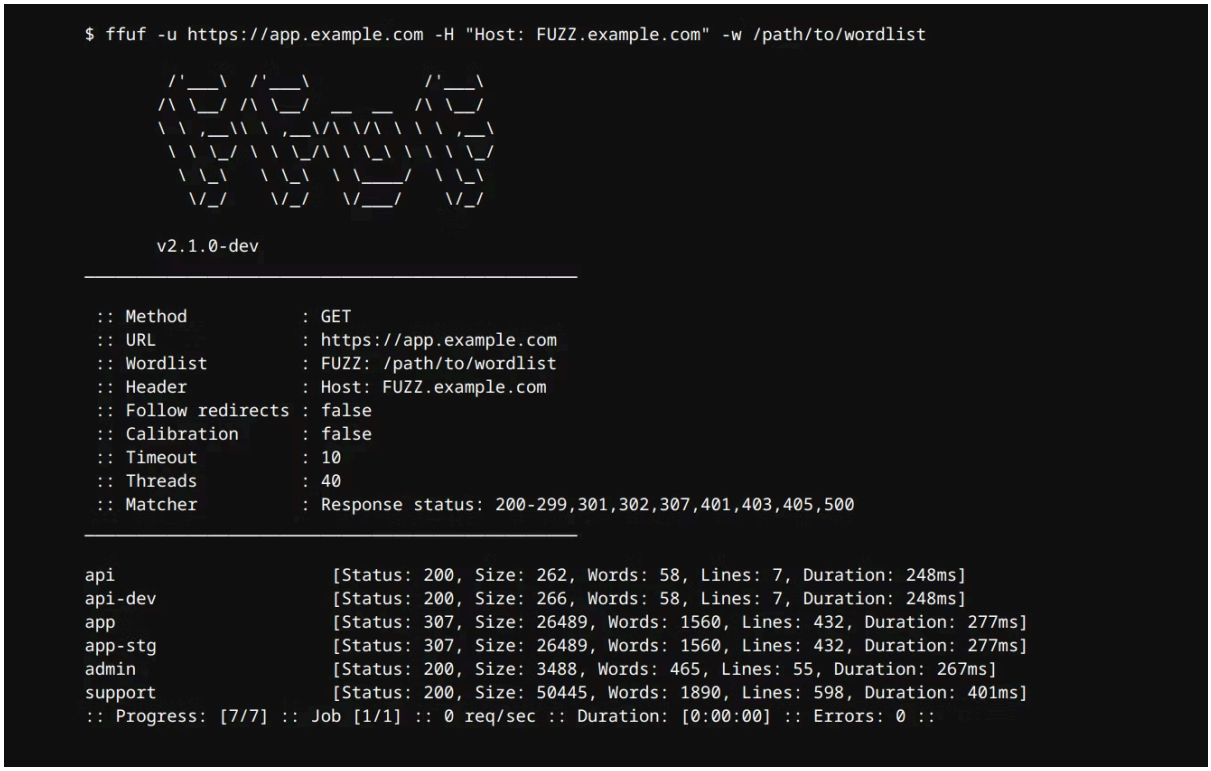
Take a look at the example configuration file below:

```
# Nginx reverse-proxy configuration
server {
    server_name app.example.com api.example.com;
    location / {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
    }

    # ...
}

server {
    server_name app-stg.example.com api-stg.example.com;
    location / {
        proxy_pass http://localhost:8081;
        proxy_set_header Host $host;
    }

    # ...
}
```

We can look for possible virtual hosts on our target by bruteforcing the host header. And we can do this with Ffuf for example:

```
$ ffuf -u https://example.com -H "Host: FUZZ.example.com" -w /path/to/wordlist
```

Afterward, once we've enumerated all virtual hosts, we can easily test each one of these applications individually.



Example of a virtual host scan using Ffuf

Forced browsing using different HTTP methods

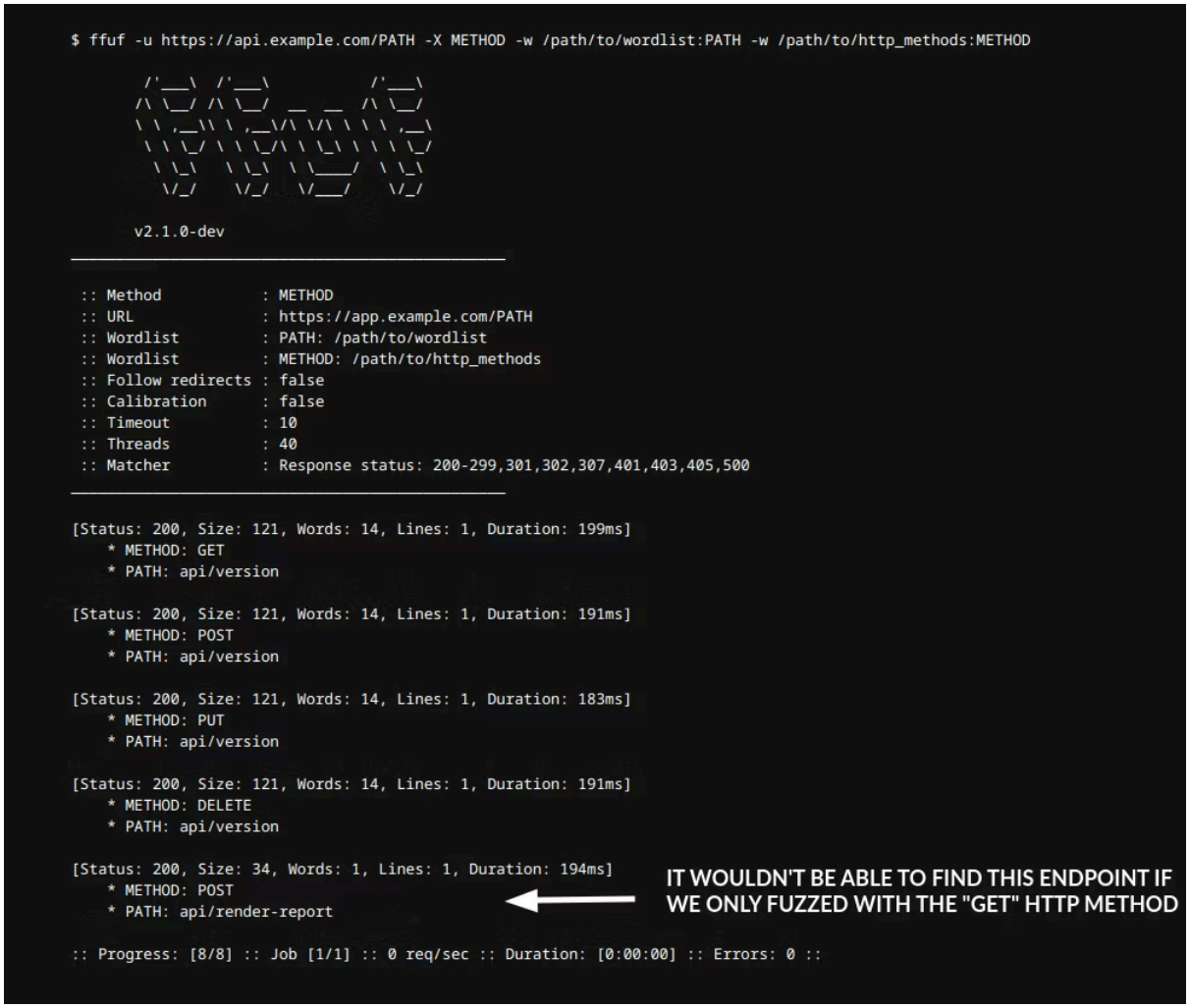
Take a look at the following code snippet:

```
import express, { Request } from 'express';

const app = express();
const port = 8080;

...

app.post('/api/render-report', async (req: Request, res: any) => {
  const { data } = req.body;

  const res: string = await renderReport(data);

  res.send(res);
});

app.listen(port, () => {
  console.log(`Server is running at port ${port}`);
});
```

We can see on line 8 that an API endpoint has been defined that only accepts the POST HTTP method. We would've never been able to find that specific API endpoint if we only bruteforced using the GET HTTP method, the default request method for most fuzzing and content discovery tools.

It's therefore recommended to always test for multiple HTTP methods, with Ffuf, you can easily do so by specifying the HTTP method option:

```
$ ffuf -u https://api.example.com/PATH -X METHOD -w /path/to/wordlist:PATH -w /path/to/http_methods:METHOD
```



The "/path/to/http_methods" is the path to a wordlist file containing all your HTTP methods separated by a newline feed character.

Using this fuzzing approach will ensure that you can also take these potential edge cases into account when certain API endpoints or application routes only respond when a specific HTTP method is used.

JavaScript file monitoring

JavaScript file monitoring comes with benefits as it can help you get notified when your target application gets updated and new API endpoints, app routes and input parameters have been referenced.

It's also an effective approach as it ensures that you'll be one of the first to test a new component, feature or API endpoint.

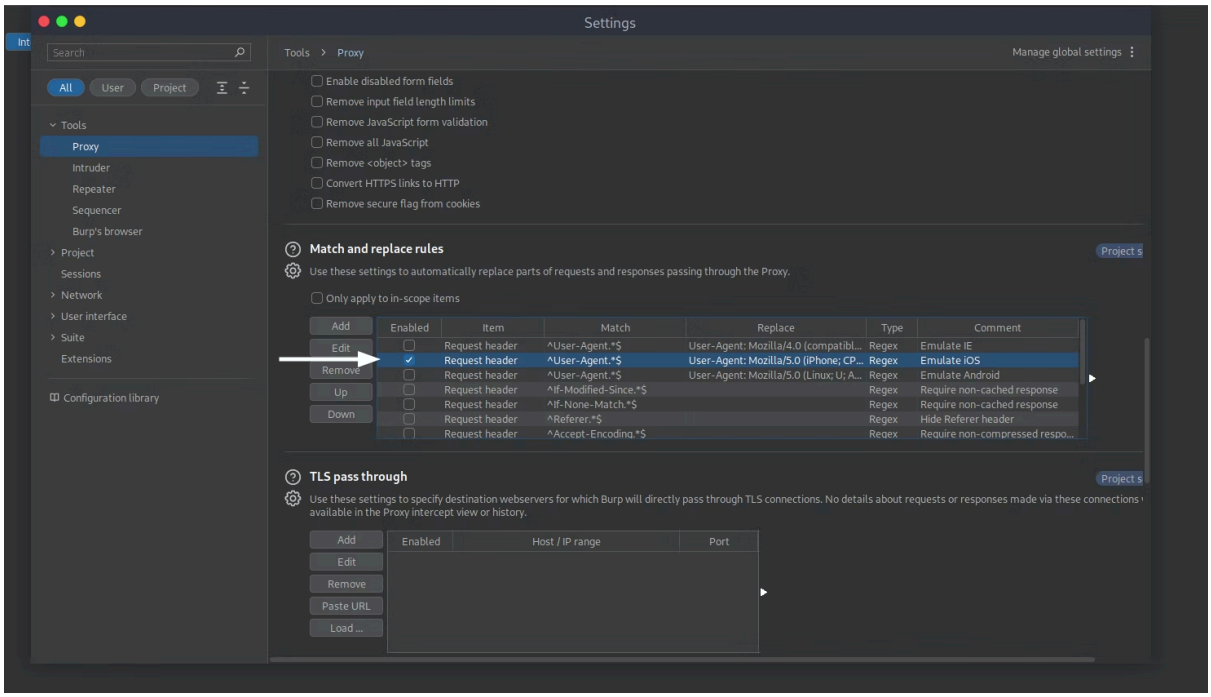
However, it does require you to set up a server that you can use to monitor JavaScript files periodically. Luckily for us, there are several open-source tools that you can make use of (such as JSMON), some of which even provide support for notifications.

Crawling with different user-agent headers

Some servers serve different versions of the application based on the user agent. This approach is most commonly used to provide extensive support for smaller screens such as mobile devices.

Therefore, it's always recommended that when crawling or intercepting requests, to set a mobile-specific user agent as well to try to discover differences in server responses and hidden functionality such as mobile-specific app routes and API endpoints.

Application versions designed for mobile devices may also contain different features. Next time when testing your target, try changing your user-agent to emulate an iOS or Android device and look for changes or application routes and API endpoints that you haven't come across earlier through the desktop version.



Configure your proxy interceptor to emulate a mobile device

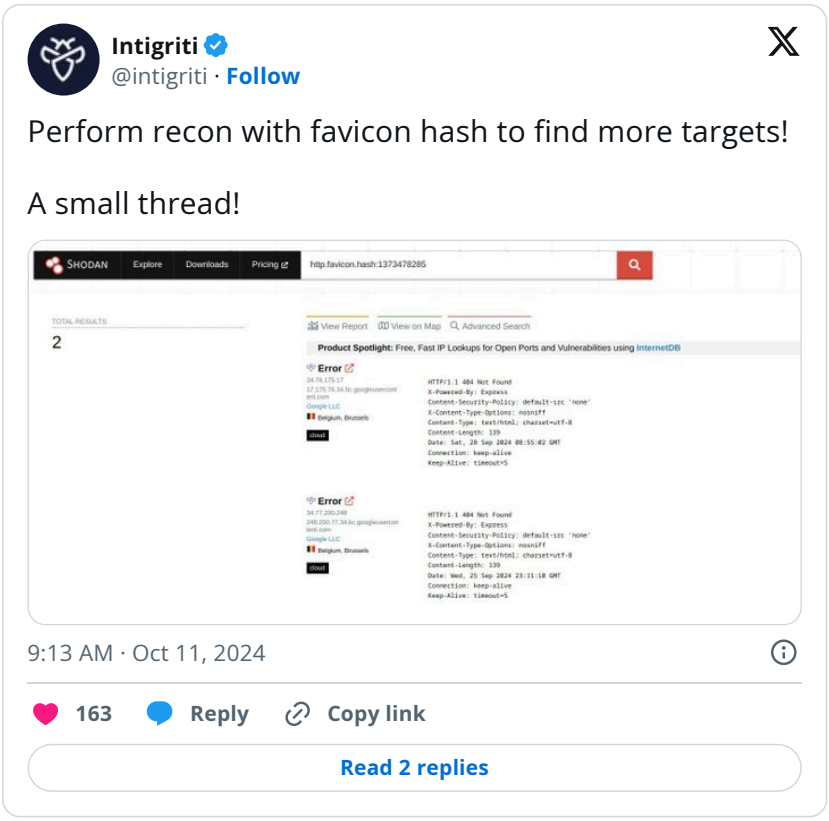
Finding related assets with favicon hashes

Some hosts are set up for development or administrative purposes only and have no (sub)domain pointing to them. They do have a few similarities that can help us track these down, and this is often the public IP or IP/CIDR range they are in.

In some cases, especially when your target is a smaller company, it may not have a reserved IP space, or the company regularly deploys hosts in various regions across the world using third-party cloud services like AWS or Azure. We can in that specific context still identify hosts that belong to your target using the favicon hash.

Favicons are the small icons that appear on the web browser's tab next to your page title. And these same icons are often used as favicons on new hosts as well. We take the hash and use tools like Shodan and Censys to list all hosts with a matching favicon hash.

In case you want to dive deeper, we recommend you read the following detailed guide:



TIP! Want to find more assets related to your target? Leverage internet search engines like Shodan & Censys to discover more untouched attack surfaces!

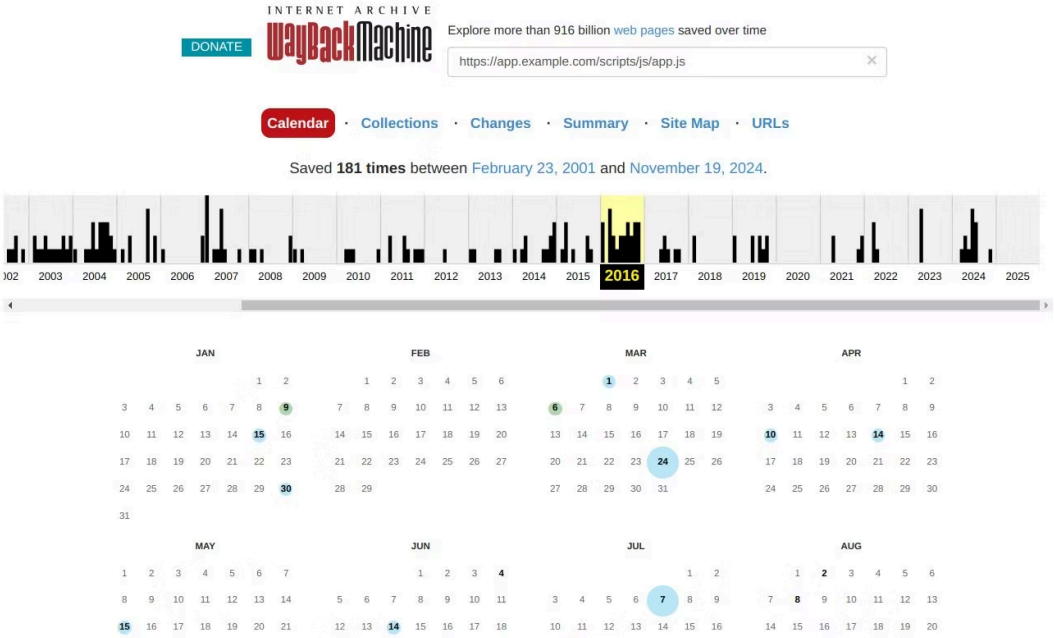
Looking up legacy versions of JavaScript files

Public code repositories and internet archives can store several versions of JavaScript files. And as JavaScript files are always the go-to resources for discovering API endpoints and app routes, we can leverage these services such as GitHub and the Internet Archive to find API endpoints in legacy code.

These API endpoints may have been removed from client-side code but may still be accessible.

Looking up legacy JavaScript files can also reveal sensitive information and vulnerabilities that may still be exploitable in current versions, such as hard-coded secrets that have never been revoked or rotated.

Taking this into consideration, whenever you're analyzing an interesting JavaScript file, make sure you always look up any previously archived versions of it. There are also automated tooling that can help you automate this entire process.



Example of a legacy version of a JavaScript file

Conclusion

We all understand the importance of reconnaissance in bug bounty. Even though not every target is the same, we can still apply creative techniques to gather useful information that could lead us to more security vulnerabilities. Moreover, these recon techniques can be particularly effective when combined with other web application testing techniques for a more comprehensive coverage.

You've just learned a few new recon techniques... Right now, it's time to put your skills to the test! Browse through our 70+ public bug bounty programs on Intigriti, and who knows, maybe your next bounty will be earned with us!

START HACKING ON INTIGRITI TODAY