Creating custom wordlists for bug bounty targets: A complete guide

BY BLACKBIRD-EU · JANUARY 31, 2025 · LAST UPDATED ON MARCH 16, 2026

Everyone understands the importance of custom wordlists in bug bounties, and how they can be deployed in targeted bruteforcing attacks to help discover new hidden endpoints. Custom wordlists can also help reduce the number of requests sent and even prevent unnecessary aggressive scanning of bug bounty targets.

In this article, you will learn how to craft custom wordlists that you can deploy on your target to find more vulnerabilities through unreferenced directories, files and even parameters.

Let's dive in!

Importance of custom wordlists

Generic wordlists often miss targeted keywords, company-specific terms, and product names and never take advantage of your target's naming convention or patterns. For this reason, we need to craft and employ custom wordlists to help discover the hidden endpoints and parameters that we'd likely miss otherwise by solely using generic wordlists.

A custom wordlist isn't only about collecting random targeted keywords, it also consists of the following 3 main lists:

- Company-specific keywords
- Technology-specific keywords
- Generic and commonly occurring keywords

Let's take a look at all 3 elements to better understand what distinguishes a generic wordlist from a good custom wordlist that can help you find much more hidden content.

1. Company-specific keywords

Company-specific terms are keywords used within the app to define endpoint names, parameters, and features.

If your target is built around a certain feature (such as an in-app messaging feature), it will have multiple references to it, including defined API endpoints, application routes and parameters.

We must make sure to include them in our wordlists to help us discover other potentially hidden functionalities and features.

2. Technology-specific keywords

Technology-specific keywords are based on the tech stack and frameworks that a target is using. These keywords are crucial as they can help you identify files and application routes on the server that are specific to a certain technology. PHP Larvel commonly has accessible paths and endpoints that are specific to this PHP framework, and we want to make sure we include these terms as well.

If you came across a WordPress instance, you'd likely want to look for vulnerable WordPress plugins or other commonly accessible application routes, such as the admin login page.

3. Generic and commonly occurring keywords

Finally, a good custom wordlist also consists of commonly occurring keywords. These are often keywords that appear across many web applications, regardless of the company or technology stack. Think of the `/api` endpoint or the `/assets` directory, for example.

Now that we've learned what makes a custom wordlist useful. Let's now dive into how we can generate our unique custom list for our specific bug bounty target!

Crafting custom wordlists

Step 1: Generating a company-specific wordlist

Extracting in-page keywords

Let's tackle the first list with company-specific keywords first. Crawling the target and tokenizing in-page keywords and URLs is a great starting point. Luckily for us, we can automate this entire part and make use of CeWL, an open-source tool that crawls your target and outputs commonly found words.

Here's a simple example of using CeWL for crawling our target. Optionally, we've set the depth to **5** and the default character length of any keyword to a minimum of **4 characters**:

```
$ cewl https://example.com --header "Cookie: PHPSESSID=7a9b4c2d8e3f1g5h6i7j8k9l0m1n2o3p" -d 5 -m 4
```

```
$ cewl https://example.com --header "Cookie: PHPSESSID=7a9b4c2d8e3f1g5h6i7j8k9l0m1n2o3p" -d 5 -m 4
CeWL 5.5.2 (Grouping) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
dashboard
users
profile
settings
admin
management
reports
analytics
metrics
config
integration
database
status
backup
monitoring
logs
security
access
```

Use CeWL to generate custom wordlists based on in-page text

> **Tip!**
>
> Make sure to set authentication headers when using CeWL, that way it will also reach authenticated parts of your target to help you yield more accurate results.

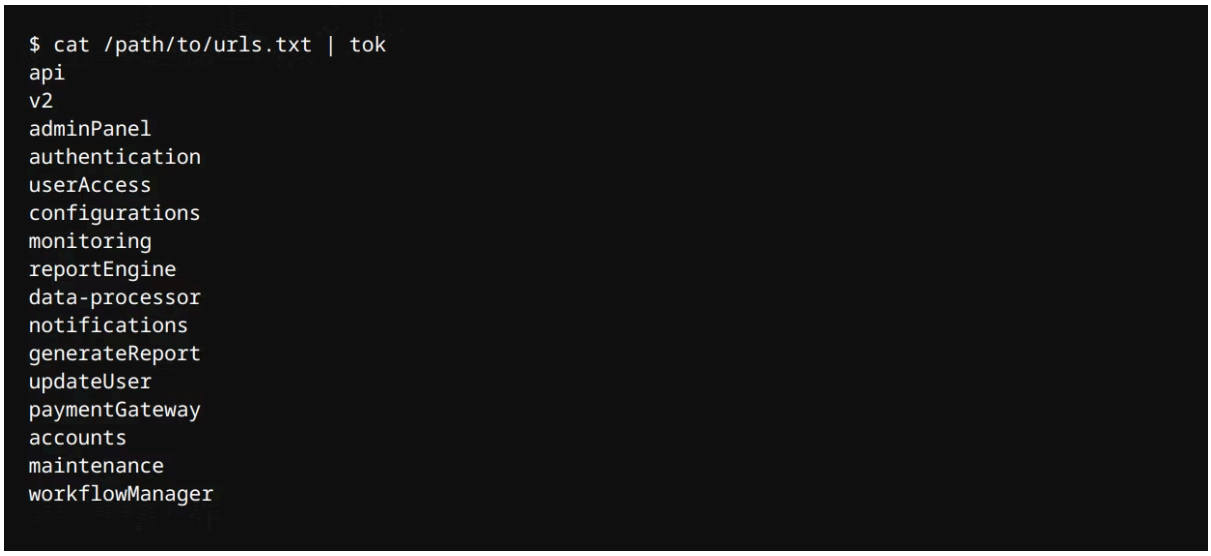
Extracting URL keywords

Another way to build a list of target-specific keywords is by tokenizing URLs. Using this method, we can transform our list of URLs gathered from our proxy intercepting tool into keywords that we can use for content discovery.

Thanks to Tok by @TomNomNom, an open-source tool that auto-tokenizes URLs, we can effortlessly provide it our list of captured URLs and generate a list of possible keywords.

Here's a quick usage example of Tok:

```
$ cat /path/to/urls.txt | tok
```



Tokenize captured URLs using Tok

Extracting keywords from JavaScript files

We can now even go a step further and use a tool like Getjswords to generate even more custom keywords derived from JavaScript files.

Getjswords is a simple open-source Python tool that takes in a list of JavaScript URLs, fetches them and returns a list of potential keywords. These generated keywords can again help us discover hidden and unreferenced files, directories and application or API routes but also help us find parameters!

Here's a quick usage guide on using getjswords:

```
$ cat /path/to/js-urls.txt | python3 getjswords.py
```

```
$ cat /path/to/js-urls.txt | python3 getjswords.py
filter
searchQuery
sessionId
apiKey
requestHeaders
avatarUrl
accessToken
initialize
logout
display
userId
configParams
fetchData
analyticsId
validateInput
returnUrl
```

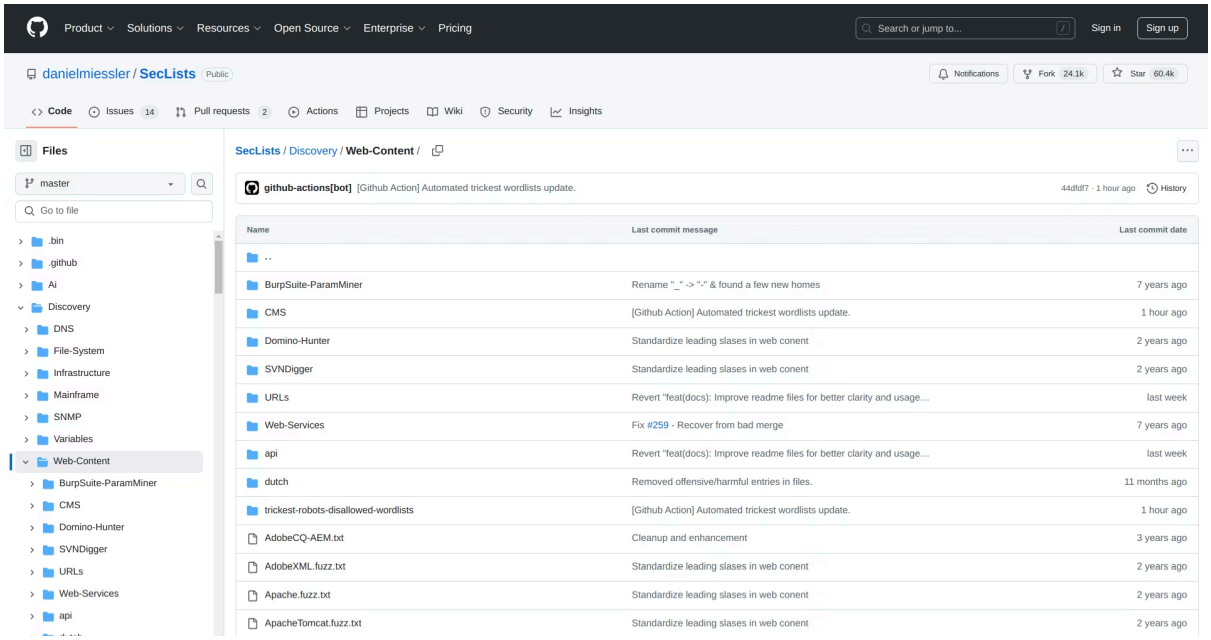
Your js-urls.txt file is your file with a list of URLs pointing to JavaScript files.

Step 2: Crafting a technology-specific wordlist

Technology-specific wordlists are helpful to add coverage for the underlying technology stack that your target consists of. For this reason, we must fingerprint all technologies used by our target and use public wordlists or curate our own.

To fingerprint your target, you may use tools like BuiltWith or Wappalyzer or simply look at response elements such as HTTP response headers.

Once you've fingerprinted all used technologies, [browse through SecList's wordlist files] and select the wordlists that match your fingerprinted technologies and services.



SecLists

Step 3: Including commonly occurring keywords

The last step in generating our custom wordlist is adding commonly occurring keywords and terminologies that are not bound to a specific target. A few quick examples:

- Assets folders (**assets** , **storage** , **files** , ...)
- API endpoints (**api** , **v1** , **graphql** , ...)
- Authentication-related paths (**login** , **signin** , **oauth** , **sso** , ...)
- Admin and other authentication panels (**admin** , **dashboard** , **console** , ...)
- Profile and account components (**profile** , **settings** , **account** , ...)
- Development paths (**dev** , **staging** , **test** , **profiler** , **debug** , ...)
- Common parameters (**id** , **userId** , **page** , ...)

JHaddix's all.txt is a great starting point, it's a general-purpose wordlist and a massive collection of these common web application terms that he's gathered over the years.

Step 4: Combining all lists

The latest step consists of combining all the lists that you've generated so far together and using them for content discovery. With this list, you can now perform:

- Hostname bruteforcing (VHost scanning)
- Content discovery to find hidden and unreferenced directories and files
- Parameter bruteforcing to find processed input parameters

> **Tip!**
>
> Want to get more advanced? Use a tool like wl to convert your wordlist to match the naming convention of your target! Most developers use a consistent naming pattern when defining endpoints, application routes or parameter names. This method can help you generate even more accurate word lists!

Example usage of wl where we transform camelCase keywords to snake_case keywords to match our targets' naming convention.

Conclusion

We've just crafted a custom wordlist that we can use and deploy on our target. The targeted keywords will ensure we find the most amount of hidden directories and files possible. We can also use this wordlist to help enumerate unreferenced input parameters or hosts that have likely never been tested before. These untouched assets are often more susceptible to web vulnerabilities as they may not have received the same security attention as the main application.

You've just learned how to craft custom wordlists that can significantly increase your chances of coming across untouched assets and attack surfaces... 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