CyExec*: A High-Performance Container-Based Cyber Range With Scenario Randomization

RYOTARO NAKATA AND AKIRA OTSUKA, (Member, IEEE)
Institute of Information Security, Yokohama 221-0835, Japan
Corresponding author: Ryotaro Nakata (dgs184101@iisec.ac.jp)

ABSTRACT With increasing threats to information security, information security education through practical exercises specifically cyber range has attracted attention. However, the use of a cyber range is not widespread because of the high initial and maintenance cost and difficulty of developing new scenarios. Because many virtual instances are executed in the cyber range, the advantage of container type virtualization, which can provide a lightweight execution environment, is expected to increase efficient hardware utilization and decrease the total cost. On the other hand, containers pose challenges in scalability and scenario development when it comes to their use in cyber ranges because their performance advantages and vulnerability reproducibility have not been reported. In this paper, we conducted an exhaustive experiment to compare the performance and reproducibility of container-type virtualization with other virtualization types. The results show that containers can provide a more efficient execution environment than the other types, with almost perfect vulnerability reproducibility of more than 99% while reducing memory consumption by half and storage consumption to 1/60. The container’s high performance and reproducibility enabled us to develop CyExec*, a cyber range system with DAG-based scenario randomization technology. CyExec* can increase educational effectiveness by automatically generating multiple scenarios with the same learning objective. Compared with a random scenario generator for CTF using another virtualization type, CyExec* shows more than three times higher performance. CyExec* can solve existing cyber range issues.

INDEX TERMS Container, cyber attacks, CSIRT, cyber range, Docker, incident response, information security education, virtualization technology.

I. INTRODUCTION
With the rapid development of information and communication technology, the internet is being used within various systems and services. At the same time, security risks such as cyber-attacks are increasing, which means the need for security education is growing.

Security education is conducted for various purposes and at various levels, such as training professionals who are active in the real world, such as SOC operators, improving security awareness for the general public, and training students in educational institutions. The content of education is not only classroom lectures, but also practical training through exercises; here, and education using cyber ranges, which are practical and highly effective, has been attracting attention [1], [2]. However, commercially available cyber range systems can cost millions of dollars. The initial cost and human and economic costs while maintaining the system are also high. These costs make it difficult, especially for universities and small businesses, to implement and maintain effective cyber ranges. [3]. In addition, the development of exercise scenarios is a complex task; thus, the number of scenarios used is very limited. Therefore, there is a risk that the effectiveness of the education will be reduced because of scenario leakage or fraud [4], [5].

The high educational effectiveness of a cyber range has a wide range of needs, not only for training professionals, but also for improving general cyber-security awareness. However, because of the above environmental and scenario concerns, the spread of the cyber range has been limited [1], [6]. Hence, we have focused on a lightweight execution environment and high portability of container-type virtualization for some time. We have developed a container-based cyber range system called CyExec [6] and
have been promoting its use as a cyber range ecosystem that can be jointly developed and used by multiple educational institutions. However, container-type virtualization is very different from the other virtualization types in conventional cyber ranges, and its superiority in terms of operating performance has not been demonstrated. In addition, it is not reported whether container-based systems can completely reproduce vulnerabilities and incidents in other virtualization types. The performance advantages and the vulnerability reproducibility of containers are not clear, hindering scalability and scenario development for container-based cyber ranges [5].

In the current paper, we conducted performance experiments using a cyber range environment to dispel concerns about container-based cyber ranges. We confirmed that container-type virtualization has an advantage in using less than half the memory usage and less than 1/60 the storage usage than other virtualization types. In addition, through exhaustive experiments using vulnerability inspection tools and attack modules, we confirmed that our system could reproduce more than 99% of the vulnerabilities required for a cyber range compared with other virtualization types.

Because a cyber range environment can make the most of container-type virtualization capabilities, we have developed a container-based cyber range, CyExec*, which automatically generates multiple scenario execution environments with the same learning objectives, here using DAG-based scenario randomization technology. By fully utilizing the functions of Docker, which is a container-type virtualization platform, we can efficiently utilize hardware performance and realize an effective exercise environment that prevents the use of the same scenario and degradation of educational effectiveness because of scenario leakage.

CyExec* can expand the scope of information security education as a next-generation cyber range platform, producing a high learning effect that can realize the same performance as existing cyber ranges but at a low cost while providing many scenarios.

We summarize the main contributions of this paper as follows:

- We provide an overview of the virtualization technologies used in cyber ranges and present that containerized virtualization can optimize cyber range environments.
- We present that the vulnerability reproducibility of the container-based cyber range is comparable to other virtualization types through exhaustive experiments and no problem in executing exercise scenarios.
- We present a DAG-based scenario randomization technique that leverages the performance of container-based cyber ranges and develop it as a cyber range platform that can be widely deployed.
- Finally, we present the key issues that must be addressed in the future to achieve a more practical cyber range exercise environment.

II. CYBER RANGE AND VIRTUALIZATION

A. CYBER RANGE

A cyber range is a general term used to describe a system that allows users to learn how to attack and defend themselves in cyberspace through real operations. Learners can efficiently improve their knowledge of attacks and defenses and their incident response skills through experiential learning either alone or in teams [7].

The cyber range reproduces individual attack methods and vulnerabilities and teaches incident response methods and a series of scenarios that start with an information security incident all the way to the completion of the response. Therefore, it is possible to prepare a medium- to large-scale system environment equivalent to the real world and learn response methods based on various roles, such as CISO (chief information security officer) and engineers, just as in a real company or organization. There is also a growing need for cyber ranges in all aspects, not only for training professionals, but in addition for increasing cybersecurity awareness among the general public [1].

Many studies have been conducted regarding the development and dissemination of cyber ranges, such as efficient cyber range execution environments and tools to develop realistic training scenarios [8]. Recently, containerized virtualization technology, which enables the efficient use of hardware resources, has been studied and verified, and it is attracting attention as a key platform technology for the widespread use of cyber ranges [9].

B. VIRTUALIZATION TECHNOLOGY

When reproducing an information security incident as a cyber range scenario, it is not realistic to use the real system environment because of cost and security issues. In addition, because multiple environments need to be prepared for each exercise scenario, learner, and group, it is necessary to prepare exercise environments smoothly, such as duplication, resetting the system, and replacement of environments. Therefore, a cyber range usually utilizes virtualization technology to prepare a virtual system environment that is equivalent to the real system, creating an environment with actual incidents and vulnerabilities [3]. Figure 1 shows the types and overview of virtualization technologies.

Many cyber range products and research to date have mainly used hypervisor (HV) or host-type virtualization software to build virtual environments, which reproduce the system environments of client terminals, server devices, security devices, and so on [1]. HV-type and host-type virtualization makes it possible to build an almost identical environment to a real machine. Thus, it is highly capable of reproducing vulnerabilities and attacks, but it requires high-performance hardware that can run the required number of virtual machines concurrently.

As the complexity of cyber-attacks and need for information security human resources development increase, the opportunities for training exercises increase as well, and
the construction of a more significant number of virtual instances in a shorter time is required. When it comes to scalability, research and verification on the construction of training environments using container-type virtualizations are being conducted [6], [10].

Table 1 shows the characteristics of each virtualization type.

### TABLE 1. Characteristics of each virtualization type.

| Type       | Isolation Level | Overhead | Guest OS | Example          |
|------------|-----------------|----------|----------|------------------|
| Hypervisor | max             | high     | require  | VMware ESXi      |
|            |                 |          |          | Citrix XenServer  |
|            |                 |          |          | Microsoft Hyper-V |
| Host       | high            | max      | require  | Docker           |
|            |                 |          |          | LXD              |
| Container  | low             | low      | not require | VMware player |
|            |                 |          |          | Dockerd         |
|            |                 |          |          | LXD              |
|            |                 |          |          | OpenVZ           |

The isolation level represents the independence from the host OS, other virtual machines, and containers running on the same hardware. If the isolation level is low, there is a high possibility that another virtual machine or container will be affected when a heavy processing load or serious trouble occurs on one virtual machine or container [11].

The overhead represents the degradation of the processing performance that occurs in the virtual environment. Because the hardware is accessed through the virtualization mechanisms, there will be a reduction in performance than the real machines [12]. This degradation is especially significant when running many virtual instances on a single cyber range environment. Figure 2 shows an operational image of various cyber range environments built on top of each virtualization type.

The HV and host types have a high isolation level and can provide a stable virtual environment. However, they have significant overhead, and guest OS operation is mandatory. Therefore, as the number of virtual instances increases, the number of running processes increases significantly and the required hardware resources increase significantly as well. Therefore, sufficiently powerful CPU, memory, and storage are needed to provide a comfortable exercise environment [13].

A container type virtualization does not necessarily require running a guest OS. The minimum number of processes are separated from the host OS and run, so the number of processes required is much smaller than other virtualization types that require a guest OS. Hardware access is handled directly by the kernel on the host OS, so there is little overhead. Further, there is no need for OS startup and shutdown processes, so lightweight and quick operation can be expected [14]. However, containers have a low isolation level and share many parts with the host OS, making them susceptible to the influence of the host OS and other containers.

### C. CYBER RANGE ISSUES

Almost all existing cyber ranges are HV-based or host-based cyber ranges, hence requiring high-performance hardware and contributing to the high cost. In addition, educational institutions, small organizations, and enterprises may not be able to prepare sufficiently powerful hardware, which hinders the widespread use of cyber ranges.

Even after introducing cyber-range systems, many scenarios are required to meet the learners’ level and learning objectives. Because cyber range exercises are often conducted by multiple teams simultaneously, the scenarios can be inferred from the progress of other teams and the content of their conversations. In addition, in recent years, there has been a growing need for online exercises. However, there is a problem here: learners may directly contact each other and leak the scenario content to other teams.

Consider a container-type virtualization that can be utilized in cyber range environments. In this case, efficient hardware utilization can promote the widespread use of inexpensive and easy-to-deploy cyber ranges. However, it is unclear how much of a performance advantage container-type virtualization has over other virtualization types, and this has become a barrier to scalability. In addition, because container-type virtualization has different characteristics when compared with other virtualization types, it is unclear to what extent this type can correctly reproduce real-world vulnerabilities, which is a barrier to scenario development.

Therefore, to promote the container-based cyber range, the following points need to be addressed:

- To quantify the performance advantage of container-type virtualizations in a cyber range.
- To reveal the reproducibility of vulnerabilities of container-type virtualizations in a cyber range.

If the above points are resolved, the concerns of container-based cyber range can be resolved. A cyber-range
system that can be easily implemented and operated by enterprises and educational institutions can be realized.

III. PERFORMANCE COMPARISON

A. EXPERIMENTAL DETAILS

To confirm the performance advantage of container-based cyber ranges compared with other virtualization types, we ran multiple equivalent environments created for the cyber range environment in each virtualization type, here assuming a typical cyber range exercise and then compared the consumption of resources required for the exercise environment, such as CPU, memory, and storage. It is more cost-effective to prepare an exercise environment that can be operated with fewer resources because then it is possible to perform an exercise with less-capable hardware.

We also compared the time required to start up each environment. The faster the startup time, the less time it will take to prepare the scenario for the actual exercise and the more smoothly the exercise can be carried out, even when the environment is replicated or replaced. The larger the exercise environment, the more critical the startup time needs to be.

In the experiment, we ran multiple cyber ranges to simulate the actual exercise. To put a high load on each of them, we let the vulnerability checking tool run after boot. In this way, we could reproduce a system environment in which many virtual instances would be running simultaneously, much like in the actual exercise.

B. EXPERIMENTAL ENVIRONMENT

Table 2 shows the environment used in the experiment.

We used VMWare ESXi for the HV-type, Oracle VirtualBox for the host-type, and Docker for the container-type platforms to construct the environment. VMWare has a large share of the market for HV virtualization software and is also used in commercial cyber ranges. VirtualBox is a free host virtualization software widely used in cyber range research applications. Docker is the de facto standard for containerized virtualization platforms, and its use is rapidly expanding. Both can be used to build a cyber-range exercise environment and were judged to be suitable for the construction of the experimental environment.

In the experiment, we prepared a machine running Ubuntu as a learner’s terminal, a Metasploitable2 server, which is a test environment with deliberate vulnerabilities, and kali linux, which has various tools installed for attacks and investigations.

We ran several of these environments simultaneously, measured the time required for startup, and then deliberately overloaded them to measure the overall resource consumption.

C. EXPERIMENTAL RESULTS

Figure 3 shows a comparison of the time required for booting. The HV type took the longest time to start up the first environment at 1 min 53 sec. However, the start-up time did not increase significantly as the number of exercise environments increased gradually. When the tenth exercise environment was started up, it took 6 min 11 sec. For the host type, the first environment started up as fast as 58 sec. However, the number increased significantly from the start of the second environment, and the tenth environment took a very long time to start up at 34 min 42 sec. Because the container type does not have an OS boot process, it boots up very quickly, with the first environment booting up in 6 sec. There was no significant increase after the second environment, and the tenth environment booting up in 1 min 27 sec, which is the fastest result compared with the other virtualization type.

Figure 4 shows the results of the comparison of resource consumption.
We measured the memory usage, storage usage, and CPU usage of up to 10 environments. For the HV type, we checked the resource consumption using the values displayed in the ESXi management console, and for the host and container types, we checked the values obtained from the system monitor and sar commands on the host OS. We compared the values after 10 min of starting the environment and running the tools.

In all the virtualization types, the amount of memory used increased almost in proportion to the number of environments. The HV and host types used almost the same amount of memory, even when the number of environments increased, with each environment using about 8 GB. The container type used only about 4 GB per environment, which is about less than half the other virtualization types.

The HV and host types consumed almost the same amount of storage, but the container type consumed less than 1/60th of the storage of the other types.

The HV type consumed the least CPU, and the host type consumed the most, but there was no significant difference in the consumption between the various virtualization types. However, there was a significant difference in the operation of each virtual machine. The HV type did not show much effect of delays in starting each virtual machine or delays in operations, even when the number of virtual instances increased. In contrast, for the host type, there were significant
delays in startup and operation after startup, which affected the exercise to the extent that it became difficult.

D. DISCUSSION
The results confirm the performance advantage of the container type. Regarding the startup time, the container type did not have the usual OS startup process, so that the environment can be started and operated quickly. As with the other virtualization types, the startup time increased slightly as the number of exercise environments increased, but the increase was smaller than that of the other virtualization types. Therefore, the larger the exercise environment, where the number of virtual instances would further increase, the faster the containerized environment can be launched and the smoother the exercise can be conducted. Because the speed of startup time is an essential factor when the environment is replaced because of changes in the scenario or when the exercise is conducted for different students on consecutive class schedules, the use of the container type is suitable.

The results also show an advantage in resource consumption.

Compared with the other virtualization types, the container type can reduce unnecessary processes, thus saving a lot of memory consumption. In the experiment, the container type used about half the memory when compared with the other virtualization types. The difference is expected to be even more significant when the number of virtual instances per exercise environment increases in a large-scale environment.

The storage usage varies depending on the scenario, but in the experiment’s environment, the container type was overwhelmingly superior to the other virtualization type, using 1/60th of the storage.

For storage consumption, container-type virtualization showed a significant advantage. The UFS (union file system) used in Docker, even if the container is duplicated and increased, does not require double the storage capacity as the HV and host types do. Any additions or changes made to each container are managed as a difference from the original image. Therefore, even if the storage space that is consumed increases slightly during operation, the container can be stopped and discarded after the exercise is over. The storage space is not consumed more than the original image and can be operated with much less space consumption.

There was no significant difference in the CPU consumption when the load was applied to any virtualization type. This is thought to be because regardless of the type of virtual environment, the content of the operation performed as an exercise was the same and there was no significant difference in the CPU consumption of the processes running as a whole. However, in actual operation, the host type, which requires a large load, including the host OS, has a sensory delay that significantly affects the startup and operation and is not suitable for operation in a large-scale exercise environment.

From these results, the container-based cyber range was found to be superior compared with the other virtualization types as an execution environment for exercises. Although it depends on the scale of the exercise environment, the container type can be beneficial especially in large-scale exercise environments because it can use hardware more efficiently than the other virtualization types in terms of memory consumption and storage usage. It can realize a comfortable exercise environment, even if memory or storage specifications are reduced. This result is significant not only when deploying hardware on premise for cyber range deployment, but also when selecting hardware resources in a cloud environment.

The results provide a guideline for conducting exercises using a cyber range in various forms.

IV. VULNERABILITY REPRODUCIBILITY ASSESSMENT
A. EXPERIMENTAL DETAILS
To assess the reproducibility of vulnerabilities on a container-based cyber range, we model the OS/HW environment in which the programs on a cyber range as an oracle \( \mathcal{O} \) and every system call invoked by a program \( A \) is sent to the oracle \( \mathcal{O} \). If a program is run in the environment over Real, HV, host or container, we write the output of the programs as \( A^{\text{Real}}, A^{\text{HV}}, A^{\text{Host}}, \) and \( A^{\text{Container}} \), respectively, where “Real” means the physical environment where no virtualization technology is used.

If all program execution results are the same, there is no problem executing any exercise scenario, hence eliminating any concerns regarding the container-based cyber range. Theoretically, the identification algorithm \( \phi \) can be defined as the inability to identify in which environment program \( A \) was executed. As shown in equations 1, 2, 3, and 4, the set of results of running the program in each environment is written as \( A^{\text{Real}}, A^{\text{HV}}, A^{\text{Host}}, \) and \( A^{\text{Container}} \), respectively.

\[
A^{\text{Real}} = \{ A | \phi(A^{\text{Real}}) = 1 \} \quad (1)
\]
\[
A^{\text{HV}} = \{ A | \phi(A^{\text{HV}}) = 1 \} \quad (2)
\]
\[
A^{\text{Host}} = \{ A | \phi(A^{\text{Host}}) = 1 \} \quad (3)
\]
\[
A^{\text{Container}} = \{ A | \phi(A^{\text{Container}}) = 1 \} \quad (4)
\]

Equation 1 is the set of results of running any program on the physical environment. Similarly, equation 2, 3, and 4 are the set of results of running any program on each virtual environment. By measuring these sets’ similarity, we can confirm the vulnerability reproducibility in the container-based cyber range. The relationship between each set is shown in Figure 5.

The similarity of a set can be measured by the Jaccard coefficient. For example, the calculation of the similarity between \( A^{\text{Real}} \) and \( A^{\text{Container}} \) is shown in equation 5.

\[
J(A^{\text{Real}}, A^{\text{Container}}) = \frac{|A^{\text{Real}} \cap A^{\text{Container}}|}{|A^{\text{Real}} \cup A^{\text{Container}}|} \quad (5)
\]

By measuring the similarity between \( A^{\text{Real}} \) and \( A^{\text{HV}}, A^{\text{Host}}, A^{\text{Container}} \) using the Jaccard coefficient, we can check vulnerability reproducibility in containerized virtualization, which can be used as an indicator for utilizing container-based cyber ranges.
B. EXPERIMENTAL ENVIRONMENT

In each environment of the real, HV, host, and container types, the results of scans by vulnerability scan tools used in cyber range exercises and attack experiments against vulnerabilities are comprehensively conducted, and the results are compared.

We use three vulnerable operating systems—Metasploitable2, Metasploitable3, and OwaspBWA—as the environment to reproduce the vulnerabilities. These are free public operating systems with many vulnerabilities intentionally created for the verification and learning of attacks and defense methods; they also contain many vulnerabilities that can be used in cyber range exercises [15]–[17].

To eliminate the differences between virtualization types as much as possible, we used Clonezilla, software that can backup and restore the entire file system. By retrieving the backup images from each vulnerable OS built on the physical environment and restoring them on the virtual environment, P2V (physical to virtual) was realized to create virtual machines in the HV-type and host-type environments.

For the container type environment, we used the docker import command, which can create an image from the file system by retrieving the configuration files on each OS running in the physical environment, except for /boot, /dev, /mnt, /proc, /sys, and /tmp, which are unnecessary for the container’s operation; we created a container that can reproduce the same environment. By manually executing various services after starting the container, it is possible to build an environment equivalent to other types, even though the startup process is different. In a manner of speaking, we were able to realize P2C (physical to container). In these environments, we conducted an inspection and attack experiments using a variety of tools.

The tools used for verification are shown in Table 3.

We then used tools that can inspect and exploit each area to conduct comprehensive experiments. The vulnerabilities detected by each tool are classified based on the common vulnerabilities and exposures (CVE) [18], and the number of vulnerabilities is compared. However, because the number of CVE detections is large, we counted each common weakness enumeration (CWE) [19], which classifies the vulnerabilities by type.

C. EXPERIMENTAL RESULTS

1) OpenVAS

OpenVAS (open vulnerability assessment scanner) is an open-source vulnerability assessment tool developed and supported by Greenbone Networks. It can detect a wide range of vulnerabilities such as software bugs, usage flaws, configuration, and so forth, and the corresponding CVEs can be checked using the latest DB. The results of the OpenVAS checks are shown in Table 4.

Experiments with OpenVAS were conducted via the network from OpenVAS, which was installed on a separately prepared physical terminal in the Metasploitable2, Metasploitable3, and OwaspBWA environments built with the real, HV, host, and container types respectively. We performed CVE-based detection in OpenVAS and aggregated the results for each corresponding CWE. In total, we were able to detect 782 vulnerabilities, and 748 of them were detected in the container environment.

In the OpenVAS inspection, the detection contents were consistent for many items. The physical environment and the HV-type and host-type environments matched entirely in terms of the number of detections. However, the container type environment had a small number of detections for some items.

The concordance rate calculated using only the OpenVAS test results was $J(A^{\text{Real}}, A^{\text{Container}}) = 0.9565$, and there were no CWE items that went undetected, confirming high reproducibility.

2) NMAP

Nmap (network mapper) is an open-source port scanner with a rich set of features such as OS and service version...
TABLE 4. Vulnerability detection results by Openvas for each virtualization type.

| CWE   | Name                        | Type               | Real | Host | Container |
|-------|-----------------------------|--------------------|------|------|-----------|
| 16    | Configuration              |                    | 1    | 2    | 2         |
| 18    | DEPRECATED- Source Code     |                    | 2    | 2    | 2         |
| 19    | Data Processing Errors      |                    | 1    | 1    | 1         |
| 20    | Improper Input Validation   |                    | 121  | 121  | 114       |
| 22    | Path Traversal              |                    | 12   | 12   | 12        |
| 59    | Link Following              |                    | 6    | 6    | 6         |
| 79    | Cross-site Scripting        |                    | 97   | 97   | 97        |
| 89    | SQL Injection               |                    | 16   | 16   | 16        |
| 93    | CRFI Injection              |                    | 3    | 3    | 3         |
| 94    | Code Injection              |                    | 15   | 15   | 15        |
| 119   | Buffer Overflow             |                    | 75   | 75   | 75        |
| 125   | Out-of-bounds Read          |                    | 21   | 21   | 21        |
| 134   | Use of External-Controlled |                    | 6    | 6    | 6         |
| 189   | Numeric Exceptions          |                    | 38   | 38   | 38        |
| 200   | Exposure of Sensitive Info  |                    | 4    | 4    | 4         |
| 234   | 7PK - Security Features     |                    | 14   | 14   | 14        |
| 235   | Credentials Management Errors|                  | 2    | 2    | 2         |
| 284   | Improper Access Control     |                    | 9    | 9    | 9         |
| 287   | Improper Authentication     |                    | 12   | 12   | 12        |
| 290   | Authentication Bypass by Spoofing |               | 1    | 1    | 1         |
| 300   | Channel Accessible to Non-Endpoint |       | 1    | 1    | 1         |
| 310   | Cryptographic Issues        |                    | 31   | 31   | 31        |
| 345   | Insufficient Verification of Data Authenticity | | 1 | 1 | 1 |
| 352   | Cross-Site Request Forgery  |                    | 12   | 12   | 12        |
| 362   | Race Condition              |                    | 8    | 8    | 8         |
| 384   | Session Fixation            |                    | 1    | 1    | 1         |
| 399   | Resource Management Errors  |                    | 70   | 70   | 70        |
| 415   | Double Free                 |                    | 1    | 1    | 1         |
| 416   | Use After Free              |                    | 3    | 3    | 3         |
| 428   | Unquoted Search Path or Element |               | 1    | 1    | 1         |
| 476   | NULL Pointer Dereference    |                    | 12   | 12   | 12        |
| 502   | Deserialization of Untrusted Data |              | 5    | 5    | 5         |
| 522   | Insufficiently Protected Credentials | | 3 | 3 | 3 |
| 552   | Files or Directories Accessible to External Parties | | 1 | 1 | 1 |
| 601   | Open Redirect               |                    | 4    | 4    | 4         |
| 640   | Weak Password Recovery Mechanism for Forgotten Password | | 1 | 1 | 1 |
| 732   | Incorrect Permission Assignment for Critical Resource | | 2 | 2 | 2 |
| 787   | Out-of-bounds Write         |                    | 1    | 1    | 1         |
| 835   | Infinite Loop               |                    | 4    | 4    | 4         |

TABLE 5. Portscan results by Nmap for each virtualization type.

| CWE   | Port | Service | Type               | Real | Host | Container |
|-------|------|---------|--------------------|------|------|-----------|
| 189,339 | 21   | ftp     |                    | 0    | 0    | 0         |
| 119,200 | 22   | ssh     |                    | 0    | 0    | 0         |
| 254,416 | 53   | domain  |                    | 0    | 0    | 0         |
| 79,287  | 80   | http    |                    | 0    | 0    | 0         |
| 399     | 111  | rpcbind |                    | 0    | 0    | 0         |
| 22,275  | 139  | netbios-ssn |          | 0    | 0    | 0         |
| 287     | 143  | imap    |                    | 0    | 0    | 0         |
| 269,400 | 445  | netbios-ssn |            | 0    | 0    | 0         |
| 264,290 | 631  | app     |                    | 0    | 0    | 0         |
| 94,119  | 1099 | java-mvi |                | 0    | 0    | 0         |
| 264     | 2049 | nfs     |                    | 0    | 0    | 0         |
| 22     | 2121 | ftp     |                    | 0    | 0    | 0         |
| 399     | 3306 | mysql   |                    | 0    | 0    | 0         |
| 134,189 | 3632 | distcc  |                    | 0    | 0    | 0         |
| 264,284 | 5432 | postgresql |             | 0    | 0    | 0         |
| 20,189  | 6667 | irc     |                    | 0    | 0    | 0         |
| 20,189  | 6697 | irc     |                    | 0    | 0    | 0         |
| 16      | 8009 | api13   |                    | 0    | 0    | 0         |
| 119,254 | 8080 | http    |                    | 0    | 0    | 0         |
| 22     | 8081 | http    |                    | 0    | 0    | 0         |
| 20,119  | 8100 | http    |                    | 0    | 0    | 0         |
| 399     | 8787 | drb     |                    | 0    | 0    | 0         |

Metasploitable2, Metasploitable3, and OwaspBWA environments, which were constructed using real, HV, host, and container types. The options were set to allow detection of port status, running services, and version. For each detection result, we checked the possible vulnerabilities based on the detected software versions.

In the results of the experiment with Nmap, we found an exact match in all environments.

Therefore, the port scan by Nmap did not distinguish whether the target environment was a physical or virtual environment, and there was no difference between the HV type or between the host and container types, so there was no difference in the reproducibility in each environment.

3) OWASP ZAP
ZAP (zed attack proxy) is an open-source web app vulnerability scanner provided by OWASP (open web application security project). ZAP can be used to check attack methods, so like Nmap, it should be handled with care. However, ZAP can be used to detect many vulnerabilities and check how to deal with them. Each server prepared as a vulnerable environment also has a web application environment, and web pages with various vulnerabilities can be checked. Table 6 shows the results of the ZAP experiment.

In inspecting ZAP, the inspection was performed in various network environments, such as from another virtual machine or host OS, as well as from another physical device. Because minor changes occurred in the results because of the different network environments, we checked the maximum number of detections in each detection result and tabulated the results for each corresponding CWE item.
Overall, the detection results show a similar trend in the number of detections for each detection category, but there were some cases where the number of detections was higher in the virtual environment than in the physical environment, such as “X-Content-Type-Options Header Missing and Remote OS Command Injection”. There was no difference in the presence or absence of detections in the same detection category although there were variations in the results, especially for those with more detections.

The similarity between the container-type virtualization and the physical environment, which was confirmed only by the ZAP test results, was $J(\text{Real}, \text{Container}) = 0.9540$. Compared with other virtual environments, $J(\text{HV}, \text{Container}) = 0.9851$ and $J(\text{Host}, \text{Container}) = 0.9877$, confirming high reproducibility.

4) InsightVM

InsightVM is a tool that is not only used for inspecting for vulnerabilities but also for providing a full range of support functions to deal with the detected vulnerabilities [21]. InsightVM was developed by Rapid7, Inc. and can be used as a comprehensive vulnerability management tool for companies and organizations, including penetration testing, because it can also check the existence of attack modules in the Metasploit framework (developed by Rapid7). The experiments using InsightVM are shown in the Table 7.

| CWE | Category | Type | Real | Hyper | Visor | Host | Container |
|-----|----------|------|------|-------|-------|------|-----------|
| 16  | X-Frame-Options Header Not Set | 6750 | 6818 | 6818 | 6791 | 6750 | 6750 |
| 16  | Cookie No HttpOnlyFlag | 44 | 60 | 60 | 27 | 44 | 44 |
| 16  | Cookie Without SameSite Attribute | 118 | 124 | 124 | 69 | 118 | 118 |
| 16  | X-Content-Type-Options Header Missing | 5585 | 5087 | 5087 | 5170 | 5585 | 5585 |
| 22  | Path Traversal | 31 | 31 | 31 | 7 | 31 | 31 |
| 78  | Remote OS Command Injection | 337 | 335 | 335 | 360 | 337 | 337 |
| 79  | Cross Site Scripting(Persistent) | 4 | 5 | 5 | 7 | 4 | 4 |
| 79  | Cross Site Scripting(Reflected) | 1143 | 1143 | 1143 | 1143 | 1143 | 1143 |
| 79  | Cross Site Scripting(DOM) | 60 | 13 | 13 | 12 | 60 | 60 |
| 89  | SQL Injection | 337 | 276 | 276 | 258 | 337 | 337 |
| 97  | Server Side Include | 1 | 1 | 1 | 1 | 1 | 1 |
| 98  | Remote File Inclusion | 211 | 211 | 211 | 208 | 211 | 211 |
| 200 | Application Error Disclosure | 266 | 266 | 266 | 250 | 266 | 266 |
| 200 | Information Disclosure - Debug Error Messages | 290 | 290 | 290 | 288 | 290 | 290 |
| 200 | Private IP Disclosure | 142 | 135 | 135 | 135 | 142 | 142 |
| 200 | Server Leaks Information via “X-Powered-By” | 512 | 525 | 525 | 444 | 512 | 512 |
| 215 | Miaccess Information Reak | 5 | 5 | 5 | 5 | 5 | 5 |
| 352 | Absence of Anti-CSRF Tokens | 6639 | 6489 | 6489 | 6519 | 6639 | 6639 |
| 472 | Parameter Tempering | 14 | 14 | 14 | 14 | 14 | 14 |
| 548 | Directory Browsing | 70 | 70 | 70 | 70 | 70 | 70 |
| 601 | External Redirect | 211 | 212 | 212 | 208 | 211 | 211 |
| 829 | Vulnerable JS Library | 17 | 17 | 17 | 17 | 17 | 17 |

| CWE | Name | Type | Real | Hyper | Visor | Host | Container |
|-----|------|------|------|-------|-------|------|-----------|
| 16  | Configuration | 4 | 4 | 4 | 4 | 4 | 4 |
| 17  | DEPRECATED: Code | 2 | 2 | 2 | 2 | 2 | 2 |
| 18  | DEPRECATED: Source Code | 1 | 1 | 1 | 1 | 1 | 1 |
| 19  | Data Processing Errors | 9 | 9 | 9 | 9 | 9 | 9 |
| 20  | Improper Input Validation | 101 | 101 | 101 | 101 | 101 | 101 |
| 22  | Path Traversal | 16 | 16 | 16 | 16 | 16 | 16 |
| 59  | Link Following | 7 | 7 | 7 | 7 | 7 | 7 |
| 74  | Injection | 3 | 3 | 3 | 3 | 3 | 3 |
| 77  | Command Injection | 2 | 2 | 2 | 2 | 2 | 2 |
| 78  | OS Command Injection | 4 | 4 | 4 | 4 | 4 | 4 |
| 79  | Cross-site Scripting | 80 | 80 | 80 | 80 | 80 | 80 |
| 89  | SQL Injection | 10 | 10 | 10 | 10 | 10 | 10 |
| 93  | CRLF Injection | 2 | 2 | 2 | 2 | 2 | 2 |
| 94  | Code Injection | 10 | 10 | 10 | 10 | 10 | 10 |
| 119 | Buffer Overflow | 96 | 96 | 96 | 96 | 96 | 96 |
| 125 | Out-of-bounds Read | 18 | 18 | 18 | 18 | 18 | 18 |
| 134 | Use of Externally-Controlled Format String | 4 | 4 | 4 | 4 | 4 | 4 |
| 189 | Numeric Errors | 48 | 48 | 48 | 48 | 48 | 48 |
| 190 | Integer Overflow or Wraparound | 11 | 11 | 11 | 11 | 11 | 11 |
| 199 | Information Management Errors | 1 | 1 | 1 | 1 | 1 | 1 |
| 200 | Exposure of Sensitive Information | 55 | 55 | 55 | 55 | 55 | 55 |
| 254 | JK - Security Features | 2 | 2 | 2 | 2 | 2 | 2 |
| 255 | Credential Management Errors | 4 | 4 | 4 | 4 | 4 | 4 |
| 264 | Permissions, Privileges, and Access Controls | 58 | 58 | 58 | 58 | 58 | 58 |
| 269 | Improper Privilege Management | 2 | 2 | 2 | 2 | 2 | 2 |
| 284 | Improper Access Control | 9 | 9 | 9 | 9 | 9 | 9 |
| 285 | Improper Authorization | 1 | 1 | 1 | 1 | 1 | 1 |
| 287 | Improper Authentication | 9 | 9 | 9 | 9 | 9 | 9 |
| 310 | Cryptographic Issues | 44 | 44 | 44 | 44 | 44 | 44 |
| 326 | Inadequate Encryption Strength | 1 | 1 | 1 | 1 | 1 | 1 |
| 327 | Use of a Broken or Risky Cryptographic Algorithm | 2 | 2 | 2 | 2 | 2 | 2 |
| 330 | Use of Insufficiently Random Values | 1 | 1 | 1 | 1 | 1 | 1 |
| 338 | PRNG | 1 | 1 | 1 | 1 | 1 | 1 |
| 352 | Cross-Site Request Forgery | 18 | 18 | 18 | 18 | 18 | 18 |
| 362 | Race Condition | 8 | 8 | 8 | 8 | 8 | 8 |
| 379 | Creation of Temporary File in Directory with Insecure Permissions | 1 | 1 | 1 | 1 | 1 | 1 |
| 399 | Resource Management Errors | 60 | 60 | 60 | 60 | 60 | 60 |
| 400 | Uncontrolled Resource Consumption | 3 | 3 | 3 | 3 | 3 | 3 |
| 415 | Double Free | 2 | 2 | 2 | 2 | 2 | 2 |
| 416 | Use After Free | 12 | 12 | 12 | 12 | 12 | 12 |
| 434 | Unrestricted Upload of File with Dangerous Type | 1 | 1 | 1 | 1 | 1 | 1 |
| 476 | NULL Pointer Dereference | 11 | 11 | 11 | 11 | 11 | 11 |
| 502 | Deserialization of Untrusted Data | 4 | 4 | 4 | 4 | 4 | 4 |
| 601 | Open Redirect | 3 | 3 | 3 | 3 | 3 | 3 |
| 640 | Weak Password Recovery Mechanism for Forgotten Password | 2 | 2 | 2 | 2 | 2 | 2 |
| 681 | Incorrect Conversion between Numeric Types | 1 | 1 | 1 | 1 | 1 | 1 |
| 732 | Incorrect Permission Assignment for Critical Resource | 1 | 1 | 1 | 1 | 1 | 1 |
| 754 | Improper Check for Unusual or Exceptional Conditions | 1 | 1 | 1 | 1 | 1 | 1 |
| 787 | Out-of-bounds Write | 4 | 4 | 4 | 4 | 4 | 4 |
| 863 | Incorrect Authorization | 1 | 1 | 1 | 1 | 1 | 1 |
| 918 | SSRF | 1 | 1 | 1 | 1 | 1 | 1 |
| 1188 | Insecure Default Initialization of Resource | 1 | 1 | 1 | 1 | 1 | 1 |
vulnerabilities were detected, the contents of all the detected vulnerabilities were consistent, and the vulnerability scores indicated independently by InsightVM were also the same.

![Image: Host types detected by InsightVM.](image)

**FIGURE 6.** Host types detected by InsightVM.

InsightVM also has an item to identify the host type. As shown in the Figure 6, the physical environment was displayed as bare metal, but all other virtualization types, including the container type, were displayed as virtual machine. The container type environment completely matched the other virtualization types.

Although the detection by InsightVM can distinguish between physical and virtual environments, there is no difference in the detection results by virtualization type, and containerized virtualization is the same environment as other virtualization types.

5) SkipFish

Skipfish is Google’s webapp vulnerability detection tool [22], [23]. By performing recursive crawls and dictionary-based probes, it can generate an interactive sitemap of the target site [22], [23]. Skipfish is particularly useful in determining whether a site is vulnerable to scripting or injection attacks [24]. The results of the skipfish scan are shown in Table 8.

Skipfish generates and scans URLs based on the keywords extracted from dictionaries and sites prepared in advance. Because Skipfish accesses a large number of non-existent URLs, we did not extract keywords, instead scanning each site four times. The results of the skipfish scans were the same for all scans, and the reproducibility was the same, regardless of the type of physical or virtual environment.

Although there are no specific CVEs or CWEs listed in the content of the detected issue types, we checked those for which the corresponding CWEs could be inferred from the content.

| CWE | Issue Types                      | Type   | Real | Hyper | Host | Container |
|-----|----------------------------------|--------|------|-------|------|-----------|
| 89  | Query injection vector           |        | 23   | 23    | 23   | 23        |
| 78  | Shell injection vector           |        | 15   | 15    | 15   | 15        |
| 19  | Server-side XML injection vector |        | 1    | 1     | 1    | 1         |
| 22  | Signature match detected         |        | 10   | 10    | 10   | 10        |
|     | Directory traversal / file inclusion possible | | 6    | 6     | 6    | 6         |
| 20  | Incorrect or missing charset     |        | 73   | 73    | 73   | 73        |
|     | Generic MMIB type                |        | 55   | 55    | 55   | 55        |
|     | External content embedded on a page |      | 14   | 14    | 14   | 14        |
| 79  | XSS vector in document body      |        | 1043 | 1043  | 1043 | 1043      |
|     | Incorrect caching directives     |        | 551  | 551   | 551  | 551       |
| 352 | HTML form with no apparent XSRF protection | | 1060 | 1060  | 1060 | 1060      |
|     | Directory listing restrictions bypassed | | 5    | 5     | 5    | 5         |
|     | Node should be a directory, detection error? | | 33   | 33    | 33   | 33        |
|     | Response varies randomly, skipping checks | | 22   | 22    | 22   | 22        |
|     | Limits exceeded, fetch suppressed |        | 241  | 241   | 241 | 241       |
|     | Resource fetch failed            |        | 144  | 144   | 144 | 144       |

6) Nikto

Nikto is a tool that performs vulnerability scans on web servers and middleware for multiple items, including dangerous files and programs, to check for software version checking misconfigurations and for the possible vulnerabilities caused by them [25]. The results of our experiments with Nikto are shown in Table 9.

| CWE | OSVDB | Example of detection                           | Type   | Real | Hyper | Host | Container |
|-----|-------|-----------------------------------------------|--------|------|-------|------|-----------|
| 48  | The /doc directory is browsable |        | 1    | 1    | 1    | 1    |
| 119 | CWE-1999-069 |                                |        | 4    | 4    | 4    | 4         |
| 284 | 576   | Vulnerability of weblogic |        | 2    | 2    | 2    | 2         |
| 630 | The web server may reveal its internal or real IP in the Location header | | 1    | 1    | 1    | 1    |
| 693 | 877   | HTTP TRACE method is active, suggesting the host is vulnerable to XST | | 1    | 1    | 1    | 1         |
| 3092| pgMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts | | 12   | 12   | 12   | 12    |
| 3253| pcache default file found      |        | 5    | 5    | 5    | 5    |
| 3268| Directory indexing found       |        | 19   | 19   | 19   | 19    |
| 119 | 3288 | Vulnerability of Abyss 1.03 |        | 1    | 1    | 1    | 1         |
| 12184| PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings | | 1    | 1    | 1    | 1    |
| 39272| favicon.ico file identifies this app/server as: owasp.org | | 1    | 1    | 1    | 1    |

Nikto performs dictionary-based vulnerability detection based on its criteria, and OSVDB (open source vulnerability database) [26]. However, because OSVDB is no longer in operation, we compiled the detection results corresponding
to CWE to the fullest extent possible. Although the number of items detected by Nikto was not significant, we were able to obtain a perfect match in all environments.

7) DEEP EXPLOIT
Deep Exploit is a fully automated penetration testing tool that uses deep reinforcement learning [27]. It identifies the state of the target port, executing a pinpoint exploit with the Metasploit module. Suppose we can confirm the success of an attack by using Deep Exploit in each environment of the cyber range. In that case, we can confirm the vulnerability reproducibility in each virtualization type and confirm no problem in executing the exercise scenario. The results of the Deep Exploit experiments are shown in Table 10.

| CWE | Exploit and Payload | Type |
|-----|---------------------|------|
| 78  | exploit/unix/ftp/vsftpd_234_backdoor - payload/cmd/unix/interact | Real |
| 77  | exploit/linux/smb/solarwinds_lcm_exec - payload/python/metpreter_bind_tcp | Hyper |
| 94  | exploit/solaris/telnet2/fuser - payload/cmd/unix/bind_perl | Visor |
| 119 | exploit/windows/antivirus/trendmicro_serverprotect_createbinding - payload/generic/shell_bind_tcp | Container |
| 119 | exploit/unix/ftp/proftpd_133c_backdoor - payload/cmd/unix/bind_perl | Real |
| 119 | exploit/mysql/mysql_yassl_getname - payload/linux/x86/shell/bind_ipv6_tcp_udp | Hyper |
| 119 | exploit/multi/windows/ncs_keybinding_exec - payload/cmd/unix/bind_perl | Visor |
| 119 | exploit/multi/mysql/yassl_getname - payload/linux/x86/shell/bind_ipv6_tcp_udp | Container |

The results show that the same exploit module successfully attacked the same vulnerability in all environments.

D. DISCUSSION
From the results of the experiments using each tool, the vulnerability detection items were aggregated by CWE classification, and the similarity between the real environment and each virtual environment was calculated based on the number of detections. The results are shown in Figure 7.

Because of the large number of items detected, the Top 10 items are shown. In many items, \( A^\text{Real} \) and \( A^\text{Container} \) showed a similarity of \( J(A^\text{Real}, A^\text{Container}) = 1 \), and the overall similarity was also very high at \( J(A^\text{Real}, A^\text{Container}) = 0.9602 \). When the similarity with other virtualization types was calculated, the similarity with \( A^\text{HV} \) was \( J(A^\text{HV}, A^\text{Container}) = 0.9907 \) and with \( A^\text{Host} \) was \( J(A^\text{Host}, A^\text{Container}) = 0.9931 \). Both values are very high, confirming that container-type virtualization has a sufficiently high vulnerability reproducibility compared with the other virtualization types.
Although we thought that many containerized environments have more trouble reproducing the same vulnerabilities and incidents than other environments, the similarity of the results obtained in our experiments were all much higher than expected, confirming that containerized virtualization has high vulnerability reproducibility compared with other virtualization types.

For vulnerabilities that could not be detected in the container environment, we could not find any essential fault intrinsic to the container-type virtualization. In all cases, we experienced that small configuration changes improve the experimental results to equalizes the initial differences because of the difference in default setting among virtualization types. Although the number of detections did not match in some instances of OpwnVAS and OWASP ZAP, we believe that further improvements can be made. These vulnerabilities can likely be reproduced by checking the status of the files and services to be detected and creating the same state. Some items were detected more frequently in the virtual environment than in the physical environment, suggesting that some of the differences in the results may be because of the essential differences in the environment, such as network configuration and drivers.

For each CWE item, the items detected by other types were always detected by the container type. In summary, for the vulnerabilities detected in each environment, the vulnerabilities classified as the same CWE can consistently be reproduced.

Concerns about the vulnerability reproducibility in the container-based cyber range were dispelled at a high level, and we confirmed that container-based cyber range systems can execute the same scenarios as those developed for the other conventional cyber range systems.
V. SCENARIO RANDOMIZATION

A. RANDOMIZATION METHODS

The experiments in section IV have confirmed the performance advantage of containerized virtualization and the high vulnerability reproducibility. However, this alone is not enough to make the container-based cyber range widely usable. To further solve the challenges of cyber ranges and encourage their widespread use, it is necessary to have a platform that can quickly provide many exercise scenarios [28].

To be able to provide many scenarios using container-based cyber ranges, we developed CyExec\textsuperscript{4}, which inherits the concept of CyExec, a Docker-based cyber range platform and implements a function that can automatically generate scenarios randomly [5], [6].

To add randomness, we refer to the concept of SecGen, which is a publicly available platform that provides many CTF exercise environments in a randomized manner. The main concept of SecGen is as follows [29]:

- Provides a randomizable, flexible, and generic method for security education that can be used in CTF and security lab exercises and simulations.
- Outputs a set of VMs, including server, client and network configurations, software and configuration vulnerabilities, randomizing their various components to create richer scenarios.
- Design and implement a specification for generating scenarios randomly.

These concepts can be used in a cyber range environment and are essential for providing students with many organized learning opportunities. Even in the cyber range, incorporating random components into scenarios can provide many scenarios, providing learning opportunities and improving educational effectiveness. However, SecGen provides a platform to automatically build a scenario in a randomized environment, but the ability to do this is based on Vagrant, which automatically builds a VirtualBox virtual machine environment. In cyber range environments that require a large number of virtual instances, this is likely to increase resource consumption and cause significant delays in startup and operation [30].

By utilizing containerized virtualization, the increase in resource consumption can be kept to a minimum and even if multiple environments are randomly generated, an exercise environment can be prepared with fewer hardware resources. In addition, by taking advantage of the high reproducibility of vulnerabilities to randomly generate scenarios, we can provide a large number of scenarios while greatly reducing the burden of preparing exercise environments and developing scenarios, hence preventing a reduction of educational effects because of scenario leakage and fraud.

B. DAG-BASED SCENARIO

To study the randomization of cyber range scenarios, we analyzed a common cyber range scenario. Figure 8 shows an example of a common cyber range scenario [31], [32].

The cyber range scenario has several milestone points within the overall scenario. The operations and actions that reach those milestones are a single scenario that deals with individual attack methods, similar to CTF and lab work. The overall scenario can be thought of as a composite scenario in which each milestone is connected through a separate scenario, ultimately forming the entire incident.

There are several possible scenarios for reaching each milestone. For example, the difference between a malicious file or malware being sent via e-mail or downloaded via a rogue website and the difference between whether an attacker exploits a configuration issue in the operating system or a vulnerability in software in an attempt to take away administrative rights. Some of these scenarios could result in the same outcome, even if the attacker uses different means.

Our approach is as follows: if we could fix some milestones and incorporate random elements into the scenarios, we could create a random cyber range scenario. Figure 9 shows an image of a cyber range scenario using our approach and that incorporates randomization.

The randomized cyber range scenario takes the form of a graph, with the milestone considered to be the same state as the vertex and the scenario directed to the next milestone as the edge. Because the attack is directed toward the final target, the scenario does not consider the possibility of going back to a previous milestone or looping back to the exact location. Therefore, a cyber range scenario that takes randomness into account forms a directed acyclic graph (DAG) [32], [33].

When developing a scenario, it is important to ensure that the overall scenario is a DAG and to consider multiple scenarios heading toward each milestone. This method can provide a scenario pattern for the total number of paths in the graph. Multiple random scenarios with different paths but the same objectives allow participants to experience security incidents in a variety of situations.
C. DEVELOPMENT AND IMPLEMENTATION

The image used to launch a virtual instance in Docker can be a simple text file called Dockerfile, which describes the entire environment, configuration, and behavior [34], [35]. In SecGen, many platforms such as Ruby, Vagrant, Puppet, and VirtualBox are used to build the VM. These environments are common and free of charge. However, because of the large number of platforms involved, they may not work correctly for various reasons, such as compatibility with different versions or modules. CyExec* is designed to minimize the impact of non-standard Docker features and to make it easy for anyone to install and develop scenarios.

CyExec* uses numerous Dockerfiles to build containers that can run various attack and defense scenarios, prepare multiple environments, and use Docker-compose to automate the construction of network environments with multiple Dockerfiles. Docker-compose can start an environment with multiple container images, including network configuration. Providing multiple Docker-compose files enables the randomization or change in the environment by specifying the Dockerfile to be used on startup [36], [37].

Figure 10 shows an image of CyExec* in operation.

CyExec*, like SecGen, consists of two stages. Stage 1 is based on the default scenario and defines the environmental elements, such as the number of environments to be generated, the number of students to be trained, and so forth. Based on the contents of stage 1, in stage 2, Docker-compose specifies the Dockerfile required to run multiple scenarios, launches the containers, and forms the scenarios. By separating them into stages 1 and 2, the environmental and scenario elements can be clearly separated and quickly developed or modified independently.

D. DEFAULT SCENARIO

CyExec* provides a default scenario with a base system configuration. Additional randomness can be accommodated by modifying some of the default scenarios. The default scenario environment can also be used to learn attack and defense techniques or develop scenarios.

The default scenario is automatically generated by Dockerfile and Docker-compose, which are highly portable; any environment running Docker can build an equivalent environment. There is no need to develop complex programs or modules for scenario development. Scenario development can be handled by adding a Dockerfile and editing docker-compose.yml file. Figure 11 shows the configuration of the default scenario.

All virtual instances are implemented in containers but behave in the same way as HV-type or host-type virtual machines. However, this default scenario was not prepared...
with any specific attacks or vulnerabilities in mind. By adding new containers of devices and services to this environment, we can expand the types of randomness by changing the attack methods and forensics to provide more scenarios. Figure 12 shows an example of a screen connected to this container.

This provides an environment similar to ordinary terminal operations, such as accessing a web server or checking the network. However, because it runs in a container, it can be expected to start up quickly and run lightly. The system can be used for various purposes by adding various tools and deleting unnecessary ones as needed. Scenarios can be developed by intentionally making them vulnerable or by running programs that attack other containers.

E. ADDING RANDOMNESS

To test for randomness, we added a randomized scenario with multiple attack methods to the default scenario. If multiple attack results against a Metasploitable2 vulnerability reach the same milestone, for example, accessing root privileges, the attack can be randomized. Table 12 shows examples of vulnerable applications running on Metasploitable2 and exploit modules that lead to the same milestone.

| Vulnerable Applications | Equivalent exploit modules          |
|------------------------|-------------------------------------|
| vsftpd                 | exploit/unix/ftp/vsftpd_234_backdoor|
| php                    | exploit/multi/http/php_arg_injection|
| Samba                  | exploit/multi/samba/usermap_script   |
| PostgreSQL             | exploit/linux/postgres/postgres_payload|
| UnrealIRCd             | exploit/unix/irc/unreal_ircd_3281_backdoor|
| distccd                | exploit/unix/misc/distccd_exec       |
| Ruby DRb RMI           | exploit/unix/misc/drbrm_remotecodeexec|

These exploit modules can be used to exploit the vulnerabilities in software running on the target server, allowing arbitrary commands to be executed remotely without the need for proper authentication. Because it is possible to execute commands with root privileges, it is possible to perform various actions, such as obtaining information to which the attacker does not have access or tampering with files, leading to the next scenario [38], [39]. If different exploit modules can create the same state, they can be used interchangeably to add randomness to the scenario.

To verify that CyExec+ can execute random scenarios, we added an exploit that sends malware to the default scenario environment, verifying the execution of a random scenario, as shown in Figure 13.

We can specify the Kali Linux container to run a separate exploit module for each and make the Metasploitable2 container capable of executing arbitrary commands with root privileges. After that, we can specify commands in the Dockerfile to send files that mimic the malware, here downloaded from another server via HTTP and the other to be sent via FTP. CyExec+ was designed to generate Docker-compose.yml, which selects a Dockerfile to launch an environment where these actions are performed in succession.

In this scenario, we investigate how the intrusion occurred and how the malware was delivered. In other words, the content of the exercise follows the path of the DAG-based
scenario and investigates it. In this exercise, we were able to generate four scenarios with $2 \times 2$ paths by using a simple scenario with pseudo-malware. More scenarios can be added using various routes, such as using a program that behaves like malware or propagating the attack to other devices.

F. RANDOMIZATION COMPARISON
In order to confirm the performance of CyExec*, we examined the comparison with other exercise platforms. Since CyExec* leverages the performance of containerized virtualization with Docker, we investigated SEED Labs and Labtainer, which also use container-type virtualization. These use Docker as well as CyExec*, and the scenario is fixed. However, they use Docker as well as CyExec*, and the scenarios are fixed. Therefore, many performance differences such as startup and resource consumption are not expected to appear. Also, it is not suitable to compare the performance of scenario randomization, but many scenarios published in these platforms may be useful for extending CyExec* scenario randomization.

We considered a comparison with SecGen, which is not container-type virtualization but offers randomized scenarios; SecGen is an exercise platform that automatically builds Virtualbox virtual machines in a randomized manner through automatic configuration using Ruby and Vagrant.

CyExec* leverages the performance of Docker-based container-type virtualization, but SecGen is based on Virtualbox. While SecGen is based on Virtualbox and utilizes Vagrant and Ruby to build an exercise environment, it is a platform based on the concept of randomizing the same scenario. Still, its implementation method is very different from CyExec*.

The default scenario of CyExec* includes six virtual instances; it uses a large image, such as kali Linux, while the default scenario of SecGen is a single virtual machine image, including a web server. However, CyExec* is an environment that can be used for general-purpose exercises and covers the functionality that the default scenario of SecGen has, including the webserver. Therefore, CyExec* uses more virtual instances than SecGen, but we compared them as they are. Figure 14 shows the comparison results of resource consumption between CyExec* and SecGen.

When we ran the default scenario of CyExec* and compared the resource consumption with the default scenario of SecGen, the memory usage was about 1/3, CPU usage was 1/4, and storage usage was only 1/10 (Compare by the amount of increase), confirming the high performance of CyExec*.

In comparison with the other virtualization type environment, we confirmed that the environment built with container-type virtualization was superior, showing that it could be used to run more complex environments with more virtual instances at the same time.

Figure 15 also shows a comparison of the startup time and the storage space used when the default scenarios are invoked. Both CyExec* and SecGen download the required image on the first run. SecGen took 39 min 35 sec to finally become usable because of the various configurations that took place after the download was complete. CyExec* took a long time to download the image because of the large size of the image used, but it eventually outperformed SecGen by 31 min 28 sec before the environment was booted and ready to use, demonstrating the speed of the time taken to boot.

After exiting the environment at the first boot, we started the same environment again. In this case, because no image download was required, CyExec* started the environment immediately and was ready to use in a very short time of 22 sec. SecGen also did not require an image download but required the same configuration work for the virtual machine as the first time, taking 6 min 19 sec to boot. After that, we booted and checked the same scenario again, and the
results were almost the same, confirming the high performance of CyExec* using containers.

Figure 16 then shows the comparison when a random scenario derived from the default scenario is launched.

CyExec* downloaded and started only the containers necessary for the changes and was ready to use in 2 min 12 sec. The storage capacity increased with the changes, but the rest of the image was the same as the default image, so it consumed less space. We booted the SecGen in a condition so that the number of virtual instances to be launched would not change; here, a new virtual machine was created that used almost the same amount of space. After that, three new derivative random scenarios launched, and in both cases, the performance of CyExec* outperformed SecGen.

This comparison was conducted with the condition that CyExec* has to launch more virtual instances than SecGen. However, the results show that CyExec* is still significantly superior not only in launching the default scenario repeatedly but also in launching random scenarios derived from the default scenario, confirming its performance as a system that maximizes the performance of a container-type virtualization.

G. DISCUSSION
To make effective use of the performance of container-type virtualization and the ability to reproduce scenarios, we considered a cyber range system that can provide a large number of scenarios randomly, hence developing CyExec*. By using DAG as the scenario development method, we can easily consider changes and additions to the scenarios, developing scenarios that can be used for various learning purposes and levels. In addition, by making it possible to provide more scenarios, we can solve the problem of introducing a cyber range and reduce the risk of scenario leakage.

In the performance comparison with SecGen, CyExec* showed significant values of about 1/3 in memory consumption, 1/4 in CPU utilization, and 1/10 in storage consumption. This is a result of CyExec* making the most of the advantages of containers and greatly outperforming the performance of virtual machines based on host-type virtualization.

This result may vary depending on the contents of the scenario, the environment in which it runs, and the tools used. However, the speed of the startup time and the small number of running processes are intrinsic to container-type virtualization compared with other virtualization types. Considering the high vulnerability reproducibility, container-type virtualization is the most effective in cyber range systems.

However, although we were able to confirm that CyExec* can randomly generate scenarios and operate them efficiently, we were not able to use it in actual exercises or measure its educational effects. In the future, it will be necessary to construct a large-scale environment and generate random scenarios with more complex routes to conduct exercises and measure the educational effects.

VI. TOWARD A PRACTICAL CYBER RANGE
Cyber range consists of an exercise scenario and the construction of an environment to reproduce it. However, not only that, various capabilities have been implemented or researched for more effective exercise execution and operation management.

Table 13 details the features considered in existing cyber ranges and compares how they are addressed in commercial products and academic research [1].

In the cyber range products, paid virtualization software such as VMware and Hyper-V and actual security appliance products are used to build environments and conduct
TABLE 13. Various capabilities and response status of cyber ranges.

| Capabilities | Details | Products /Business | Research /Academic |
|--------------|---------|--------------------|--------------------|
| Scenarios    | The execution environment as well as the storyline that indicates the execution steps of a training exercise. | Single/multiple scenarios developed by experts Not open to public | Many single scenarios Published and shared |
| Monitoring   | Methods and tools for real-time monitoring of exercises. The methods teachers employ, the tools they use, and the layers they monitor are further categorized. | P/W, IDS/IPS, SIEM products, that are actually in use | Open source F/W, IDS/IPS, SIEM, and Log Monitoring Tools etc. |
| Teaming      | Teaming includes individuals and groups that design, develop, manage, and participate in cybersecurity exercises and tests. | Many teams for different roles Just like an actual incident response | Mainly Red Team and Blue Team Focus on individual exercises |
| Learning     | Capabilities to monitor learning and understanding, such as scenario progress and automatic grading by behavior. | Exercises for different purposes and levels. Not much need to manage learning status | Linkage with learning status management and LMS Consider a systematic curriculum |
| Management   | Allocation of computational and other resources required for conducting a cyber security exercise, or a test, and the overall management of the cyber range. | Various useful functions Many functions for instructors are also available | Dependent on virtualization software and tools. Some have original management functions |
| Security     | Security of the execution environment and the impact of misuse of the tools used. | Strictly enforced through network isolation and pledges during exercises | Need to manage virtualization software and various tools used Provide legal and ethical education to students |

realistic and gradual security incident response exercises to train security professionals.

By effectively utilizing open source and publicly available tools and platforms, the academic cyber range provides an inexpensive and easily deployable exercise environment, and conducts exercises of various objectives and levels, focusing on single scenarios that allow users to experience attack methods and vulnerabilities. Various researches and developments are being conducted to address the functional shortcomings in the cyber range for academics.

Scenarios, monitoring, and teaming are related to the exercise environment and content, and various research and development efforts have been made, especially for scenarios.

CyExec* utilizes the capabilities of containerized virtualization to provide multiple randomized scenarios for realistic security incident response exercises and has enhanced the scenario aspect of the system.

Learning and management are not directly related to the content of the exercise, but are necessary for operation and management, and are essential for the spread of cyber ranges.

The security of the cyber range is also important. The programs used to reproduce attacks and the management of the security of the cyber range environment must be considered in academic environments that use more open-source software and tools than products.

In particular, when the platform is open to the public, it is necessary to ensure that the exercise environment is not affected by unintended settings or vulnerabilities of the platform and that outside parties do not exploit it. In addition, it is necessary to check the operational security as well, for example, to prevent students from misusing the knowledge and skills acquired in the exercise.

More research and development and validation are needed to enhance the various features of the cyber range and make it a practical exercise environment. Especially in the academic, which is the main target of CyExec*, many features are limited compared to cyber range products.

Suppose the research in this paper can facilitate replacing existing cyber range environments with container-type virtualization as a unified platform. In that case, it can be expected to develop into an academic cyber range with appropriate scalability and the potential to add new features while leveraging the results of various existing researches.

In the future, there will be an increase in the need for new types of exercises, such as online learning, which have not been considered in the cyber range. We believe that the lightweight execution environment and high portability of container-type virtualization can be the best solution to minimize the burden on constructing the environment on the cloud and the connection terminals. We will continue our research and development of CyExec* considering the possibility of going online.

VII. CONCLUSION

Cyber ranges, which are used in information security training, have a wide range of needs, not only for training professionals, but also for general system users and as an introduction into the educational curriculum of higher education institutions because it provides a highly effective educational environment that faithfully reproduces real-world system environments and security incidents. However, there are issues such as high implementation and maintenance costs, difficulties in scenario development, and reduced educational effectiveness due to content leakage.

To promote the widespread use of cyber ranges, we have been promoting CyExec, a cyber range ecosystem using a container-type virtualization. Although container-type virtualization and its lightweight technology have been attracting attention as a scalable technology for cyber ranges, its use has been limited because there are only a few examples of a container-based cyber range, and its
performance advantages and vulnerability reproducibility were not clear.

Therefore, we conducted exhaustive experiments, here assuming a cyber range environment to confirm the superiority of containers compared with other virtualization types and the vulnerability reproducibility. As a result, in this experiment, we confirmed that the memory consumption was reduced to about 1/2 and the storage consumption to about 1/60. Although there was no significant difference in the CPU utilization, there was a significant difference in the startup and actual operation of the environment, confirming the superiority of container-type virtualization because of its features such as fast startup and minimal process operation.

In terms of vulnerability reproducibility, the results show a similarity of \( J(A_{\text{Real}}, A_{\text{Container}}) = 0.9602 \) with the physical environment. In comparison with other virtualization types, the results are very high, \( J(A_{\text{HV}}, A_{\text{Container}}) = 0.9907 \) with the HV type and \( J(A_{\text{Host}}, A_{\text{Container}}) = 0.9931 \) with the host type, confirming that container-type virtualization can reproduce vulnerabilities and incidents as well as other virtualization types.

Based on these results, we developed CyExec\(^*\), which implements a DAG-based scenario randomization technique on CyExec. By using CyExec\(^*\), we can provide multiple scenarios with the same learning objective, hence increasing the efficiency, hence eliminating the concern of high system load because of the increase of scenarios. Compared with SecGen, a CTF random scenario generator using the host-type virtualization, Cyexec\(^*\) showed the advantage of 1/3 memory consumption, 1/4 CPU consumption, and 1/10 storage consumption. This result indicates that SecGen is at least three times more capable of reproducing scenarios than the conventional virtual machine-based cyber range environment and can run more complex environments simultaneously.

In the current paper, we developed a default scenario as a base and verified randomization. More advanced scenarios can be developed for use with many learning levels. CyExec\(^*\) is a realistic means of information security education that can be used in many situations. In future studies, we will test CyExec\(^*\) in actual exercises, measure its educational effects, and release it to the public after confirming its scalability and scenarios to promote its spread as an inexpensive and easy-to-install cyber range ecosystem.

REFERENCES

[1] M. M. Yamin, B. Katt, and V. Gkioulous, “Cyber ranges and security testbeds: Scenarios, functions, tools and architecture,” Comput. Secur., vol. 88, Jan. 2020, Art. no. 101636.

[2] D. Fenton, T. Traylor, G. Hokanson, and J. Straub, “Integrating cyber range technologies and certification programs to improve cybersecurity training programs,” in The Challenges of the Digital Transformation in Education, M. E. Auer and T. Tsiatsos, Eds. Cham, Switzerland: Springer, 2019, pp. 632–643.

[3] K. E. Stewart, J. Humphries, and T. Andel, “Developing a virtualization platform for courses in networking, systems administration and cyber security education,” in Proc. SpringSim, 2009, pp. 1–7.

[4] R. Beuran, T. Inoue, Y. Tan, and Y. Shinoda, “Realistic cybersecurity training via scenario progression management,” in Proc. IEEE Eur. Symp. Secure Priv. Workshops, Jun. 2019, pp. 67–71.

[5] R. Nakata and A. Otsuka, “CyExec*: Automatic generation of randomized cyber range scenarios,” in Proc. 7th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP), Setúbal, Portugal: Science and Technology Publications, 2021, pp. 226–236.

[6] N. Maki, R. Nakata, S. Toyoda, Y. Kasai, S. Shinh, and Y. Seto, “An effective cybersecurity exercises platform CyExec and its training contents,” Int. J. Inf. Educ. Technol., vol. 10, no. 3, pp. 215–221, 2020.

[7] R. Beuran, C. Pham, D. Tang, K.-I. Chinen, Y. Tan, and Y. Shinoda, “CyTrONE: An integrated cybersecurity training framework,” in Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy, Setúbal, Portugal: Science and Technology Publications, 2017, pp. 157–166.

[8] G. Erdogan, A. Romero, N. Zazzeri, A. Žitnik, M. Basile, G. Aprile, M. Osório, C. Pani, and I. Kechaoglou, “Developing cyber-risk centric courses and training material for cyber ranges: A systematic approach,” in Proc. 7th Int. Conf. Inf. Syst. Secur. Privacy, Setúbal, Portugal: Science and Technology Publications, 2021, pp. 702–713.

[9] J. Vykopal, R. Ošlejšek, P. Čeleda, M. Vživýř, and D. Tovarnák, “KYPO cyber range: Design and use cases,” in Proc. 12th Int. Conf. Softw. Techn. (ICSOFT), vol. 1, J. Cardoso, J. Cardoso, L. Maciaszek, L. Maciaszek, M. van Sinderen, and E. Cabello, Eds. Madrid, Spain: SciTePress, 2017, pp. 310–321.

[10] C. E. Irvine, M. F. Thompson, and J. Khosalim. (2017). Labtainers: A Framework for Parameterized Cybersecurity Labs Using Containers. [Online]. Available: https://calounh.nps.edu/handle/10945/56211

[11] M. A. Babar and B. Ramsey, “Understanding container isolation mechanisms for building security-sensitive private cloud,” Univ. Adelaide, Adelaide, SA, Australia, Tech. Rep., Jan. 2017.

[12] D. K. Damodaran, B. R. Mohan, M. S. Vassudevan, and D. Naik, “Performance evaluation of VMware and VirtualBox,” in Proc. IPSICS, vol. 29, Singapore: IACISIT Press, 2012, doi: 10.13140/RG.2.2.3404.85769.

[13] J. Horalek, J. Matyska, and V. Sobeslav, “Performance comparison of selected virtualization platforms,” in Proc. IEEE 14th Int. Symp. Comput. Intel. Informat. (CINTI), Nov. 2013, pp. 327–332.

[14] G. Nakagawa and S. Okawa, “Behavior-based memory resource management for container-based virtualization,” in Proc. 4th Int. Conf. Appl. Comput. Inform. Technol., 3rd Int. Conf. Comput. Sci., Intell. Appl. Informat., 1st Int. Conf Big Data, Cloud Comput., Data Sci. Eng. (ACIT-CSI-BDC), Dec. 2016, pp. 213–217.

[15] Metasploitable 2 | Metasploitable Documentation. (Accessed: Apr. 30, 2021). [Online]. Available: https://docs.rapid7.com/metasploit/metasploitable-2/

[16] (Accessed: Apr. 30, 2021). GitHub—Rapid7/Metasploitable3: Metasploitable is a VM That is Built From the Ground Up With a Large Amount of Security Vulnerabilities. [Online]. Available: https://github.com/rapid7/metasploitable3

[17] OWASP Broken Web Applications. (Accessed: Apr. 30, 2021). [Online]. Available: https://owasp.org/www-project-broken-web-applications/

[18] CVE—Documents and Guidance. (Accessed: Apr. 30, 2021). [Online]. Available: https://cve.mitre.org/about/documents.html

[19] CWE—About—CWE Overview. (Accessed: Apr. 30, 2021). [Online]. Available: https://cwe.mitre.org/about/index.html

[20] NMAP: The Network Mapper—Free Security Scanner. (Accessed: Apr. 30, 2021). [Online]. Available: https://nmap.org/

[21] (Accessed: Apr. 30, 2021). Vulnerability Management Tools | InsightIVM | Rapid7. [Online]. Available: https://www.rapid7.com/products/insightvm/

[22] R. Mohammed, “Assessment of web scanner tools,” Int. J. Comput. Appl., vol. 133, no. 5, pp. 1–4, Jan. 2016.

[23] Y. makino and V. Klyuiev, “Evaluation of web vulnerability scanners,” in Proc. IEEE 8th Int. Conf. Intell. Data Acquisition Adv. Comput. Syst., Technol. Appl. (IDAACS), vol. 1, 2015, pp. 399–402, doi: 10.1109/IDAACS.2015.7340766.

[24] I. M. A. Lakhan, Web Penetration Testing With Kali Linux. Birmingham, U.K.: Packt, 2013.

[25] (Accessed: Apr. 30, 2021). Nikto: A Practical Website Vulnerability Scanner. [Online]. Available: https://securitytrails.com/blog/nikto-website-vulnerability-scanner

[26] J. Kouns. (Jun. 2008). Open Source Vulnerability Database. Open Source Business Resource. [Online]. Available: http://timreview.ca/article/155
RYOTARO NAKATA received the M.Tech. degree in information systems from the Advanced Institute of Industrial Technology, Osaka, Japan, in 1996. He received the B.E. and M.E. degrees from Osaka University, in 1989 and 1991, respectively, and the Ph.D. degree from The University of Tokyo, in 2002. In 2002, he was a Postdoctoral Fellow and a Cooperative Researcher with The University of Tokyo. In 2005, he was with the National Institute of Advanced Industrial Science and Technology (AIST), where he served as a Leader of Research Security Fundamentals, from 2006 to 2010. From 2007 to 2014, he was a Visiting Professor at the Research and Development Initiative, Chuo University. Since 2017, he has been a Professor with the Graduate School of Information Security, Institute of Information Security.

AKIRA OTSUKA (Member, IEEE) was born in Osaka, Japan, in 1966. He received the B.E. and M.E. degrees from Osaka University, in 1989 and 1991, respectively, and the Ph.D. degree from The University of Tokyo, in 2002. In 2002, he was a Postdoctoral Fellow and a Cooperative Researcher with The University of Tokyo. In 2005, he was with the National Institute of Advanced Industrial Science and Technology (AIST), where he serves as a Leader of Research Security Fundamentals, from 2006 to 2010. From 2007 to 2014, he was a Visiting Professor at the Research and Development Initiative, Chuo University. Since 2017, he has been a Professor with the Graduate School of Information Security, Institute of Information Security.

* * *

[27] (Accessed: May 1, 2021). Machine_Learning_Security/Deepexploit at Master · 13o-BBR-BBQ/Machine_Learning_Security · Github. [Online]. Available: https://github.com/13o-bbr-bbq/machine_learning_security/tree/master/DeepExploit

[28] T. Saito, S. Yashiro, K. Tanabe, and Y. Saito, “A proposal and the evaluation of a hands-on training system for cyber security,” in Advances on Broadband and Wireless Computing, Communication and Applications, L. Barolli, F.-Y. Leu, T. Enokido, and H.-C. Chen, Eds. Cham, Switzerland: Springer, 2019, pp. 339–349.

[29] Z. C. Schreuders, T. Shaw, M. Shan-A-Khuda, G. Ravichandran, J. Keighley, and M. Ordean, “Security scenario generator (SecGen): A framework for generating randomly vulnerable rich-scenario VMs for learning computer security and hosting CTF events,” in Proc. USENIX Workshop Adv. Secur. Educ. (ASE). Vancouver, BC, Canada: USENIX Association, Aug. 2017, pp. 1–10. [Online]. Available: https://www.usenix.org/conference/ase17/workshop-program/presentation/schreuders

[30] V. Zenk and M. Malmström, “A comparative study of Docker and Vagrant regarding performance on machine level provisioning,” Int. J. Pure Appl. Math., vol. 117, no. 7, pp. 175–184, 2017.

[31] V.-V. Patriiciu and A. C. Furtuna, “Guide for designing cyber security exercises,” in Proc. 8th WSEAS Int. Conf. E-Activities Inf. Secur. Privacy (E-ACTIVITIES). Stevens Point, WI, USA: World Scientific and Engineering Academy and Society, 2009, pp. 172–177.

[32] W. Ahn, M. Chung, B.-G. Min, and J. Seo, “Development of cyber-attack scenarios for nuclear power plants using scenario graphs,” Int. J. Distrib. Sensor Netw., vol. 11, no. 9, Sep. 2015, Art. no. 836258, doi: 10.1155/2015/836258.

[33] Z. Huang, C.-C. Shen, S. Doshiy, N. Thomasy, and H. Duong, “Difficulty-level metric for cyber security training,” in Proc. IEEE Int. Multi-Disciplinary Conf. Cognit. Methods Situation Awareness Decis., Mar. 2015, pp. 172–178.

[34] D. Nüst, V. Sochat, B. Marwick, S. J. Eglen, T. Head, T. Hirst, and B. D. Evans, “Ten simple rules for writing Dockerfiles for reproducible data science,” PLOS Comput. Biol., vol. 16, no. 11, pp. 1–24, Nov. 2020, doi: 10.1371/journal.pcbi.1008316.

[35] I. Emsley and D. De Roure, “A framework for the preservation of a Docker container,” in Proc. 12th Int. Digit. Curation Conf. (IDCC17), Edinburgh, U.K., Feb. 2017.

[36] L. Merel and J. Pavlik, “The comparison of container orchestrators,” in Proc. 3rd Int. Congr. Inf. Commun. Technol., X.-S. Yang, S. Sherratt, N. Dey, and A. Joshi, Eds. Singapore: Springer, 2019, pp. 677–685.

[37] A. Freeman, Docker Compose. Berkeley, CA, USA: Apress, 2017, pp. 97–117.

[38] F. Holik, J. Horalek, O. Marik, S. Neradova, and S. Zitta, “Effective penetration testing with metasploit framework and methodologies,” in Proc. IEEE 15th Int. Symp. Comput. Intell. Informat. (CINTI), Nov. 2014, pp. 237–242.

[39] S. Raj and N. K. Walia, “A study on metasploit framework: A pen-testing tool,” in Proc. Int. Conf. Comput. Perform. Eval. (ComPE), Jul. 2020, pp. 296–302.