Discovering hidden parameters: An advanced guide

BY BLACKBIRD-EU · JUNE 3, 2025

Reconnaissance plays an integral part in bug bounty hunting, with hidden parameter discovery an even more crucial role as they are often left with inadequate validation. Making these types of parameters usually more susceptible to common injection vulnerabilities such as SQLs, XSS, IDORs and even command injections.

In this article, we will cover 5 various ways to detect possible hidden input parameters, including open-source tools to help you automate the entire process at scale.

Let's dive in!

How do hidden parameters arise?

Before we dive into the 5 methods to discover hidden parameters, we must first understand how they arise. As many developers unconsciously rely on obscurity as a defense mechanism. They believe attackers won't find the (body or query) parameter if it isn't documented or used in the web application interface. This is especially common with:

- Debug parameters left from development (such as cache bypass parameters)
- Internal-only or admin functionality parameters (such as tracking parameters)
- Legacy API endpoints and application routes
- Any other undocumented parameters not meant for the end-user but still accessible to anyone

Using the methods documented throughout this article, we will be able to identify any hidden parameters that lack adequate validation and are accessible to the end-user. Let's get to the first discovery method.

1. HTML input fields

The most straightforward way to discover more parameters is by scraping the **'id'** and the **'name'** attributes of every HTML element, including non-form input fields. This is often done by crawling the target and parsing the attribute values. Automated tools like GoSpider and GetAllParams (GAP) can be specifically deployed for this task.

In the following simplified example below, we can retrieve back 2 parameters; **'password'** and **'user_id.'** One is from a visible input field, while the other was used to reflect the user ID.

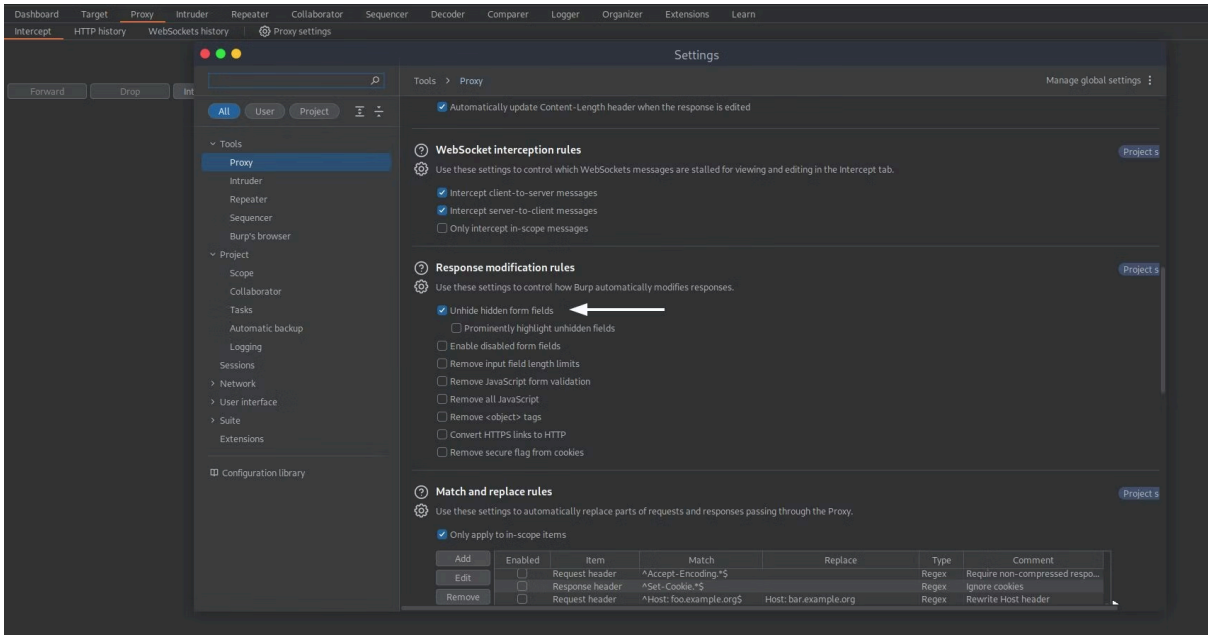
```
<body>
    <div class="container">
        <h1>Danger Zone</h1>
        <p>This section allows you to make non-reversable changes to your account.
           Proceed with caution.
        </p>
        <div class="form-section danger">
            <h3>Delete My Account</h3>
            <p>Delete my account and all the data that is associated with it.</p>
            <form id="auth" action="/api/account/delete" method="POST">
                <div>
                    <label>Confirm password:</label><br>
                    <input type="password" id="user_password" name="user_password" />
                </div>
                <input type="hidden" id="user_id" value="1337" />
            </form>
        </div>

...
```

Finding hidden parameters in HTML source code

Highlighting hidden form input fields in Burp Suite

Proxy interceptors like Burp Suite and ZAProxy allow you to highlight hidden input fields. Since HTTP is stateless, developers often use hidden input fields to help preserve data between page loads in web applications. Basically, the ideal place to inject any of our payloads for SQLi, XSS or any other payload string. Just like we've seen in the previous example.



Highlighting hidden parameters in Burp Suite

2. JavaScript file enumeration

We all know that JavaScript files are a goldmine for bug bounty hunters as they contain several references to API endpoints, app routes and of course input parameters. Examining JavaScript files and

looking for specific function calls that parse and retrieve values of parameters is key here.

In the simplified example below, we can spot several body parameters, including the **'type'** query parameter.

```
...

async function uploadFile(file, options) {
    // Create FormData with file and body parameters
    const form = new FormData();
    form.append('file', file);
    form.append('file_name', file.name);
    form.append('file_size', file.size);
    form.append('file_type', file.type);
    form.append('upload_timestamp', Date.now());

    try {
        const response = await fetch(`https://api.example.com/api/uploads/handler?type=asset`, {
            method: 'POST',
            body: form,
            headers: {
                'X-Requested-With': 'XMLHttpRequest',
            },
        });

        if (!response.ok) {
            throw new Error(`Upload failed: ${response.status}`);
        };

        return await response.json();
    } catch (error) {
        console.error('Upload error:', error);
        throw error;
    };
};

...
```

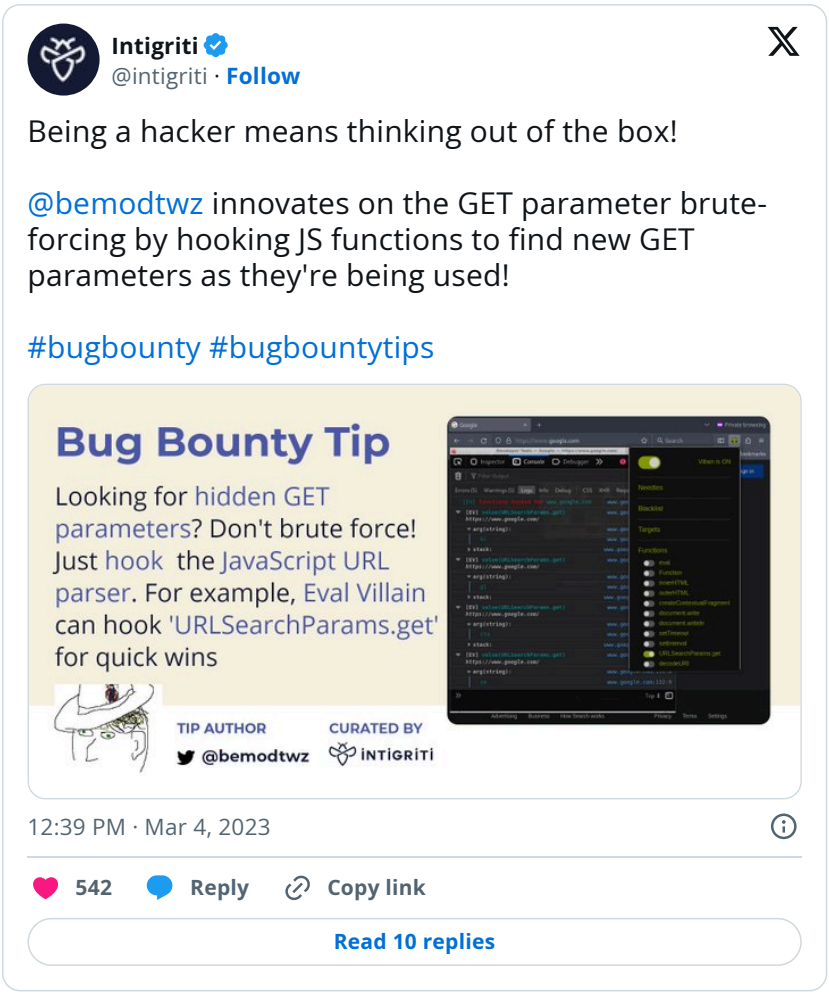
Finding hidden parameters in JavaScript code

Variable names

Besides only looking at potential parameters in pre-defined HTTP requests or parsing functions, you can also try and collect a list of variable names and request them as body or query parameters! Huge thanks to @hakluke for sharing this awesome tip with us!

Intercepting client-side parameters

Another way to detect new parameters is to intercept them on the client-side. Tools like Eval Villain allow you to intercept and get notified if any of the supplied input parameters are processed by the DOM. This behaviour often indicates a possible DOM-based vulnerability, such as DOM-based cross-site scripting (XSS).

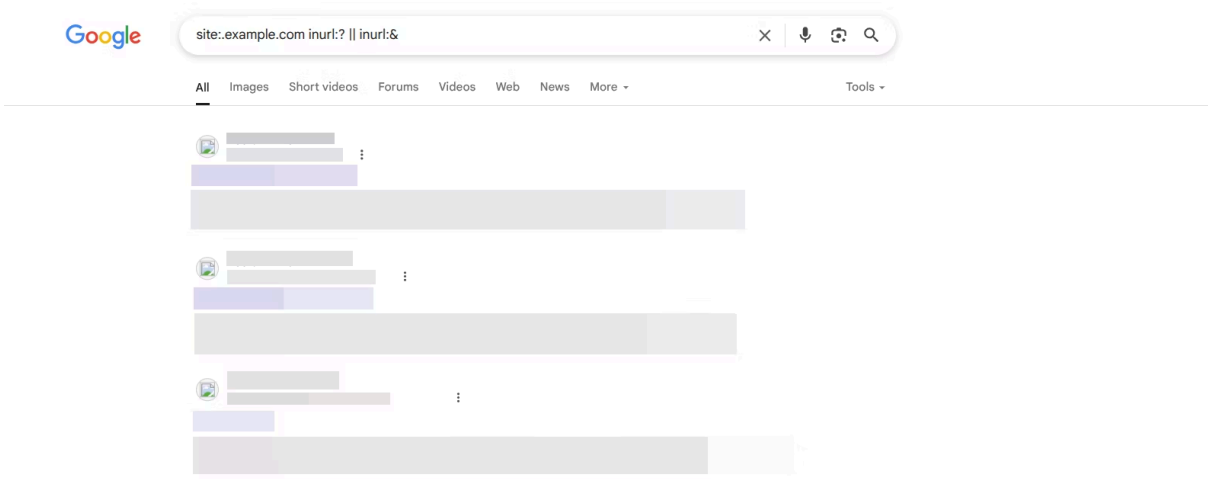


3. Google/GitHub/Wayback Machine enumeration

Google dorking is another method to perform reconnaissance and find more interesting URLs, including parameters linked to your target. Even though this method is well-known, automating it proves to be difficult, making most researchers skip searching for undiscovered content via search engines.

A simple search could yield you 10s of new potential parameters that you can try out when testing:

```
site:.example.com inurl:? || inurl:&
```



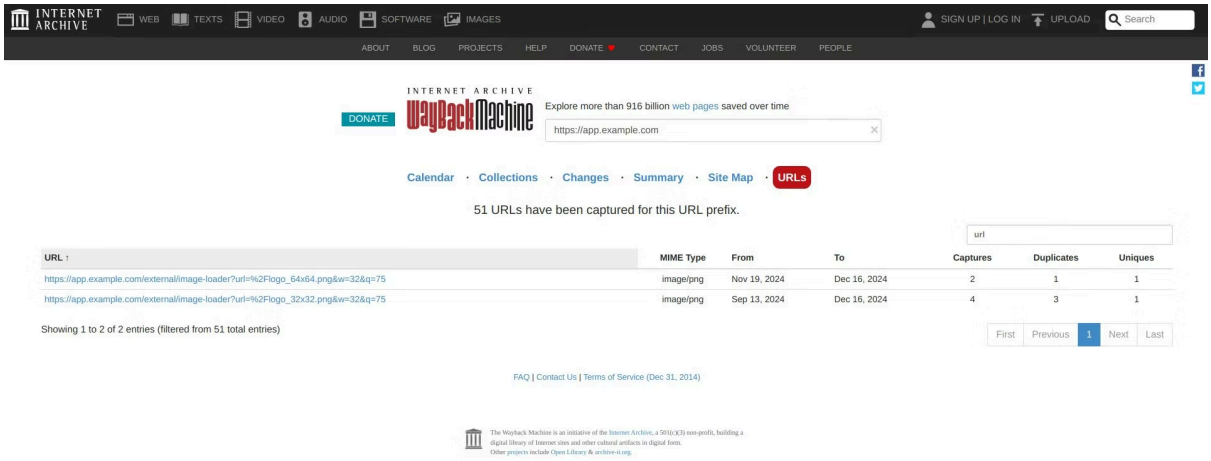
Finding hidden parameters using Google search (dorking)

You can apply the same for any other major search engine or platform, including Bing, DuckDuckGo, Yandex and even GitHub!

Getting more advanced using the Internet Archive

Internet Archive (also referred to as Wayback Machine) can help you discover even more parameters as it indexes multiple versions of almost all web pages on the internet. Making it a great tool to help you find, for instance, more indexed URLs with (query) parameters.

To find parameters using the WaybackMachine, head over to web.archive.org and search for your target. Afterward, filter for indexed URLs that contain query parameters.



Example of leveraging the Internet Archive (Wayback Machine) for finding hidden parameters

TIP! Found an interesting JavaScript file? Try to view an indexed version on the Wayback Machine to potentially find even more (legacy) parameters! Dive deeper into advanced reconnaissance techniques!

4. Parameter fuzzing

Parameter fuzzing is the most accurate and scalable way to discover hidden and unreferenced parameters. By observing response changes possibly induced by a parameter, you can accurately get an overview if a certain parameter is processed by the backend.

Luckily for us, there are also multiple tools available to automate this task, such as Ffuf, Arjun, x8, ParamMiner, etc. The main difficulty originates from the wordlist that you're using for bruteforcing.

Arjun is an open-source, Python-based parameter discovery tool

Generally, custom wordlists created by taking in several application-dependent factors such as naming conventions, localization, programming language or framework, tend to perform better than generic parameter wordlists.

However, a combination of both, a custom and a generic wordlist containing the most used parameter names, is more recommended and can provide more accurate results.

TIP! Custom wordlists can be deployed for any type of content discovery, including the enumeration of unreferenced endpoints and application routes! Read our in-depth article to learn more about how you can craft your own targeted wordlists for your specific targets!

5. Re-use of parameters

In some instances, query and body parameters are re-used and accepted throughout multiple application routes and API endpoints. Both ZAProxy and Burp Suite provide you with a crawl history that you can easily export and parse. Trying some of the parsed parameters on different endpoints is always worth trying out.

TIP! If your target has a mobile application, make sure to examine the source code as well and look for URLs or other links pointing to the web application that include parameters!

Automated tools

Automated tooling can severely help speed up your repetitive scanning tasks, including the discovery of hidden parameters. Below are several free, open-source tools listed (in no particular order) that can help you passively and actively enumerate any unreferenced input parameters.

Arjun

Arjun is a popular open-source, Python-based parameter discovery tool. It provides support for multiple content types and HTTP request methods, ideal for discovering parameters in several contexts. Arjun is available on GitHub:

https://github.com/s0md3v/Arjun

x8

x8 is a blazing-fast, Rust-powered hidden parameter discovery suite. With its extensive configuration support and the ability to use custom wordlists, it makes it easy to perform any type of parameter discovery scan. x8 is available on Github:

https://github.com/sh1yo/x8

ParamMiner

ParamMiner is a simple-to-use plugin to actively find unreferenced parameters within Burp Suite. It also comes with several extensive wordlists to help you quickly start new bruteforcing scans. ParamMiner is available on the BApp Store and on GitHub:

https://github.com/portswigger/param-miner

GetAllParams (GAP)

GetAllParams (GAP) is an alternative Burp Suite extension capable of actively and passively detecting new parameters. The extension can also be used to generate custom wordlists, making this an indispensable tool. GAP is available on GitHub:

https://github.com/xnl-h4ck3r/GAP-Burp-Extension

Conclusion

Reconnaissance is crucial whenever you approach any type of target. Identifying hidden input parameters is even more crucial and can provide you with an edge, specifically since these are often forgotten (or purposefully hidden) and lack adequate validation.

So, you've just learned how to discover more hidden parameters to find more vulnerabilities... Right now, it's time to put your skills to the test! You can start by practicing on vulnerable labs or... browse through our 70+ public bug bounty programs on Intigriti and who knows, maybe earn a bounty on your next submission!

START HACKING ON INTIGRITI TODAY