

CSCE 413: Software Security
PoC 1

Theory

Show how to find the OWASP Top 10 vulnerabilities via Google Dorks (Show the links and screenshots).

NOTE: Even though you (the grader) are most likely very well-versed in cybersecurity, I would still like to offer a warning. The websites listed in this document are not only vulnerable to certain exploits, but could be malicious themselves. Please exercise caution when browsing these links.

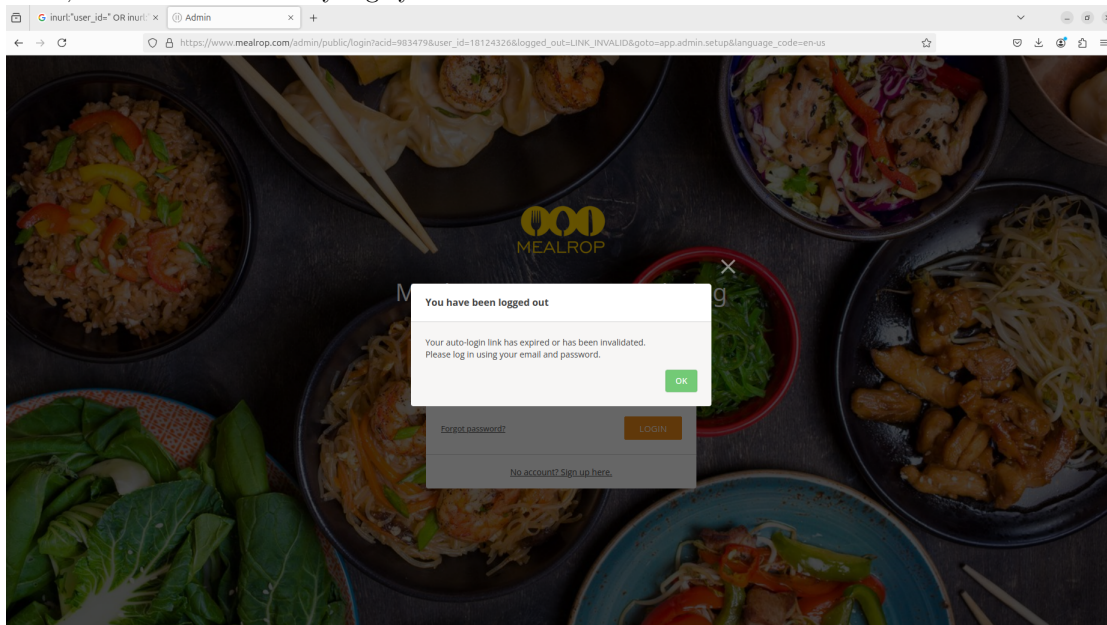
A01:2021-Broken Access Control

Site: Mealrop

I did not have much success in finding explicit cases of broken access control, but I did find one website using the search prompt:

```
inurl:"user_id=" OR inurl:"account_id=" inurl:admin
```

I was aiming to find websites that explicitly let me access links that had admin URLs and/or specific accounts. This attempt focused on logging into specific admin accounts. This resulted in a website called Mealrop that temporarily loads, and then immediately logs you out.



This relates to Broken Access Control as it violates the principle of "deny by default", as access is temporarily granted for a non-admin user.

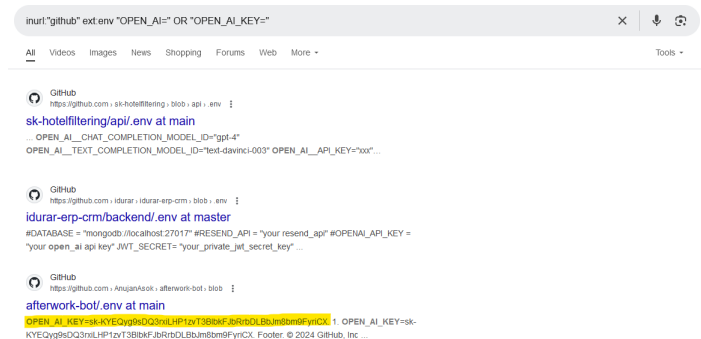
A02:2021-Cryptographic Failures

Site: GitHub

OWASP provides a preventative measure for cryptographic failures, this being “Don’t store sensitive data unnecessarily” (A02 Cryptographic Failures). I’ve noticed that many people fail to remove their keys from .env files, so I targeted .env files on Github searching for Open AI keys. I used the search prompt:

```
inurl:"github" ext:.env "OPEN_AI=" OR "OPEN_AI_KEY="
```

This resulted in one result containing a possible key,



This relates to Cryptographic Failures as this user publicly stored sensitive cryptographic information that allows a malicious user to initiate a passive attack, grabbing keys they shouldn’t have access to.

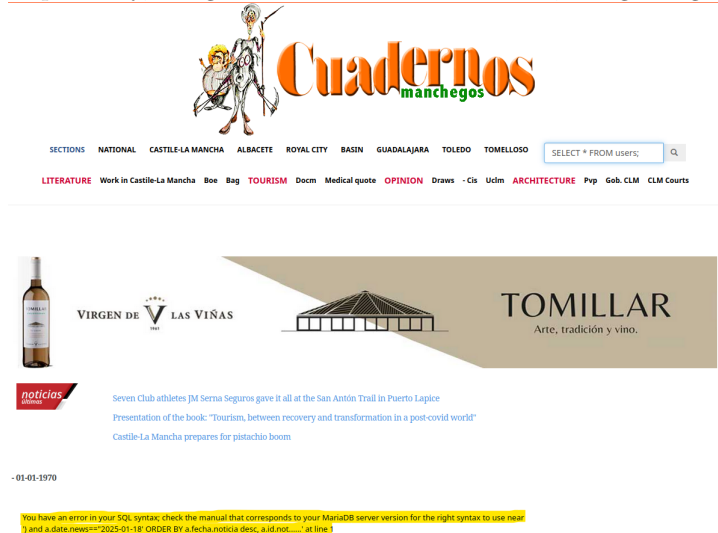
A03:2021-Injection

Sites: DERC, Cuadernos manchegos

While I do not have the skills to test for injection exploits on websites directly, I can use my knowledge of MySQL and MongoDB to attempt to find websites with errors when accessing data from their databases. These errors could then be used to understand the website’s architecture, making it easier to exploit the database. I used a search prompt attempting to find leaking MySQL errors and avoiding any discussions or forums on these errors,

```
"you have an error in your sql syntax" OR "mysql_fetch_array()" OR "mysql_num_rows()"
-"ask" -"help" -"questions" -"forum" -"forums" -site:github.com
```

This resulted in two websites with explicit errors attempting to retrieve information concerning posts and the date respectively, Drug Education Resources Center, Hong Kong and Cuadernos manchegos,



This relates to Injection as both websites not only exposed some parts of their database architecture but it is seen that the creators of this website are taking some input that has not been fully sanitized and displaying the output on the main page.

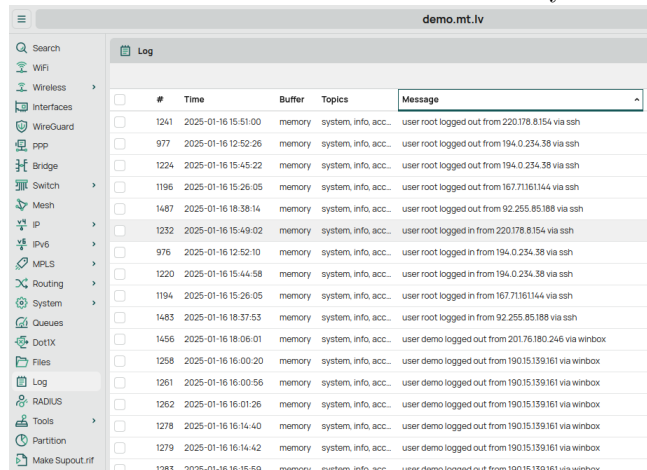
A04:2021-Insecure Design

Site: MicroTik

Insecure design would include any websites launched as a demo, testing phase, etc., yet still host sensitive information. I used the following prompt to find a website that allows you to log in as admin,

`nurl:staging OR inurl:demo OR inurl:beta`

The website MikroTik allows the user to log in as "admin" with no password and just clicking "Login". You can view internal files and web traffic on the site by clicking on "Files" or "Logs", respectively.



#	Time	Buffer	Topics	Message
1241	2025-01-16 15:51:00	memory	system, info, acc...	user root logged out from 220.178.8.154 via ssh
977	2025-01-16 12:52:26	memory	system, info, acc...	user root logged out from 194.0.234.38 via ssh
1224	2025-01-16 15:45:22	memory	system, info, acc...	user root logged out from 194.0.234.38 via ssh
1196	2025-01-16 15:26:05	memory	system, info, acc...	user root logged out from 167.71.161.144 via ssh
1487	2025-01-16 18:38:14	memory	system, info, acc...	user root logged out from 92.255.85.188 via ssh
1232	2025-01-16 15:49:02	memory	system, info, acc...	user root logged in from 220.178.8.154 via ssh
976	2025-01-16 12:52:10	memory	system, info, acc...	user root logged in from 194.0.234.38 via ssh
1220	2025-01-16 15:44:58	memory	system, info, acc...	user root logged in from 194.0.234.38 via ssh
1194	2025-01-16 15:26:05	memory	system, info, acc...	user root logged in from 167.71.161.144 via ssh
1483	2025-01-16 18:37:53	memory	system, info, acc...	user root logged in from 92.255.85.188 via ssh
1456	2025-01-16 18:06:01	memory	system, info, acc...	user demo logged out from 201.76.180.246 via winbox
1258	2025-01-16 16:00:20	memory	system, info, acc...	user demo logged out from 190.15.139.161 via winbox
1261	2025-01-16 16:00:56	memory	system, info, acc...	user demo logged out from 190.15.139.161 via winbox
1262	2025-01-16 16:01:26	memory	system, info, acc...	user demo logged out from 190.15.139.161 via winbox
1278	2025-01-16 16:14:40	memory	system, info, acc...	user demo logged out from 190.15.139.161 via winbox
1279	2025-01-16 16:14:42	memory	system, info, acc...	user demo logged out from 190.15.139.161 via winbox
1283	2025-01-16 16:15:59	memory	system, info, acc...	user demo logged out from 190.15.139.161 via winbox

This relates to insecure design as any user can log in as an admin and view sensitive data such as live connection requests and files stored within the website. The needed security controls to prevent a user from entering their website were never implemented, thus lending itself to insecure design.

A05:2021-Security Misconfiguration

Site: OPNsense

I wanted to attempt to find login credentials for users or admins through configs or default pages of websites. I specifically targeted .XML files under a config folder using this search prompt,

`inurl:"/config/" filetype:xml -site:github.com -site:android.googleusercontent.com`

I found a group of passwords and login information within an XML file attached to a website called OPNsense, however, I cannot confirm if this was for this specific website or another.

```
</sysctl>
-<system>
  <serialspeed>115200</serialspeed>
  <primaryconsole>serial</primaryconsole>
  <optimization>normal</optimization>
  <hostname>OPNsense</hostname>
  <domain>localdomain</domain>
  <dnsallowoverride>1</dnsallowoverride>
  <group>
    <name>admins</name>
    <description>System Administrators</description>
    <scope>system</scope>
    <gid>1999</gid>
    <member>0</member>
    <priv>page-all</priv>
  </group>
  <user>
    <name>root</name>
    <descr>System Administrator</descr>
    <scope>system</scope>
    <groupname>admins</groupname>
  </user>
  <password>
    $2y$10$YRVoF4SgsksrX0vOQjGieB9XqHPKra9K7d80B3RZdbYj21TwBfS
  </password>
  <uid>0</uid>
```

This relates to security misconfiguration as it unnecessarily exposes critical configuration information for the website, exposing passwords and the features that compose the application.

A06:2021-Vulnerable and Outdated Components

Site: Style SPB

I've seen that WordPress websites are extremely common, rarely updated, and come with quite a few vulnerabilities. Following this, I looked for websites containing README's with WordPress, by which I used the following search query,

```
"wordpress" inurl:readme.html
```

With this query, I found this website with this README detailing that it is using WordPress 3.4.1. Following, WordPress 3.4.1 has some notable vulnerabilities due to the fact that it is outdated.



Сначала главное

Добро пожаловать. WordPress для меня — очень особенный проект. Каждый разработчик и участник добавляет что-то свое, и вместе мы создаём нечто прекрасное, частью мне очень приятно быть. На WordPress ушли тысячи часов, и каждый день мы посвящаем его совершенствованию. Спасибо вам за то, что сделали его частью своего мира.

— Мэтт Мулленвег

Установка: Знаменитая 5-минутная установка

This relates to vulnerable and outdated components because the creators of this website failed to update their version of WordPress which has notable vulnerabilities.

A07:2021-Identification and Authentication Failures

Site: Hyper Salon's .env

A bad password is one that everyone knows, which is a blatant authentication failure. I structured this search query targeting files that exposed database passwords and login information,

```
intext:"db_password" OR intext:"db_user" filetype:env
```

With this query, I found Hyper Salon with this .env file exposing many of the major environment settings and passwords.

```
APP_ENV=local
APP_KEY=base64:nsUEsAyaQIhF6+W2hBopjA6ZzKYLGtdV5bZwak000B+
APP_NAME=Hypersaloon
APP_DEBUG=false
APP_LOG_LEVEL=debug
APP_URL=http://hypersaloon.cybussolutions.com
APP_TIMEZONE=Asia/Karachi

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=hypersaloon
DB_USERNAME=root
DB_PASSWORD=lk6mW60dpusb

BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
```

This relates to Identification and Authentication Failures as, despite them using strong seemingly randomized passwords, exposing a .env containing passwords stored in plain text allows anyone to violate their attempts at authentication.

A08:2021-Software and Data Integrity Failures

Site: Arch Linux RISC-V

This vulnerability concerns many internal features of the program, so I decided it best to use a search query looking for the build logs of websites and software,

```
ifiletype:log "deployment" OR "build succeeded" OR "build failed"
```

I did not find many obviously helpful websites or logs, however, I was still looking for build logs that showed failures, errors, or dependencies on out-dated or insecure frameworks that could be exploited. Here is an example log, however, to my knowledge, there is not much helpful information to be found.

This relates to Software and Data Integrity Failures as we can attempt to find the logs or config files of applications that may be using exploitable or unreliable libraries.

A09:2021-Security Logging and Monitoring Failures

Site: Nowcast Log

While most logging and security features are really only viewable by administrators, I constructed this query attempting to find server logs to expose if websites had any security frameworks or logging,

```
filetype:log OR filetype:txt inurl:log
```

While generically looking for logs might not expose much, I did find a somewhat live log from the University of Columbia. In viewing logs like these, we can assess if they have any logging or services for security.

```
2025-01-19 19:31:26,586 DEBUG [collect_weather] moved /SalishSeaCast/datamart/hrdps-continental/00/003/20250120T00Z_MSC_HRDPS_DS5RF_Sfc_RLatLon0.0225_PT003H.grib2 to /results/forcing/atmospheric/continental2.5/GRIB/20250120/00/003/
2025-01-19 19:31:27,027 DEBUG [crop_gribs] wrote GRIB file cropped to SalishSeaCast subdomain: /results/forcing/atmospheric/continental2.5/GRIB/20250120/00/003/20250120T00Z_MSC_HRDPS_DS5RF_Sfc_RLatLon0.0225_PT003H_SSC.grib2
2025-01-19 19:31:27,028 DEBUG [crop_gribs] observer thread files remaining to process: 505
2025-01-19 19:31:27,314 DEBUG [collect_weather] moved /SalishSeaCast/datamart/hrdps-continental/00/003/20250120T00Z_MSC_HRDPS_PRATE_Sfc_RLatLon0.0225_PT003H.grib2 to /results/forcing/atmospheric/continental2.5/GRIB/20250120/00/003/
2025-01-19 19:31:27,802 DEBUG [crop_gribs] wrote GRIB file cropped to SalishSeaCast subdomain: /results/forcing/atmospheric/continental2.5/GRIB/20250120/00/003/20250120T00Z_MSC_HRDPS_PRATE_Sfc_RLatLon0.0225_PT003H_SSC.grib2
2025-01-19 19:31:27,802 DEBUG [crop_gribs] observer thread files remaining to process: 504
2025-01-19 19:31:28,058 DEBUG [collect_weather] moved /SalishSeaCast/datamart/hrdps-continental/00/003/20250120T00Z_MSC_HRDPS_LHTFL_Sfc_RLatLon0.0225_PT003H.grib2 to /results/forcing/atmospheric/continental2.5/GRIB/20250120/00/003/
2025-01-19 19:31:28,066 DEBUG [crop_gribs] wrote GRIB file cropped to SalishSeaCast subdomain: /results/forcing/atmospheric/continental2.5/GRIB/20250120/00/003/20250120T00Z_MSC_HRDPS_LHTFL_Sfc_RLatLon0.0225_PT003H_SSC.grib2
2025-01-19 19:31:28,067 DEBUG [crop_gribs] observer thread files remaining to process: 503
```

This relates to Security Logging and Monitoring failures as we can view the logs of applications to see what they do and don't report on. Once we know what they do and do not scan for, we could possibly find common vulnerabilities to exploit that wouldn't be caught.

A10:2021-Server-Side Request Forgery

Site: Mental Health Law

For this vulnerability, I looked for websites that included common SSRF tags that could be potentially exploited by loading in certain URLs,

```
inurl:"url=" OR inurl:"uri=" OR inurl:"fetch=" OR inurl:"callback=" OR inurl:"link=" inurl:"http"
```

While SSRF vulnerabilities depend on how the website may use the website-requested link, I tried sending webhooks to various websites to see what they did with the link. In doing this, I found a UK website that loads a "special link". Using webhook.site, I sent a webhook that was received by the website, as seen by its logged IP. I am not well-versed with SSRF vulnerabilities, but instances like this could be used to infiltrate the service.

The screenshot shows the Webhook.site interface. At the top, there's a navigation bar with links like 'Docs & API', 'Custom Actions', 'WebhookScript', 'Terms & Privacy', and 'Support'. Below this is a toolbar with icons for 'Share', 'Schedule', 'Form Builder', 'CSV Export', 'Custom Actions', 'Replay', and 'XHR Redirect'. The main area is divided into two panels. The left panel, titled 'INBOX (1/100)', shows a list of webhooks with a search bar and a 'Newest First' sort option. A single entry is visible: 'HEAD #d0de2 161.35.175.57' with a timestamp of '01/19/2025 10:58:06 PM'. The right panel, titled 'Request Details', shows the details of the selected request. It includes a 'HEAD' tab, the URL 'https://webhook.site/9d3c600d-81b2-4eba-bb0a-642a303270ab', and various metadata: Host '161.35.175.57' (with links to Whois, Shodan, Netlify, Censys, and VirusTotal), Date '01/19/2025 10:58:06 PM (a few seconds ago)', Size '0 bytes', Time '0.000 sec', ID 'd0de2ad7-33bb-4f79-8cd8-b66657fb8d49', and a note 'Add Note'. Below this, the 'Query strings' section is empty, and the 'No content' message is displayed.

This relates to Server-Side Request Forgery as we sent a resource to the web application and, without doing checks, attempted to invoke a HTTP HEAD request.

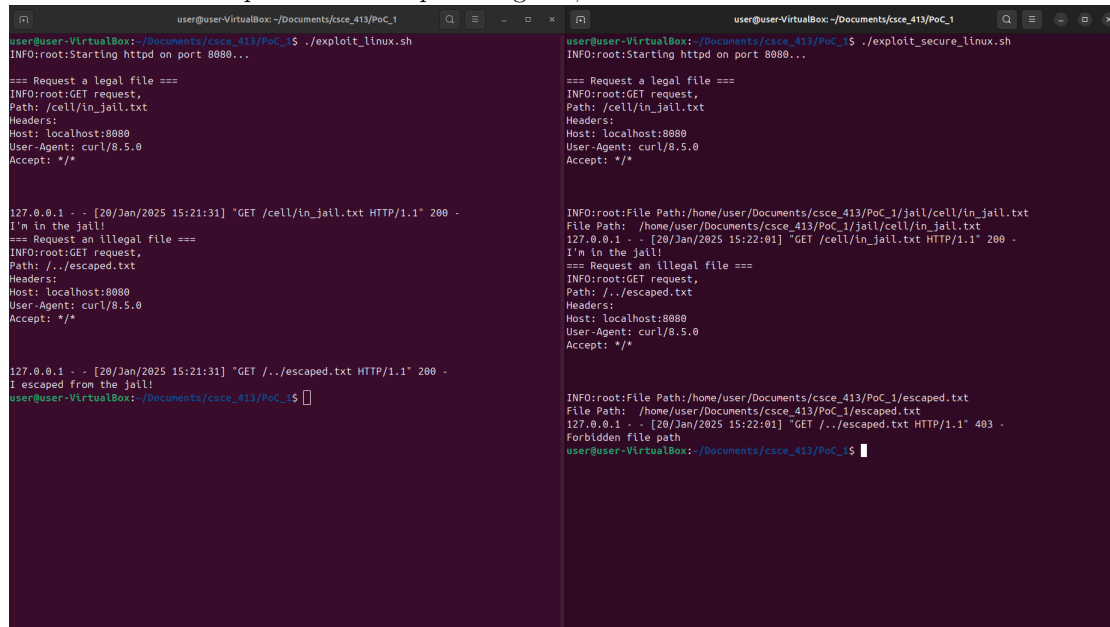
Practice

Create your own Web application vulnerable to directory traversal and exploit it. (Show code and screenshots)

Steps

1. Unzip PoC1_traversal.zip
2. Run `exploit_linux.sh` in `./PoC1/` to view the exploit performed in the `jail` directory.
3. (Optional) `exploit_secure_linux.sh` to view the exploit performed on the secure version of the server.

Here is a brief example of both scripts being ran;



```
user@user-VirtualBox: ~/Documents/csce_413/PoC_1
user@user-VirtualBox: ~/Documents/csce_413/PoC_1$ ./exploit_linux.sh
INFO:root:Starting httpd on port 8080...

=== Request a legal file ===
INFO:root:GET request,
Path: /cell/in_jail.txt
Headers:
Host: localhost:8080
User-Agent: curl/8.5.0
Accept: */*

127.0.0.1 - - [20/Jan/2025 15:21:31] "GET /cell/in_jail.txt HTTP/1.1" 200 -
I'm in the jail!

=== Request an illegal file ===
INFO:root:GET request,
Path: ../escaped.txt
Headers:
Host: localhost:8080
User-Agent: curl/8.5.0
Accept: */*

127.0.0.1 - - [20/Jan/2025 15:21:31] "GET ../escaped.txt HTTP/1.1" 200 -
I escaped from the jail!
user@user-VirtualBox: ~/Documents/csce_413/PoC_1$

user@user-VirtualBox: ~/Documents/csce_413/PoC_1$ ./exploit_secure_linux.sh
INFO:root:Starting httpd on port 8080...

=== Request a legal file ===
INFO:root:GET request,
Path: /cell/in_jail.txt
Headers:
Host: localhost:8080
User-Agent: curl/8.5.0
Accept: */*

INFO:root:File Path: /home/user/Documents/csce_413/PoC_1/jail/cell/in_jail.txt
File Path: /home/user/Documents/csce_413/PoC_1/jail/cell/in_jail.txt
127.0.0.1 - - [20/Jan/2025 15:22:01] "GET /cell/in_jail.txt HTTP/1.1" 200 -
I'm in the jail!

=== Request an illegal file ===
INFO:root:GET request,
Path: ../escaped.txt
Headers:
Host: localhost:8080
User-Agent: curl/8.5.0
Accept: */*

INFO:root:File Path: /home/user/Documents/csce_413/PoC_1/escaped.txt
File Path: /home/user/Documents/csce_413/PoC_1/escaped.txt
127.0.0.1 - - [20/Jan/2025 15:22:01] "GET ../escaped.txt HTTP/1.1" 403 -
Forbidden file path
user@user-VirtualBox: ~/Documents/csce_413/PoC_1$
```

Explanation

I created a simple Python HTTP file server that runs on Port 8080. The vulnerability in this code is found in the following lines found within the `do_GET` handler,

```
1 # Construct the file path
2 base_directory = os.getcwd()
3 base_path = self.path.lstrip('/')
4 # Truncate the path
5 file_name = os.path.join(base_directory, base_path)
```

In this code, `self.path` is the path specified in the HTTP GET request. The only modifications done to the acquired path is stripping any leftward `/` to prevent issues with joining the `base_directory`. This is a vulnerability, as anyone could send a request accessing the parent directories of where the server is being run, potentially accessing sensitive information.

For my example, I created the following file structure:

```
PoC_1/
├── jail/
│   ├── vulnerableServer.py
│   ├── secureServer.py
│   └── cell/
│       └── in_jail.txt
├── escaped.txt
├── exploit_linux.sh
├── exploit_secure_linux.sh
└── PoC1_grahamd.pdf
```

In this scenario, we would not want any GET requests to attempt to open the `escape.txt` file outside of the jail. Using curl, we can demonstrate this exploit by accessing the legal file `in_jail.txt`, and then attempting to access the illegal file `escape.txt`. `exploit_linux.bat` and `exploit_secure_linux.sh` are scripts that, when run, attempt the exploit on the insecure and secure versions of the python scripts respectively. What follows is a side-by-side view of the server being exploited, and breaking out of the jail;

As seen in this example, using the `--path-as-is` parameter for curl, it is possible to perform a directory traversal exploit.

A defense against this exploit is checking that the constructed path exists within the jail folder, and to send a 403 code otherwise,

```
1  # Construct the file path
2  base_directory = os.getcwd()
3  base_path = self.path.lstrip('/')
4  # Resolve a path from the base directory and given path
5  file_path = os.path.abspath(os.path.join(base_directory, base_path))
6  # * Ensure the path exists in the current working directory, otherwise, throw a 403
7  if not file_path.startswith(base_directory):
8      self.send_response(403)
9      self.end_headers()
10     self.wfile.write(b"Forbidden␣file␣path")
11     return
```

After the path resolution on line 5, we check that the path exists within the `base_directory` and, if it does not, we return a 403 code stating that the file path is forbidden. This prevents the traversal of any paths that are not within the jail directory.