

Information Disclosure

: 24/08/2022

📅 Aug 23, 2022

🕒 13 min read

📁 [Info-Disclosure Web-Notes](#)

These bugs are common; in fact, they're the type of bug I find most often while bug bounty hunting, even when I'm searching for other bug types.

References

[Sensitive Information Disclosure](#)

Mechanisms

Information disclosure occurs when an application fails to properly **protect sensitive information**, giving users access to information they shouldn't have available to them.

This sensitive information can include technical details that aid an attack, like **software version numbers, internal IP addresses, sensitive filenames, filepaths, user's age, bank account numbers, email addresses, and mailing addresses**, to unauthorized third parties.

For example, learning the exact **software versions** an application uses will allow attackers to look for publicly disclosed vulnerabilities that affect the application. Configuration files often contain information such as **access tokens and internal IP addresses** that attackers can use to further compromise the organization.

For example, the `X-Powered-By` header, which is used by many applications, shows you which framework the application runs: `X-Powered-By: PHP/5.2.17`

Finally, applications often leak sensitive information by including it in their **public code**. Developers might accidentally place information such as credentials, internal IP addresses, informative code comments, and users' private information in **public source code** such as the HTML and JavaScript files that get served to users.

Hunting for Information Disclosure

You can use several strategies to find information disclosure vulnerabilities, depending on the application you're targeting and what you're looking for.

Step 1: Attempt a Path Traversal Attack

Path traversal attacks are used to access files outside the web application's root folder.

For example, let's say a website allows you to load an image in the application's image folder by using a relative URL.

This URL, for example, will redirect users to <https://example.com/images/1.png>:

<https://example.com/image?url=/images/1.png>

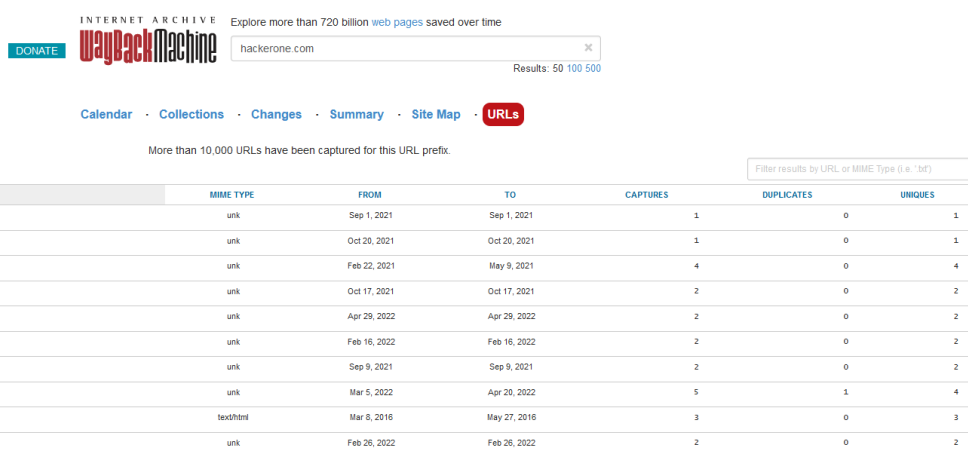
In this case, the url parameter contains a **relative URL** (/images/1.png) that references files within the web application root. You can insert the `../` sequence to try to navigate out of the images folder and out of the web root.

It might take some trial and error to determine how many `../` sequences you need to reach the system's **root directory**. Also, if the application implements some sort of input validation and doesn't allow `../` in the filepath, you can use encoded variations of `../`, such as `%2e%2e%2f` (URL encoding), `%252e%252e%255f` (double URL encoding), and `..%2f` (partial URL encoding).

Step 2: Search the Wayback Machine

On the Wayback Machine's site, simply search for a domain to see its past versions. To search for a domain's files, visit https://web.archive.org/web/*/DOMAIN.

For example, <https://web.archive.org/web//example.com/> will return a list of URLs related to example.com.



The screenshot shows the Wayback Machine interface. At the top, it says "INTERNET ARCHIVE" and "Explore more than 720 billion web pages saved over time". There is a search bar with "hackerone.com" entered and a "DONATE" button. Below the search bar, it says "Results: 50 100 500". There are links for "Calendar", "Collections", "Changes", "Summary", "Site Map", and "URLs". Below these links, it says "More than 10,000 URLs have been captured for this URL prefix." There is a table with columns: "URL", "MIME TYPE", "FROM", "TO", "CAPTURES", "DUPLICATES", and "UNIQUES". The table contains 9 rows of data.

URL	MIME TYPE	FROM	TO	CAPTURES	DUPLICATES	UNIQUES
http://hackerone.com/or1=1--	unk	Sep 1, 2021	Sep 1, 2021	1	0	1
http://hackerone.com/.%2	unk	Oct 20, 2021	Oct 20, 2021	1	0	1
http://hackerone.com/Onlymohammed	unk	Feb 22, 2021	May 9, 2021	4	0	4
http://hackerone.com/0x90cc?type=us	unk	Oct 17, 2021	Oct 17, 2021	2	0	2
http://hackerone.com/0xcyborg	unk	Apr 29, 2022	Apr 29, 2022	2	0	2
http://hackerone.com/3mm3	unk	Feb 16, 2022	Feb 16, 2022	2	0	2
http://hackerone.com/7hamoody1	unk	Sep 9, 2021	Sep 9, 2021	2	0	2
http://hackerone.com/7odamo	unk	Mar 5, 2022	Apr 26, 2022	5	1	4
http://hackerone.com/?ref=producthunt	text/html	Mar 8, 2016	May 27, 2016	3	0	3
http://hackerone.com/abd_4fg	unk	Feb 26, 2022	Feb 26, 2022	2	0	2

You can then use the search function to see whether any sensitive pages have been archived. For example, to look for admin pages, search for the term `/admin` in the found URLs

URL ↑	MIME TYPE	FROM	TO	CAPTURES	DUPLICATES	UNIQUES
https://hackerone.com/admin101010	text/html	Dec 3, 2020	May 13, 2021	2	0	2
https://hackerone.com/admin402	text/html	Jul 31, 2021	Jul 31, 2021	1	0	1
https://hackerone.com/admin420420	text/html	Aug 9, 2020	May 18, 2021	3	0	3
https://hackerone.com/admin616	text/html	Aug 6, 2020	May 12, 2021	3	0	3
https://hackerone.com/admin6666	text/html	May 19, 2021	May 19, 2021	1	0	1
https://hackerone.com/admin_op	text/html	Aug 3, 2020	May 16, 2021	3	0	3
https://hackerone.com/admin_301	text/html	Aug 9, 2020	Jun 15, 2021	4	1	3
https://hackerone.com/admin_bvwn	text/html	May 6, 2021	May 6, 2021	1	0	1
https://hackerone.com/admin_hackerone	text/html	Jan 17, 2021	May 8, 2021	2	0	2
https://hackerone.com/adminadminadmin	text/html	Oct 28, 2020	Apr 21, 2021	2	0	2
https://hackerone.com/adminbesilli	text/html	Jul 29, 2021	Jul 29, 2021	1	0	1

You can also search for backup files and configuration files by using common file extensions like `.conf` and `.env`, or look for source code, like JavaScript or PHP files, by using the file extensions `.js` and `.php`.

Download interesting archived pages and look for any sensitive info. For example, are there any hardcoded credentials that are still in use, or does the page leak any hidden endpoints that normal users shouldn't know about?

Step 3: Search Paste Dump Sites

Next, look into paste dump sites like **Pastebin** and **GitHub** gists. But on a site like Pastebin, for example, shared text files are public by default. If developers upload a **sensitive file**, everyone will be able to read it. For this reason, these code-sharing sites are pretty infamous for leaking credentials like **API keys and passwords**.

Tools like PasteHunter or pastebin-scraper can also automate the process. Pastebin-scraper <https://github.com/streaak/pastebin-scraper/> uses the Pastebin API to help you search for paste files. This tool is a shell script, so download it to a local directory and run the following command to search for public paste files associated with a particular keyword. The `-g` option indicates a general keyword search:

```
./scrape.sh -g KEYWORD
```

Step 4: Reconstruct Source Code from an Exposed .git Directory

When a developer uses Git to version-control a project's source code, Git will store all of the project's version-control information, including the **commit history of project files**, in a `.git` directory. Normally, this `.git` folder shouldn't be accessible to the public, but sometimes it's accidentally made available. This is when information leaks happen. When a `.git` directory is exposed, attackers can obtain an application's source code and therefore gain access to **developer comments**, hardcoded **API keys**, and other sensitive data via secret scanning tools like truffleHog <https://github.com/trufflesecurity/trufflehog> or Gitleaks <https://github.com/zricethezav/gitleaks>.

Checking Whether a .git Folder Is Public

To check whether an application's `.git` folder is public, simply go to the application's root directory (for example, example.com) and add `/.git` to the URL: <https://example.com/.git>

Three things could happen when you browse to the `/.git` directory. If you get a `404 error`, this means the application's `.git` directory isn't made available to the public, and you won't be able to leak information

this way. If you get a 403 error, the .git directory is available on the server, but you won't be able to directly access the folder's root, and therefore won't be able to list all the files contained in the directory. If you don't get an error and the server responds with the directory listing of the .git directory, you can directly browse the folder's contents and retrieve any information contained in it.

Downloading Files

The wget command retrieves content from web servers. You can use wget in recursive mode (-r) to mass-download all files stored within the specified directory and its subdirectories: `$ wget -r [example.com/.git] (http://example.com/.git)`

You can do this by trying to access the directory's config file: `$ curl [https://example.com/.git/config] (https://example.com/.git/config)`

A .git directory is laid out in a specific way. When you execute the following command in a Git repository, you should see contents resembling the following:

```
$ ls .git
COMMIT_EDITMSG HEAD branches config description hooks index info logs
objects refs
```

the following command will return a list of folders:

```
$ ls .git/objects
00 0a 14 5a 64 6e 82 8c 96 a0 aa b4 be c8 d2 dc e6 f0 fa info pack
```

And this command will reveal the Git objects stored in a particular folder:

```
$ ls .git/objects/0a
082f2656a655c8b0a87956c7bcd93dfda23f8
4a1ee2f3a3d406411a72e1bea63507560092bd 66452433322af3d3
19a377415a890c70bbd263 8c20ea4482c6d2b0c9cdaf73d4b05c2c8c44e9
ee44c60c73c5a622bb1733338d3fa964
b333f0
0ec99d617a7b78c5466daa1e6317cbd8ee07cc
52113e4f248648117bc4511da04dd4634e6753
72e6850ef963c6aeee4121d38cf9de773865d8
```

You can determine an object's type by using this command:

```
$ git cat-file -t OBJECT-HASH
```

You can display the file associated with a Git object by using the following command:

```
$ git cat-file -p OBJECT-HASH
```

The `/config` file is the Git configuration file for the project, and the `/HEAD` file contains a reference to the current branch:

```
$ cat .git/HEAD
ref: refs/heads/master
```

But how do you find out which files on the server are available when object files have complex paths, such as `.git/objects/0a/72e6850ef963c6aeee4121d38cf9de773865d8`? You start with filepaths that you already know exist, like `.git/HEAD`! Reading this file will give you a reference to the current branch (for example, `.git/refs/heads/master`) that you can use to find more files on the system:

```
$ cat .git/HEAD
ref: refs/heads/master
$ cat .git/refs/heads/master
0a66452433322af3d319a377415a890c70bbd263
$ git cat-file -t 0a66452433322af3d319a377415a890c70bbd263
commit
$ git cat-file -p 0a66452433322af3d319a377415a890c70bbd263
tree 0a72e6850ef963c6aeee4121d38cf9de773865d8
```

Now examine that tree object:

```
$ git cat-file -p 0a72e6850ef963c6aeee4121d38cf9de773865d8
100644 blob 6ad5fb6b9a351a77c396b5f1163cc3b0abcde895 .gitignore
040000 blob 4b66088945aab8b967da07ddd8d3cf8c47a3f53c source.py
040000 blob 9a3227dca45b3977423bb1296bbc312316c2aa0d README
040000 tree 3b1127d12ee43977423bb1296b8900a316c2ee32 resources
```

Bingo! You discover some source code files and additional object trees to explore.

On a remote server, your requests to discover the different files would look a little different. For instance, you can use this URL to determine the HEAD: [<https://example.com/.git/HEAD>]

(<https://example.com/.git/HEAD>) Use this URL to find the object stored in that HEAD:

[<https://example.com/.git/refs/heads/master>]

(<https://example.com/.git/refs/heads/master>) Use this URL to access the tree associated with the commit:

[<https://example.com/.git/objects/0a/72e6850ef963c6aeee4121d38cf9de773865d8>]

(<https://example.com/.git/objects/0a/72e6850ef963c6aeee4121d38cf9de773865d8>)

Finally, use this URL to download the source code stored in the `source.py` file:

<https://example.com/.git/objects/4b/66088945aab8b967da07ddd8d3cf8c47a3f53c>

After recovering the project's source code, you can grep for sensitive data such as `hardcoded credentials`, `encryption keys`, and `developer comments`. If you have time, you can browse through the entire recovered codebase to conduct a source code review and find potential vulnerabilities.

Step 5: Find Information in Public Files

View page source. You should see the HTML source code of the current page. Follow the links on this page to find other HTML files and JavaScript files the application is using. Then, on the HTML file and the JavaScript files found, grep every page for hardcoded credentials, API keys, and personal information with keywords like password and api_key. You can also locate JavaScript files on a site by using tools like LinkFinder : <https://github.com/GerbenJavado/LinkFinder/>

Finding Your First Information Disclosure!

Now that you understand the common types of information leaks and how to find them, follow the steps discussed in this chapter to find your first information disclosure:

1. Look for software version numbers and configuration information by using the recon techniques presented in Chapter 5.
 2. Start searching for exposed configuration files, database files, and other sensitive files uploaded to the production server that aren't protected properly. Techniques you can use include path traversal, scraping the Wayback Machine or paste dump sites, and looking for files in exposed .git directories.
 3. Find information in the application's public files, such as its HTML and JavaScript source code, by grepping the file with keywords.
 4. Consider the impact of the information you find before reporting it, and explore ways to escalate its impact.
 5. Draft your first information disclosure report and send it over to the bug bounty program!
-

Prevention

- Make sure that everyone involved in producing the website is fully aware of what information is considered sensitive. Sometimes seemingly harmless information can be much more useful to an attacker than people realize. Highlighting these dangers can help make sure that sensitive information is handled more securely in general by your organization.
 - Audit any code for potential information disclosure as part of your QA or build processes. It should be relatively easy to automate some of the associated tasks, such as stripping developer comments.
 - Use generic error messages as much as possible. Don't provide attackers with clues about application behavior unnecessarily.
 - Double-check that any debugging or diagnostic features are disabled in the production environment.
 - Make sure you fully understand the configuration settings, and security implications, of any third-party technology that you implement.
 - Take the time to investigate and disable any features and settings that you don't actually need.
-

PortSwigger

How to test for information disclosure vulnerabilities

Fuzzing

Even if the content of an **error message** doesn't disclose anything, sometimes the fact that one error case was encountered instead of another one is useful information in itself.

- Add payload positions to parameters and use pre-built wordlists of fuzz strings to test a high volume of different inputs in quick succession.
- Easily identify differences in responses by comparing HTTP status codes, response times, lengths, and so on.
- Use grep matching rules to quickly identify occurrences of keywords, such as `error`, `invalid`, `SELECT`, `SQL`, and so on.
- Apply grep extraction rules to extract and compare the content of interesting items within responses.

Common sources of information disclosure

- **Files for web crawlers**

Many websites provide files at `/robots.txt` and `/sitemap.xml` to help crawlers navigate their site. Among other things, these files often list specific directories that the crawlers should skip, for example, because they may contain sensitive information.

- **Directory listings**

Web servers can be configured to automatically list the contents of directories that do not have an index page present. This can aid an attacker by enabling them to quickly identify the resources at a given path, and proceed directly to analyzing and attacking those resources. It particularly increases the exposure of sensitive files within the directory that are not intended to be accessible to users, such as temporary files and crash dumps.

- **Developer comments**

During development, in-line HTML comments are sometimes added to the markup. These comments are typically stripped before changes are deployed to the production environment. However, comments can sometimes be forgotten, missed, or even left in deliberately because someone wasn't fully aware of the security implications. Although these comments are not visible on the rendered page, they can easily be accessed using Burp, or even the browser's built-in developer tools.

- **Error messages**

Verbose error messages can also provide information about different technologies being used by the website. For example, they might explicitly name a template engine, database type, or server that the website is using, along with its version number. This information can be useful because you can easily search for any documented exploits that may exist for this version. Similarly, you can check whether there are any common configuration errors or dangerous default settings that you may be able to exploit. Some of these may be highlighted in the official documentation.

Lab: Information disclosure in error messages [Web Security Academy](#)

- **Debugging data**

For debugging purposes, many websites generate custom error messages and logs that contain large amounts of information about the application's behavior. While this information is useful during development, it is also extremely useful to an attacker if it is leaked in the production environment.

Debug messages can sometimes contain vital information for developing an attack, including:

- ⇒ Values for key session variables that can be manipulated via user input
- ⇒ Hostnames and credentials for back-end components
- ⇒ File and directory names on the server
- ⇒ Keys used to encrypt data transmitted via the client

Lab: Information disclosure on debug page [Web Security Academy](#)

- **User account pages**

By their very nature, a user's profile or account page usually contains sensitive information, such as the user's email address, phone number, API key, and so on. As users normally only have access to their own account page, this does not represent a vulnerability in itself. However, some websites contain `Logic Flows` that potentially allow an attacker to leverage these pages in order to view other users' data.

For example, consider a website that determines which user's account page to load based on a `user` parameter.

```
GET /user/personal-info?user=carlos
```

- **Source code disclosure via backup files**

Typical examples of this include API keys and credentials for accessing back-end components.

If you can identify that a particular open-source technology is being used, this provides easy access to a limited amount of source code.

For example, text editors often generate temporary backup files while the original file is being edited. These temporary files are usually indicated in some way, such as by appending a tilde (~) to the filename or adding a different file extension. Requesting a code file using a backup file extension can sometimes allow you to read the contents of the file in the response.

Lab: Source code disclosure via backup files [Web Security Academy](#)

- **Information disclosure due to insecure configuration**

Websites are sometimes vulnerable as a result of improper configuration. This is especially common due to the widespread use of third-party technologies, whose vast array of configuration options are not necessarily well-understood by those implementing them. In other cases, developers might forget to disable various debugging options in the production environment. For example, the HTTP `TRACE` method is designed for diagnostic purposes. If enabled, the web server will respond to requests that use the `TRACE` method by echoing in the response the exact request that was received. This behavior is often harmless, but occasionally leads to information disclosure, such as the name of internal authentication headers that may be appended to requests by reverse proxies.

[Lab: Authentication bypass via information disclosure Web Security Academy](#)

- Version control history

Virtually all websites are developed using some form of version control system, such as Git. By default, a Git project stores all of its version control data in a folder called `.git`. Occasionally, websites expose this directory in the production environment. In this case, you might be able to access it by simply browsing to `/.git`. While it is often impractical to manually browse the raw file structure and contents, there are various methods for downloading the entire `.git` directory. You can then open it using your local installation of Git to gain access to the website's version control history. This may include logs containing committed changes and other interesting information.