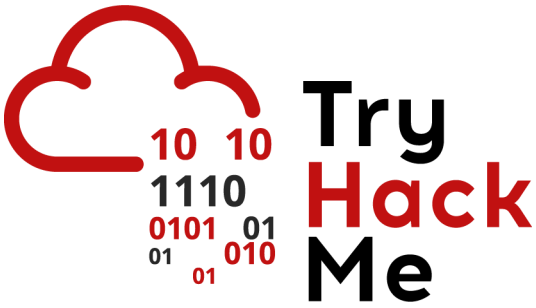



# Brute Forcing Directories

It's common practice to use off the shelf applications. This can be anything from default servers like apache to using common frameworks like Django (built using Python), and Express (built using Javascript on NodeJS).

Unfortunately, many off the shelf components are not default by secure; many use default credentials and even leave sensitive pages open to the public including:

- Admin panels
- Server status checks
- Debug pages

These pages contain information and functionality that at least allow an attacker to learn more about a system; paths to sensitive files on the server and version information. At most, these pages have live user data and actual functionality that should not be left open; querying and manipulating users' data and much more.

*What would an attacker do:* Some web applications make it easier for attackers to find these sensitive pages. The most common way is enumerating directories by brute-forcing. This involves sending requests to different pages on the server and using the server's response to verify the existence of a page (and even access the page). Here's a short example:

As an attacker, I would use a list of common pages that are known to be exposed(more on this later) to access the pages. For example, I know that the /css page exists because the server responds with a 200 status code which indicates that the server retrieved the page successfully. Suppose I try to access 2 pages (/random-page and /server-status). Accessing the /random-page returns a response status code of 404 (this status code means the page is not found). However, accessing the /server-status page returns a response status code of 200. Accessing /server-status returns the same status code as a normal page so I know that it is accessible. On the other hand, accessing /random-page gives a different response code from accessing a normal page so we know it doesn't exist.

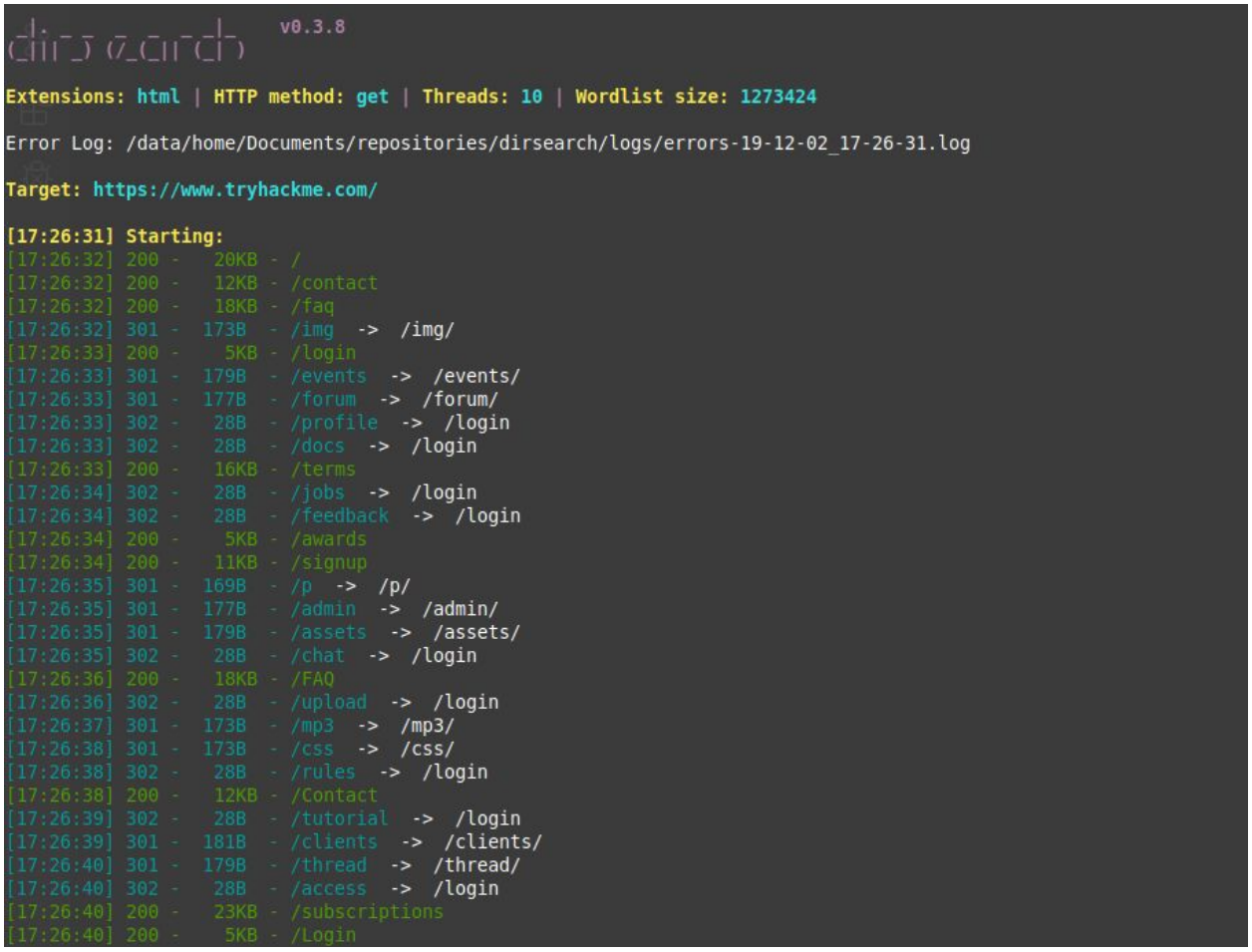
Luckily, we can do this using an automated tool instead of sending a request manually. There are a lot of different tools that do this, but today we'll use DirSearch. To use this tool, you need Python installed on your system!

Once you have this tool installed, you need to use a wordlist to look for common pages/directories. Again, there can be a lot of different word lists you can use, but for this, we will use the **directory-list-2.3-medium.txt** available to download from here.

An example of running this tool shows:
./dirsearch.py -u https://www.tryhackme.com/ -w ./DirBuster-Lists/directory-list-2.3-big.txt -e html

Syntax:

- -u is the hostname of the website
- -w is the wordlist
- -e is the extension:
    - Different web pages use different technologies(you can usually identify this by the file it loads in the browser e.g. if it's a .js, .aspx page)
- -f is the flag used to force extensions applied to the pages in the word list:
    - Mostly used when you're quite sure about what kind of technology a server is running
    - If you don't know what extension to brute force, you don't need to specify this flag




*Sample Output from Dirsearch*

As you can see, some pages have 200 response codes but others have 301. In this case, the 301 response code means that the page is redirecting to the login page. Like mentioned earlier, different applications respond differently. In this case, we've learned that the pages with 301 response codes can only be accessed after login.

Sensitive/Default Information

Depending on the application used, an attacker can easily find sensitive information when accessing the application. Example include:

- Comments and API keys in the source code
- Password (Hashes) in requests and responses:
    - For responses, having sensitive information as part of GET requests is insecure as these requests are usually logged for debugging. This would mean anyone with access to the logs has information about it
    - For responses, sensitive information is usually put in headers, cookies or source code

*What would an attacker do:*

After checking the aforementioned locations, an attacker can mostly use this extra information to enumerate the application. Examples of enumeration include:

- Cracking password hashes in the response to access users' accounts
- Using API keys to access functions and API calls without authorization
- Find hidden URLs and System information that they can use to find public exploits

*Are these realistic?:* Yes! Companies either use default credentials or have exposed web pages up.