Penetration Testing for Linux Jail as Exploit-resistant Linux Container

Shintaro Suzuki
Future University Hakodate
Hokkaido, Japan
g2123032@fun.ac.jp

Yuki Nakata
SAKURA internet Inc.
Hokkaido, Japan
y-nakata@sakura.ad.jp

Katsuya Matsubara
Future University Hakodate
Hokkaido, Japan
matsu@fun.ac.jp

***Abstract*—Containers provide lightweight and fine-grained isolation for computational resources, such as central processing units' CPUs, memory, storage, and networks. Still, they are not genuinely sandboxed as they share the host OS kernel. Consequently, vulnerabilities in the host OS kernel can be exploited to escape the container isolation. We have proposed a Container Transplantation approach by adopting FreeBSD as the runtime platform for Linux containers to reduce the attack surface to Linux kernel vulnerabilities with low overhead. To verify our proposal, this work runs code that exploits Linux kernel vulnerabilities on the Linuxulator to explore whether it can defend or neutralize the attacks. Our test adopts known Linux kernel vulnerabilities and attack codes released as proof of concept but excludes ones that depend on functionalities unsupported by the Linuxulator. Therefore, the test results would indicate whether differences in implementation provide effectiveness in resisting vulnerability exploits, even if they implement compatible OS functionalities.**

I. Introduction

Container-virtualization technologies, such as Docker [1], are now widely used as lightweight application-execution environments in cloud services. They are based on operating system (OS) processes, which are lightweight compared with traditional virtual machines (VMs), resulting in faster startup and a lower memory footprint. Many cloud services [2], [3] take advantage of the characteristics of containers for rapid scale-out, scale-in, and more efficient resource usage.

When using container virtualization as an application-execution environment in the cloud, cloud-service providers must ensure robust isolation between containers to prevent one user from negatively impacting another. Unfortunately, the OS kernel is shared among containers. This means that vulnerabilities in the OS kernel can be exploited to escape from the container to the host environment or obtain unauthorized root privileges [4]–[6]. Sandboxes that create additional isolation environments to address such attacks improve isolation between containers by creating VMs for each container [7] and isolating OS kernels between containers using user-space kernels [8].

However, creating additional isolation environments hinders rapid container startup and degrades application performance. Kata Containers [7] creates a VM for each container, isolating the OS to keep the impact of privilege escalation and container-escaping attacks contained within VMs. This hinders the characteristic of a fast container startup. The container sandbox gVisor [8] reduces the attack surface of the OS kernel by intercepting system calls issued by applications and redirecting them to a securely re-implemented user-space kernel. It significantly degrades the performance of applications due to intercepting all system calls issued by the application and reimplementing many OS resources in user space [9].

We aim to achieve a container sandbox with Container Transplantation in which Linux containers run compatibly in another OS and apply another OS-specific security mechanism to resolve the trade-off between robust container isolation and container characteristics and application performance. Since many cloud services use Linux as the environment for hosting containers, vulnerabilities in the Linux kernel and its functions are directly linked to container vulnerabilities. Unfortunately, attacks that exploit those vulnerabilities often have fatal adverse effects that make it impossible for cloud services to run correctly, such as unauthorized access to hosts and other containers through privilege escalation and the theft of sensitive information. We argue that executing Linux containers in another OS can avoid these various attacks that exploit vulnerabilities related to the Linux kernel and its functionality. It is also possible to apply unique security models and functions implemented in various OSes to Linux containers to achieve more finely grained access control and isolation between containers resilient to attacks targeting specific OSes.

To resolve the trade-off between robust container isolation, container characteristics, and application performance, we have proposed a lightweight container sandboxing that runs Linux containers compatibly on a heterogeneous OS and applies uncommon security mechanisms in Linux, named 'Container Transplantation' [10]. The implementation of the proposed Container Transplantation adopts FreeBSD as the heterogeneous OS.

This study investigates the possibility of resisting attacks using Linux kernel vulnerabilities through the Linux jails. Among exploit codes against the vulnerabilities in Linux Kernel 5.15.0, emulated by Linux jails on FreeBSD 14.0, we have chosen ones that the Linuxulator can execute. We then categorized the Linux kernel vulnerabilities that the secure container must be able to avoid, tried the attacks on both Linux and the Linuxulator, and summarized the success or failure of the attacks. Our experimental results indicate that the

Linuxulator can efficiently neutralize exploits against Linux kernel vulnerabilities with low overheads.

II. Container Transplantation in FreeBSD

We aim to enable Container Transplantation, which compatibly executes Linux containers on heterogeneous OSes and applies heterogeneous OS-specific security mechanisms to prevent attacks on containers that exploit vulnerabilities in a specific OS kernel. It makes the host OS and container's OS kernel different, so that attacks on the host OS that exploit OS kernel-specific vulnerabilities fail. It can be resistant to attacks targeting specific OSes by combining the isolation provided by conventional containers with heterogeneous OS-specific security mechanisms (e.g., limiting unnecessary system call invocations) for fine-grained access control/restriction. As a similar approach, live migration to heterogeneous environments is effective in mitigating attacks that exploit vulnerabilities in specific OS kernels or Virtual Machine Monitors (VMMs) [11], [12]. We expect our Container Transplantation approach to be similarly resistant to attacks that exploit those vulnerabilities because it runs virtual environments on a heterogeneous OS, similar to live migrations.

Our Container Transplantation approach reduces application performance and startup overhead compared with traditional approaches for container sandboxes while preventing attacks that exploit OS kernel vulnerabilities. It is a primitive emulation that runs containers created for one OS on a heterogeneous OS but without VMs. While VMs emulate computing resources and intercept all I/O, our approach translates differences between OSs only for system calls and OS functions within the OS kernel space. Therefore, it not only eliminates the OS startup process required for VMs but also reduces the overhead of application-performance degradation due to user-space interception and redirection.

We proposed a lightweight container sandbox that emulates Linux containers on FreeBSD, based on FreeBSD's LinuxJail, as one implementation of Container Transplantation [10]. LinuxJail can execute Linux binaries by using the Linuxulator, which translates system calls invoked by Linux binaries into FreeBSD system calls. It provides the Linux system call interface and translates arguments and options in kernel space, thus having less overhead on application performance. Furthermore, we combine LinuxJail with FreeBSD's Capsicum to reduce the attack surface. Figure 1 shows an overview of our Container Transplantation for FreeBSD.

When using a system-call translation such as the Linuxulator, we have to give due consideration to the compatibility of the system calls. With such translations, specific options or the system calls are not supported, limiting the applications that can be run. The Linuxulator has been reported to work with major server applications such as Apache HTTP Server and MySQL [13]. Our discussion of the compatibility between the Linuxulator and other methods [10] also concluded that it is sufficient to support common web applications running in cloud environments.

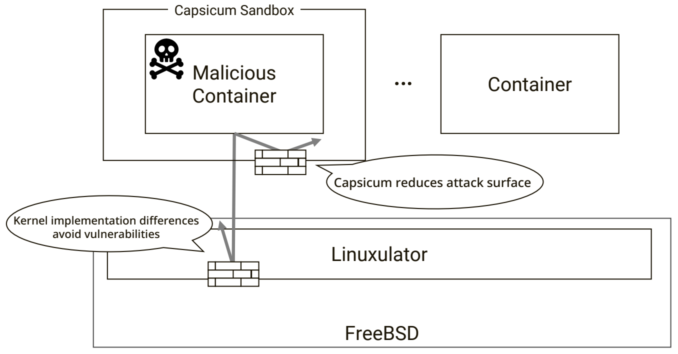

Capsicum Sandbox
Malicious Container
...
Container
Capsicum reduces attack surface
Kernel implementation differences avoid vulnerabilities
Linuxulator
FreeBSD

Fig. 1. Overview of Container Transplantation for FreeBSD.

III. Penetration Test

We evaluated the robustness of our proposed sandbox with Container Transplantation by testing whether the Linuxulator system call translation that we used in our implementation could prevent exploits that exploit known vulnerabilities in the Linux kernel implementation. The first step of this penetration test was to list the vulnerabilities and their exploit codes that can be exploited in the environment we targeted from among the large number of known vulnerabilities reported in the Linux kernel. We then executed the listed vulnerabilities on the Linux kernel and LinuxJail environment and compared the success or failure of the exploits.

A. OS Environments

It is important to standardize the Linux kernel version and settings related to security mechanisms on both Linux kernel environments and FreeBSD's Linuxulator to correctly compare the results of penetration tests. There are various Linux kernel vulnerabilities, including those that depend on a specific version and those that cross multiple versions, and differences in kernel versions are related to the success or failure of the execution of exploit codes. To evaluate the vulnerability in a realistic environment, it is also desirable to evaluate whether the exploit can be avoided with kernel versions that are supported by the Linux community and patched for the vulnerability. Many OSes have security mechanisms, such as memory-access restrictions and memory-address randomization, that can prevent the execution of certain exploit codes. However, since the support for these mechanisms differs among OSes, it is necessary to standardize the security mechanisms used so that evaluation can be based only on differences in kernel implementations.

We chose Linux Kernel 5.15.0 as the kernel version for our testing environment because it is the kernel version available for the Linuxulator and currently supported by the Linux community. We used FreeBSD version 14.0-RELEASE to use the Linuxulator with Linux Kernel 5.15 compatibility.

In the experimental environment, the Supervisor Mode Access Prevention (SMAP) and Supervisor Mode Execution Prevention (SMEP), mechanisms for restricting unauthorized memory access, were enabled, and Kernel Address Space Layout Randomization (KASLR), which randomizes kernel memory addresses, was disabled. SMAP prohibits access to user-space memory from kernel space, and SMEP prohibits the execution of user-space code while kernel-space code is executing. KASLR applies ASLR to the kernel to randomize binary memory addresses. SMAP and SMEP were enabled because they are implemented in both Linux and FreeBSD. However, KASLR is implemented in Linux but not in FreeBSD, so we disabled KASLR in Linux.

B. exploit Codes

We are currently evaluating known vulnerabilities in the Linux kernel 5.15, although not all published vulnerabilities can be used in our experiments. Vulnerabilities that cannot be used in our experiments are those under the following conditions.

- *Vulnerability related to functions that do not exist in Linuxulator*: the library or functionality required by the exploit code is not available, so the exploit does not always succeed
- *Architectures not expected to be used in cloud environments*: Since we targeted cloud environments, CPU architectures used in embedded environments or laptops are out of scope
- *Features not expected to be used in cloud environment*: Vulnerabilities related to specific hardware devices or minor protocols are not included because they are not available in the cloud
- *exploit code is not published*: If the exploit code is not published on a vulnerability-reporting page or source-code hosting service, it is not covered because it cannot be reproduced

On the basis of these conditions, we mechanically selected 12 available vulnerabilities out of the 600 vulnerabilities that had been publicly reported as of January 16, 2024. All vulnerabilities related to the Linux kernel are managed by version at Linux Kernel CVE, a project that tracks Linux vulnerabilities. It lists vulnerabilities in the form of a CVE-ID, the commit that was fixed, function for which the vulnerability was found, version in which it was fixed, and a description of the vulnerability. We excluded vulnerabilities that fit conditions 1, 2, and 3 on the basis of the function in which the vulnerability was found listed in it and extracted 408 vulnerabilities. For the remaining vulnerabilities, we surveyed vulnerability-reporting pages and source-code hosting services such as GitHub for the existence of exploit code and excluded those we could not find, resulting in the extraction of 14 vulnerabilities. Figure 2 shows the flow of vulnerability extraction.

C. exploit Environment

We used a VM for each exploit code and ran the exploit on an independent environment. The VMs were VirtualBox, with 4 GB and 4 cores per VM. Some exploits made a single core desirable, depending on the exploit. The Linux VMs used Ubuntu 22.04 as the VM image, built Kernel 5.15.0,

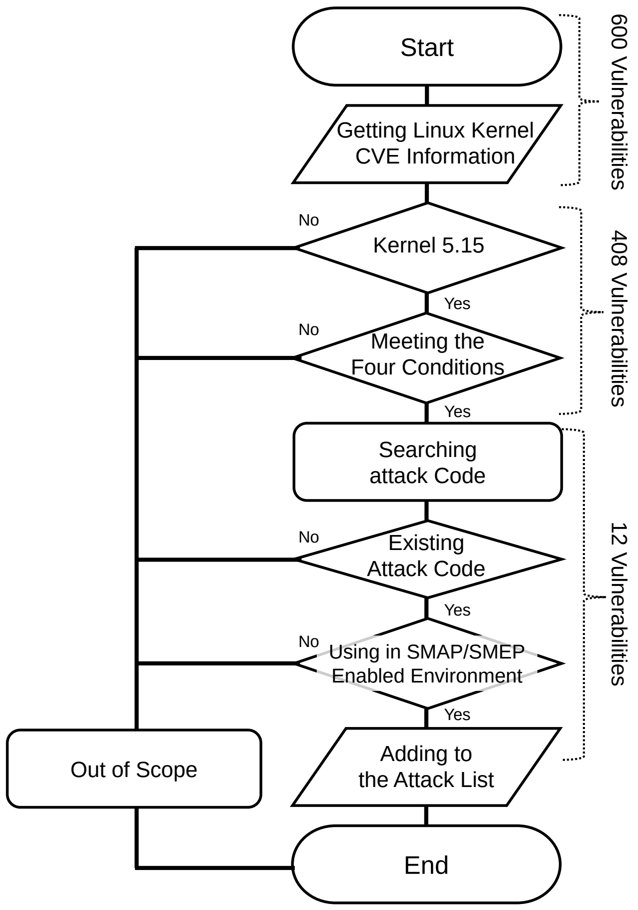


Fig. 2. The flow of extracting vulnerabilities to be evaluated.

and booted it. For the kernel config, we used localmodconfig from generic/ubuntu2204. The kernel configurations that had to be changed for each exploit code were changed as needed. We configured grub to enable smap, smep and disable kaslr. The FreeBSD VM was based on FreeBSD 14.0-RELEASE, and the kernel build was done according to the exploit. The exploit used bastille to manage the jail, and Linux emulation was used. exploit tests were conducted in this environment.

D. Test Results

In our experiments, we tested exploit codes on Linuxulator that had previously succeeded on Linux. However, we didn't test codes relying on Linux-specific features not available in Linuxulator, focusing instead on the underlying vulnerabilities. For instance, eBPF not supported in FreeBSD, and fuse, it can't be used on Linuxulator, were excluded from our tests.

Success datermination of the exploit depends on the each code. When explicit success indicators were provided within the exploit code, we adhered to them. For race condition exploits, we followed the set number of trials when specified, otherwise considering the exploit unsuccessful if it didn't succeed within 24 hours. For result is shown I. confirmed (exploit confirm), failed (exploit failed), N/A (can't build)

TABLE I
LIST OF PENETRATION TESTING AT LINUX VULNERABILITIES.

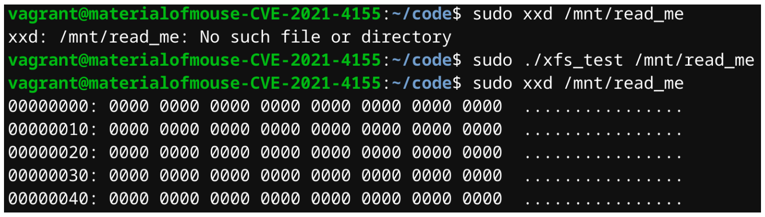
CVEID	Vulnerability detail	result in Linux	result in Linux jail
CVE-2021-4155	data leak caused by ’IOCTL with XFS’	confirmed	failed (target functionality not exist)
CVE-2022-2585	UAF caused by ’POSIX CPU timer’	failed (race condition never reproduced)	not tested
CVE-2022-2588	UAF caused by ’net scheduler’	failed	not tested
CVE-2022-24122	UAF and privilege escalation caused by ’ucount’	failed (No Kernel crashed)	not tested
CVE-2022-25258	Memory corrupt caused by ’USB Gadget subsystem’	N/A (USB gadget func. required in H/W)	not tested
CVE-2022-25265	Arbitrary code execution caused by binary files exec-all attribute	failed	not tested
CVE-2022-27666	privilege escalation caused by ’IPsec ESP’	failed	not tested
CVE-2022-36402	privilege escalation or Dos caused by ’vmwgfx driver’	failed	not tested
CVE-2022-38096	privilege escalation or Dos caused by ’vmwgfx driver’	failed	not tested
CVE-2023-0045	data leak caused by Spectre-BTI	failed	not tested
CVE-2023-3640	privilege escalation caused by ’x86 memory management’	failed	not tested
CVE-2023-6606	crash system or data leak caused by ’samba cifs’	N/A (Error occurred)	not tested

The results showed that most exploits that succeeded on Linux. The conditions necessary for a successful exploit are difficult to reproduce and were caused by the complexity of deciphering the exploit code It due to their use of magic numbers and unconventional functions. Identifying the necessary build and kernel options for setting up a comparable exploit environment proved challenging.

Additionally, certain exploit codes employed loopback devices and system calls like unshare and fuse, which lack emulation in Linuxulator, thus couldn’t be directly executed. We modified such codes to work on Linuxulator where possible, substituting functionalities with equivalent ones available in FreeBSD or Linuxulator, and replacing virtual devices with physical ones where necessary. Some exploits were abandoned due to infeasibility of emulation, such as with fuse, which, despite being available in FreeBSD, couldn’t be used on Linuxulator. Through these modifications, we aimed to adapt the exploit codes for Linuxulator to the best of our ability. However, the only exploit we managed to verify, CVE-2021-4155, involved an IOCTL exploit that failed because the specific system call options used were not supported on Linuxulator

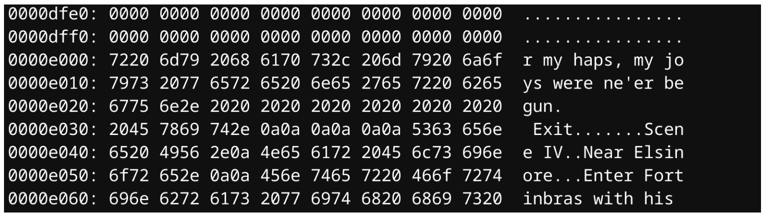
E. Analysis of CVE-2021-4155 Exploit

The vulnerability, CVE-2021-4155, which allows unauthorized access to files by using the IOCTL XFS_IOC_ALLOCSP on devices with an XFS filesystem, only succeeded in a Linux environment. This was due to the exploit leveraging a system call option not supported by Linuxulator. For the exploit, a loopback device was used to mount the XFS filesystem, writing an XFS-formatted img file to loopback, then mounting it at /mnt. The exploit code, when executed against /mnt/read_me, enabled illegal reading of the XFS data. Given the unavailability of the loopback device in Linuxulator, we adapted the code to use a physical partition instead. This modified code was tested on both Linux and Linuxulator environments. The execution results are shown in III-E, III-E, III-E On Linux, the exploit was successful, allowing the



```
vagrant@materialofmouse-CVE-2021-4155:~/code$ sudo xxd /mnt/read_me
xxd: /mnt/read_me: No such file or directory
vagrant@materialofmouse-CVE-2021-4155:~/code$ sudo ./xfs_test /mnt/read_me
vagrant@materialofmouse-CVE-2021-4155:~/code$ sudo xxd /mnt/read_me
00000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Fig. 3. CVE-2021-4155 result on Linux 1



```
0000dfe0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000dff0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000e000: 7220 6d79 2068 6170 732c 206d 7920 6a6f  r my haps, my jo
0000e010: 7973 2077 6572 6520 6e65 2765 7220 6265  ys were ne'er be
0000e020: 6775 6e2e 2020 2020 2020 2020 2020 2020  gun.
0000e030: 2045 7869 742e 0a0a 0a0a 0a0a 5363 656e   Exit.......Scen
0000e040: 6520 4956 2e0a 4e65 6172 2045 6c73 696e  e IV..Near Elsin
0000e050: 6f72 652e 0a0a 456e 7465 7220 466f 7274  ore...Enter Fort
0000e060: 696e 6272 6173 2077 6974 6820 6869 7320  inbras with his
```

Fig. 4. CVE-2021-4155 result on Linux 2

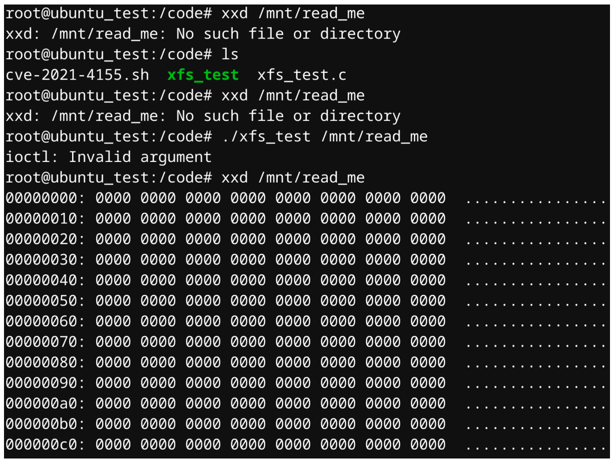
contents of the img file to be read when executing xxd against /mnt/read_me. However, on Linuxulator, the eploit failed, with xxd revealing only 0-filled data, due to the absence of the IOCTL option XFS_IOC_ALLOCSP. XFS_IOC_ALLOCSP is an IOCTL option and is related to file size expansion. Fortunately, the functionality must be minor and useless for most container applications.

IV. CONCLUSION

While it is known that vulnerabilities of the underlying OS kernel may allow to escape the container isolation, the finding of this penetration test is that if Linux containers can be transplanted to Linux jails on FreeBSD, it may be possible to neutralize exploits to the Linux kernel vulnerabilities. This study aimed to evaluate the vulnerability resistance of Linux jails by executing code that attacks known Linux vulnerabilities on both Linux and Linux jails. Unfortunately, reproducing the attacks was not easy. Many attack codes released as PoCs are poorly documented; even if the reproduction of

TABLE II
LIST OF MD5 HASHES OF EXPLOIT CODE URL

CVE ID	Source of Exploit Code	
	Repository	MD5 hash of URL
CVE-2021-4155	gist	ec70d8a77cab68fcd24b149046c7dffd
CVE-2022-2585	github	19ec3bf26fbef0c3f1929b7f771d9181
CVE-2022-2588	github	e241e11a514c93846a95c358276db850
CVE-2022-24122	github	08dbd96faed08fa20e0e9e0c1ace7af1
CVE-2022-25258	github	fe0a7553ecc457fe230b4031fff3650f
CVE-2022-25265	github	6bdca7fc4f72aca7e66dafb172e179b3
CVE-2022-27666	github	25fb27e898ac93e88b2366e64fa83ebb
CVE-2022-36402	github	7a0b62ff3eb293c829e52807d0bb2671
CVE-2022-38096	github	52ae378444a25983fca01639bf0c85d1
CVE-2023-0045	github	76f65a6f26c638be1ad2f86ac57fb9cd
CVE-2023-3640	github	647ad4e927826b1e18abe585598c0429
CVE-2023-6606	bugzilla	1518be6f4c8e9e791e31a5c5c7f842a1



```
root@ubuntu_test:/code# xxd /mnt/read_me
xxd: /mnt/read_me: No such file or directory
root@ubuntu_test:/code# ls
cve-2021-4155.sh  xfs_test  xfs_test.c
root@ubuntu_test:/code# xxd /mnt/read_me
xxd: /mnt/read_me: No such file or directory
root@ubuntu_test:/code# ./xfs_test /mnt/read_me
ioctl: Invalid argument
root@ubuntu_test:/code# xxd /mnt/read_me
00000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Fig. 5. CVE-2021-4155 result on freebsd

the attack on Linux fails, the unreadable and abnormal code implementation could hinder the analysis of the exploit failure. In addition, the code implementation, which is not directly related to the vulnerability, uses Linux-specific features such as namespace, which has required some code modification to execute the code on Linux jails. The penetration tests have confirmed that an exploit against CVE-2021-4155, a vulnerability in Linux kernel 5.15.0, makes it fail by running it in a Linux jail. This is because the functionality that causes the vulnerability is not supported by Linuxurator.

The following two points need to be considered in our future work. We plan to implement a robust Linux-compatible container environment against vulnerabilities in the OS kernel and libraries by establishing a method of applying Capsicum and Casper to Linux jails. In addition, since new vulnerabilities are still being discovered one after another, we would like to evaluate to what extent Linux jails can resist zero-day attacks against these new vulnerabilities.

REFERENCES

[1] D. Merkel, “Docker: Lightweight linux containers for consistent development and deployment,” *Linux J.*, vol. 2014, no. 239, Mar. 2014.

[2] I. Salesforce.com. Cloud application platform — heroku. (Accessed on 2022/10/18). [Online]. Available: https://www.heroku.com/

[3] G. LLC. Cloud functions — google cloud. (Accessed on 2022/10/18). [Online]. Available: https://cloud.google.com/functions

[4] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, “A measurement study on linux container security: Attacks and countermeasures,” in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC ’18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 418–429.

[5] S. Sultan, I. Ahmad, and T. Dimitriou, “Container security: Issues, challenges, and the road ahead,” *IEEE Access*, vol. 7, pp. 52 976–52 996, 2019.

[6] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang, “Containerleaks: Emerging security threats of information leakages in container clouds,” in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 237–248.

[7] O. I. Foundation, “Kata containers - open source container runtime software,” (Accessed on 2021/05/08). [Online]. Available: https://katacontainers.io/

[8] T. gVisor Authors, “gvisor,” 2021, (Accessed on 2021/05/08). [Online]. Available: https://gvisor.dev/

[9] E. G. Young, P. Zhu, T. Caraza-Harter, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “The true cost of containing: A gvisor case study,” in *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. Renton, WA: USENIX Association, Jul. 2019.

[10] Y. Nakata, S. Suzuki, and K. Matsubara, “Reducing attack surface with container transplantation for lightweight sandboxing,” in *Proc. of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems*, 2023, pp. 58–64.

[11] T. D. Ngoc, B. Teabe, A. Tchana, G. Muller, and D. Hagimont, “Mitigating vulnerability windows with hypervisor transplant,” in *Proceedings of the Sixteenth European Conference on Computer Systems*, ser. EuroSys ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 162–177. [Online]. Available: https://doi.org/10.1145/3447786.3456235

[12] K. Matsubara and Y. Takagawa, “Adaptive os switching for improving availability during web traffic surges: A feasibility study,” in *Proc. of 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2020, pp. 1176–1182.

[13] “LinuxApps - FreeBSD Wiki — wiki.freebsd.org,” https://wiki.freebsd.org/LinuxApps, [Accessed 11-12-2023].