Ryan Schanzenbacher
12/1/23
Vulnerability Technical Details

The vulnerability I am going to write about was actually found by a friend of mine while trying to break the server. It is a version of directory traversal (...again) that I hadn't thought of, but this time I'm going to talk about why it really isn't that terrible. In this case, symlinks are improperly handled for one specific file: `index.html`

Steps to Reproduce:

1. Create a file named `index.html` that is a symlink to a secret file.
2. Launch the server either using root permissions or using the docker container (since this runs as root by default). Make sure you use the `-t` parameter to have the server attempt to load a `index.html` whenever it navigates to a directory.
3. Navigate to the directory that contains the file. When you navigate into it, the symlink will be followed and data can be leaked.

Alternate steps:

1. Run the python script provided to create a file outside the directory and spin up the server, launch firefox and verify that data has been leaked.

Details:

Since the symlink is not resolved, the code assumes that it is within the server's root and is safe to visit, it doesn't assume a malious link within the filesystem. While this can be seen as a feature, it is technically a logic bug since symlinks should all get resolved properly and checked before returning the file. Or at least have an option that allows the user to specify whether symlinks should be followed at all or not. The fix is quite simple, though. All that needs to be changed is in each handler file, where the directory is checked for index.html when attempting to "try_files". In each handler, simply add a check in an `if` statement that uses the `.resolve()` function to resolve the true location of the symbolic link, checking whether it is in the document root, and returning a 403 if it is not. This needs to be changed in each handler individually, though it is only a 3-4 line fix, looking something like the following:

```
p = canonicalized_path / index.html
p.resolve()
if doc_root not in canonicalized_path.parents and doc_root != canonicalized_path:
    return HTTPResponse(HTTPStatusCode.FORBIDDEN, version=version, content="Forbidden")
```

More implications are listed in the risk assessment.