Journal of Cybersecurity and Privacy

MDPI

Article

GoibhniUWE: A Lightweight and Modular Container-Based Cyber Range

Alan Mills *, Jonathan White and Phil Legg

School of Computing and Creative Technologies, University of the West of England, Bristol BS16 1QY, UK; jonathan6.white@uwe.ac.uk (J.W.); phil.legg@uwe.ac.uk (P.L.)
* Correspondence: alan.mills@uwe.ac.uk

Abstract: Cyberattacks are rapidly evolving both in terms of techniques and frequency, from low-level attacks through to sophisticated Advanced Persistent Threats (APTs). There is a need to consider how testbed environments such as cyber ranges can be readily deployed to improve the examination of attack characteristics, as well as the assessment of defences. Whilst cyber ranges are not new, they can often be computationally expensive, require an extensive setup and configuration, or may not provide full support for areas such as logging or ongoing learning. In this paper, we propose GoibhniUWE, a container-based cyber range that provides a flexible platform for investigating the full lifecycle of a cyberattack. Adopting a modular approach, users can seamlessly switch out existing, containerised vulnerable services and deploying multiple different services at once, allowing for the creation of complex and realistic deployments. The range is fully instrumented with logging capabilities from a variety of sources including Intrusion Detection Systems (IDSs), service logging, and network traffic captures. To demonstrate the effectiveness of our approach, we deploy the GoibhniUWE range under multiple conditions to simulate various vulnerable environments, reporting on and comparing key metrics such as CPU and memory usage. We simulate complex attacks which span multiple services and networks, with logging at multiple levels, modelling an Advanced Persistent Threat (APT) and their associated Tactics, Techniques, and Procedures (TTPs). We find that even under continuous, active, and targeted deployment, GoibhniUWE averaged a CPU usage of less than 50%, in an environment using four single-core processors, and memory usage of less than 4.5 GB.

Keywords: containerisation; cyber range; vulnerability analysis; traffic analysis

Citation: Mills, A.; White, J.; Legg, P. GoibhniUWE: A Lightweight and Modular Container-Based Cyber Range. *J. Cybersecur. Priv.* **2024**, *4*, 615–628. https://doi.org/10.3390/jcp4030029

Academic Editor: Danda B. Rawat

Received: 20 June 2024
Revised: 12 August 2024
Accepted: 21 August 2024
Published: 24 August 2024

1. Introduction

As cyber threats continue to evolve, having adaptive and scalable experimentation platforms that are both reliable and realistic is crucial for successful research and education on the nature of these threats. Given the nature of experiments when it comes to offensive and defensive cyber security assessment, it is not feasible to conduct such work on real-world production systems. At the other end of the spectrum, there may well be characteristics that we wish to study or examine that cannot be fully satisfied with a small scale set up such as a single Virtual Machine (VM) running a set of vulnerable applications. This has given rise to the idea of a 'cyber range', and much like the concept of a shooting range to practice and hone a skill, the cyber range sets out to provide a similar platform for refining cyber security skills in controlled scenarios before doing so in a live environment.

In recent years, much focus has been concentrated around the creation of VMs for providing suitable cyber security exercises. This has been popularised further by the introduction of cloud services such as TryHackMe and HackTheBox for deploying VMs. Cloud providers such as Azure and AWS also allow for the creation of multiple machines to be networked together; however, this comes at a cost and is not trivial to configure. A notable project was developed by Chris Long, Senior Security Analyst of Netflix, who developed the open-source project, DetectionLab [1]. This provides a set of VMs that are

pre-configured with connectivity and vulnerable appliances, as well as having built-in defensive tools, such as Security Information and Event Management (SIEM) monitoring, using tools like Splunk and the Threat Hunting plug-ins for this. Whilst this works well to deploy four VMs, this can be a time-consuming and resource-intensive deployment, requiring at least 16 GB of RAM and 55 GB of free hard disk space to run locally. This platform is no longer supported, highlighting the issues around ongoing support and maintenance for such systems. A similar example is the Splunk Attack Range [2], which is set up to utilise multiple server instances (VMs) and provides red team tooling such as Atomic Red [3] and logging support through Splunk. While this provides support for local deployment, it is resource-intensive and seems to be targeted at cloud-based deployments, with multiple open issues for local deployment, some of which are over 12 months old as of writing [4].

In our work, we propose GoibhniUWE (pronouncedgo-bin-you-ee), a platform that extends the concept of a cyber range to offer a much more modular and scalable approach using containerisation. Our system facilitates a modular methodology where multiple vulnerable services can be run as part of a single deployment that may span across multiple networks, allowing users to create larger, more complex attack scenarios using real-world vulnerabilities. We utilise an existing open-source (OS) resource Vulhub [5], which allows users to practice against services that are vulnerable to a range of vulnerabilities from misconfiguration to high-impact CVEs such as log4j [6]. Furthermore, the introduction of containerisation enables researchers and practitioners to study additional infrastructure concepts such as container segregation and escape. These challenges are increasingly significant in the latest computing paradigms and move the focus away from the security of individual workstations or VMs. Importantly, we provide logging at multiple levels, enabling end users to observe common indicators of different attacks in real time; facilitating ongoing, layered learning throughout; and allowing for the simple provision of datasets made up from multiple logging sources.

2. Technical Background

In this section, we introduce some of the key virtualisation concepts discussed.

2.1. Virtual Machines

Virtual Machines (VMs) are a way of packaging and deploying an entire "machine", which can easily be shared or used by others. This form of virtualisation comes with a complete Operating System (OS), including the kernel. As such, it can be fairly resource-intensive, requiring significant storage. However, this allows a VM to run a completely different OS to the host machine, as it is using its own (virtualised) kernel, and provides robust isolation from the host machine. This has made VMs a perfect candidate for use cases such as malware sandboxes [7]. However, the storage requirements for VMs limit how many instances can be run on a single server and they can also incur slower boot times and a more complex configuration and management, especially at a large scale [8].

2.2. Containerisation

Containers are a more lightweight form of virtualisation, with a focus on the application layer. A container will contain an application and all the required libraries and packages to run that application. Additional packages and utilities are often missing to reduce "bloat", and it does not include OS-level components, such as the kernel, utilising the host OS. This weakens the isolation between the container and the host machine and introduces a co-dependency, with containers needing to run on the target OS they were built for. However, they have lower storage requirements, faster startup times, and are easy to manage at a large scale [8]. These factors have increased their usage, with a recent report by Statista showing that " in 2021, 96 percent of respondents from a global survey state were using container technology" [9].

3. Related Works

We address the existing work on the research and development of cyber range techniques.

3.1. VM-Based Cyber Ranges

In [10], Yamin and Katt created a software-based cyber range which aims to provide realistic cyberthreat scenarios in cyber security education settings. Designed to support both red and blue team scenarios, the proposed approach allows for the creation of attacker and defender agents. The vulnerability injector module is responsible for adding three types of vulnerabilities (software, services, and configurations) into the simulated infrastructure. Whilst the tool provides a useful learning platform, there are some limitations, such as the non-standard integration with existing Common Vulnerabilities and Exposures (CVE) frameworks for the use of vulnerability injection. Furthermore, the platform is designed on the concept of a Capture The Flag (CTF) challenge; therefore, it does not have the same emphasis on logging and monitoring that a real-world environment would rely upon.

The AIT cyber range developed by Leitner et al. [11] adopts a similar approach in cyber range development. AIT uses OpenStack for managing the computing platform, while Terraform is used to manage the provisioning of the underlying infrastructure. GameMaker, a Scenario Engine, is used to define the cyber range scenarios and manage its flow. While the modular approach of the range allows for scalability and ease of deployment, the scenario engine is limited to defining non-technical aspects of the cyber scenario.

Vykopal et al. [12] adopt a similar approach in the implementation of two cyber ranges, namely KYPO CRP (Cyber Range Platform) and Cyber Sandbox Creator (CSC). While both use the same network topology and the underlying learning analytics stack, KYO CRP is designed to run on the cloud, while CSC is designed to run on the students' laptops. The use of open-source technologies (e.g., OpenStack and VirtualBox) allows for ease of deployment.

Beuran et al. [13] also adopted a cloud-based approach to cyber range development in demonstrating their Cybersecurity Training and Operation Network Environment (CyTrONE) cyber security training framework. Building upon the Cyber Range Instantiation System (CyRIS) [14], it consists of modules to manage resource creation and provisioning for the Virtual Machines (VMs) with content creation managed by the framework's Content Installation module. While CyTrONE is able to cater for up to 600 students, it is primarily aimed at providing training and thus is limited in executing the entire lifecycle of a sophisticated attack.

3.2. Container-Based and Hybrid Cyber Ranges

In [15], Oh et al. present a Raspberry Pi cyber range as a low-cost approach to cyber security education. They use Docker Swarm to provide a cluster-based container platform across four connected Raspberry Pi 3 devices. However, they then focus on the use of the Damn Vulnerable Web Application (DVWA) as a containerised application that could be used for teaching with this platform. This lacks the red/blue team investigation, and as previous platforms have also shown, it puts the focus on cyber ranges more as a CTF platform, rather than as a logging and monitoring platform akin to real-world security operations.

Nakata and Otsuka [16] investigate the use of container-based cyber ranges and compare resource usage between their system CyExec* and hypervisor or host-based approaches. Overall, their container-based cyber ranges used half as much memory and 1/60th of the storage, while still being able to reproduce 99% of vulnerabilities. Their scenario deployment included randomisation but was built on a singular default scenario with a Metaspolitable2 [17] target server. Though some logging was provided via SNORT [18], it does not easily facilitate complex attack scenarios or extensive logging by default, making the design and testing of complex attack scenarios a more manual process.

In the work of Chouliaras et al. [19], the authors present their cyber range platform (UNIWA) based on Docker using Infrastructure as Code (IaC) methodology. UNIWA itself

consists of six core modules and uses both VMs and containers as part of the complete system architecture. While scenarios can be created and configured dynamically, the platform itself has a number of pre-requisites, including multiple OpenStack APIs. Testing was conducted on systems with 32 GB and 16 GB of dedicated RAM and it still showed delays during scenario deployment when it was stress-tested. The tested scenarios themselves made no mention of logging or IDS deployment which could be utilised by students for the analysis or identification of attacks.

3.3. Frameworks and Overview

In [20], the authors propose the Cyber Range Design Framework (CRDF), which aims to provide a set of standards for the deployment of cyber ranges. They highlight identified weaknesses in existing cyber range designs and implementations, which the proposed CRDF seeks to address. These include (amongst others) the need for ongoing, layered learning and the consideration of "workspace requirements" for on-demand learning and deployment.

In a similar work, Ukwandu et al. review existing cyber ranges and test beds, providing a breakdown of the underlying technologies used, the intended use case(s) or environments, and a taxonomy for cyber ranges [21]. Within their review, they highlight the benefits of container-based cyber range platforms, including their flexibility, scalability, and portability.

Many of the cyber range deployments previously mentioned are relatively resource-intensive, particularly in terms of the usage and number of VMs. This presents a potential challenge when deploying in memory-constrained environments or those without a stable and reliable connection to cloud-hosted resources. Additionally, there are considerations around setup complexity, with some ranges using multiple existing IaC platforms, which themselves may have hardware and software requirements, making them unsuitable for local deployments. Many are also largely focused on a CTF-style approach, with limited attention given to real-time logging and monitoring of the attacks or scenarios. We contend that the logging and monitoring aspects of the cyber range are essential to provide both an educational experience for students and a sophisticated experimental platform for researchers, akin to real-world deployment. In this paper, we present our cyber range, GoibhniUWE, which aims to address identified issues surrounding logging and lightweight deployment. Furthermore, it aligns with the proposed CRDF framework [20] to provide a platform that enables on-demand, layered learning within an academic context. By leveraging existing open-source (OS) resources such as Vulhub [5] we aim to meet the requirement for ongoing learning, with new vulnerable environments being regularly added by the OS community. These can be utilised to provide attack scenarios or targets, based on real-world CVEs or common misconfigurations, with minimal setup required by the end user, while still allowing for full customisation of the deployment itself. At the time of writing, there were over 350 CVEs covering 41 Common Weakness Enumerations (CWEs), with over 150 additional, misconfigured vulnerable environments (categorised as Misc withing GoibhniUWE) present within Vulhub [5], totalling over 500 vulnerable environments which could be deployed using GoibhniUWE. A full list of the CWEs represented can be found in Appendix A.

4. Design of the GoibhniUWE Cyber Range

The idea behind the development of the GoibhniUWE cyber range is to provide a containerised environment which can be used for both education and research purposes. The design principles behind its development are supported by the following requirements:

- *R*1—Minimal deployment: The cyber range should deploy and run with minimal resource overhead.
- *R*2—Modular: The cyber range should support the ability to easily swap out different vulnerable services depending on the desired scenario.

- *R*3—Real-time: The cyber range should support real-time monitoring and logging from multiple sources.
- *R*4—Scalability: The cyber range should support the ability to scale up the infrastructure as required by the deployed scenario.

These requirements also underpin our proposed solution to issues highlighted as part of the CRDF [20]; the consideration of "workspace requirements" for on-demand learning and deployment (*R*1); and the need for ongoing, layered learning (*R*2–*R*4).

Figure 1 provides a high-level graphical overview of the GoibhniUWE cyber range. The use of containers as an alternative to VMs in the implementation of our cyber range allows for quicker deployment, with minimal performance overhead (*R*1). Different vulnerable services can be selected through the User Interface (UI), and these will be deployed using Docker (through a Python API). These vulnerable services are then run as containers, with networking and IP assignment managed by GoibhniUWE. This allows for ease of scaling (*R*4) of the range to add other functionalities, such as internal services, which require pivoting from the initial access point or target OS to access them.

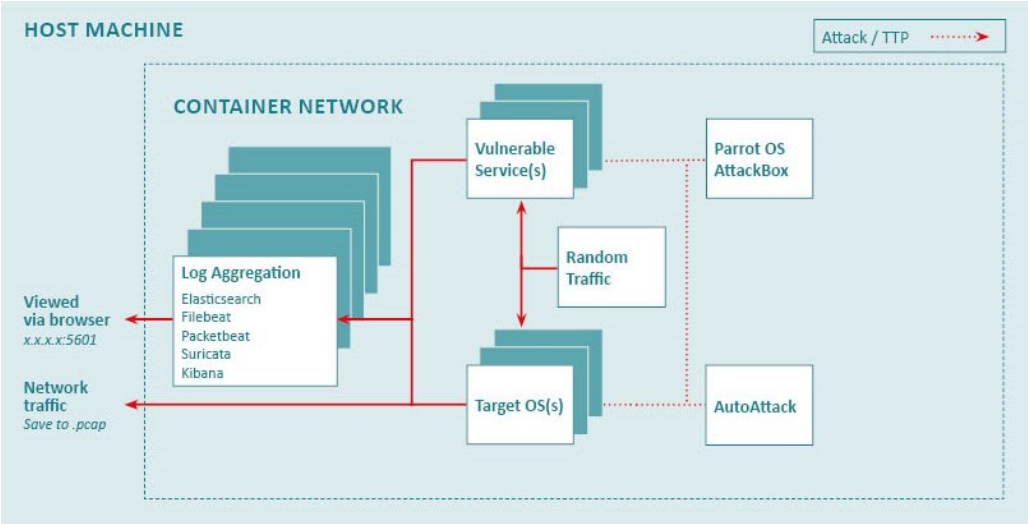


Figure 1. Container-based architecture of GoibhniUWE.

Designed to facilitate the analysis of various attacks and Tactics, Techniques, and Procedures (TTPs), our container-based cyber range is composed of multiple containers. The AttackBox runs Parrot OS, a Linux-based distribution that is designed for penetration testing and that comes pre-loaded with multiple, relevant tools, such as Nmap, msfvenom, etc. This is used to simulate an engagement by a malicious actor.

Vulnerable services are launched separately and will vary depending on the target architecture desired. These can consist of single to multiple services which are positioned throughout the target network, with the option to have these set up to mimic external or internal services. End users can set up process monitoring on a specific "target" container. This can be a default Ubuntu or Alpine OS or one of the selected vulnerable container instances (*R*2).

Auxiliary containers can also be launched as part of the scenario setup and deployment. For example, a RandomTraffic container that will generate random traffic targeted at a provided list of endpoints (or single running service) and an AutoAttack container which runs automated scans against vulnerable service(s) can be utilised, adding additional "malicious" traffic to the captured logging. These containers add a degree of "realism" to scenarios and provide noise to captured logging which can be useful when using the generated logging in support of research (such as input to Machine Learning (ML), Deep Learning (DL), or Artificial Intelligence (AI) tooling), as well as for educational purposes.

Logging is conducted at multiple points to ensure maximum coverage of actions taken. Network traffic is captured using tcpdump, which is deployed on the host, and Packetbeat is deployed as a container. A Suricata container configured with default rules is used as an Intrusion Detection System (IDS). Container logs are captured using Filebeat and, along with the Suricata logs, are fed into a containerised Elasticsearch instance. The log aggregation simulates a Security Information and Event Management (SIEM) system and is fed into a Kibana container, which provides a dashboard for real-time monitoring and analysis, which can be accessed on the host machine. Logging can be turned on or off, and different elements of the logging can be run depending on the end user's requirements; for example, container logging via Filebeat can be run without Packetbeat, Suricata, or tcpdump (*R*2, *R*3).

All traffic from the host machine is altered to appear as if it originated from the AttackBox in the saved pcaps. This allows a simulated attack to use web browsers and other UI tools that would not be available to the AttackBox container, without introducing further complexity and while avoiding a "secondary" attacking IP and ensures consistency of events within the traffic capture. In this way, traffic capture can be used for post-event analysis or even as part of wider academic research on attack scenarios.

The entire architecture is deployed using a UI created and launched from a single Python script, using the Django web application framework. This manages the creation of the container network, if it does not already exist; deployment of various containers as required for the target architecture; and the management of logging components and logging aggregation (*R*1).

Vulnerable services can be deployed across two Docker networks labelled "external" and "internal", with at least one container having access to both networks in each deployment. This allows for the creation of complex attack scenarios that would require an attack to move between networks. Figure 2 shows an example infrastructure diagram that is generated within the UI as part of the environment setup. This particular example shows the creation of a multi-network setup and highlights the information made available to end users at the point of deployment. This diagram is dynamically generated and changes to match the deployment configuration, and it can be saved as a .png file to be used as a reference point with the associated logging data. Vulnerable containers can also have ports exposed via the localhost, allowing for attacks to originate from outside the hosting machine (instead of using the provided AttackBox container) and the use of GoibhniUWE as part of a CTF-style deployment.

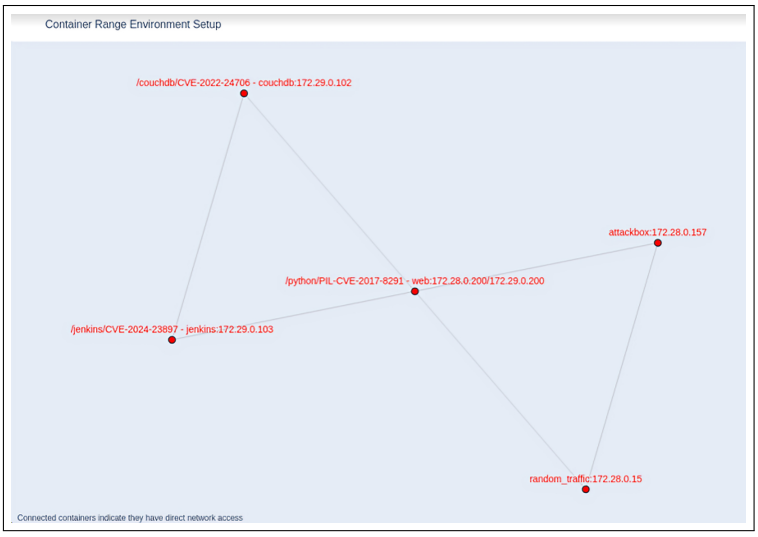


Figure 2. GoibhniUWE UI infrastructure diagram.

These vulnerable services are pre-populated using the Vulhub [5] repository and are mapped against their associated CWE [22]; these CWEs are presented for users to select

from in the first instance, via the UI, as shown in Figure 3. Once a CWE has been selected, the user can see a list of services and the associated CVEs, as shown in Figure 4. This allows a user to easily create an attack environment or scenario that fits their requirements. By utilising an existing repository of vulnerable services, the cyber range facilitates ongoing learning, allowing users to create scenarios using a growing number of vulnerable services.

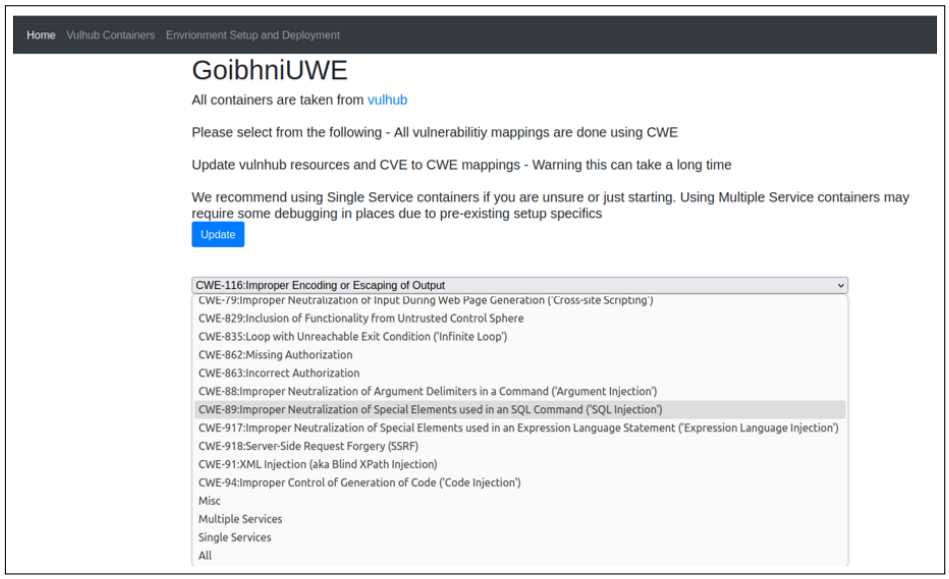


Figure 3. GoibhniUWE UI initial Vulhub categorisation selection.

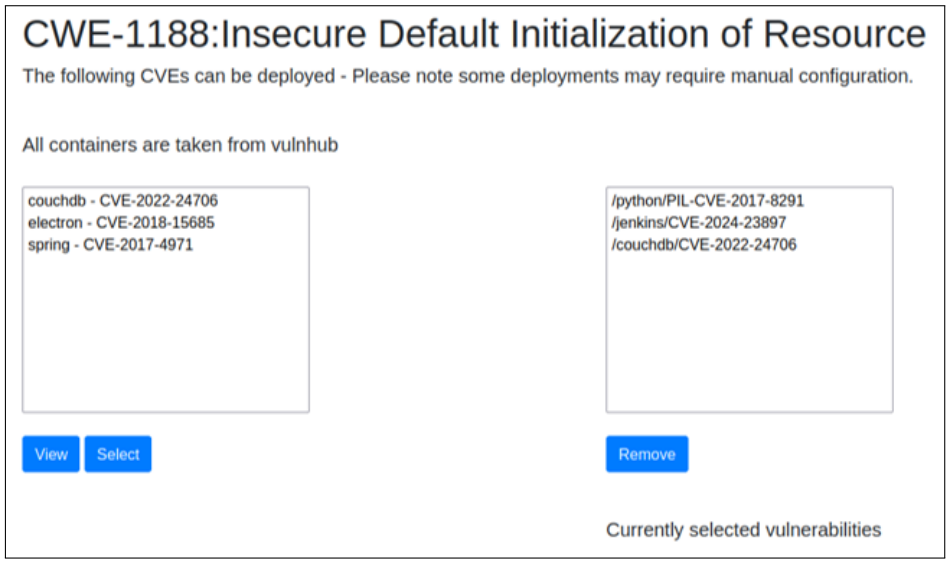


Figure 4. GoibhniUWE UI CVE and service selection.

We also include options to modify the deployments and allow for attackers to practice container escapes, such as through mounted folders, default root user permissions, or by making the container itself privileged (or any combination of all three). To facilitate analysis and training in the detection of such escape methods, process monitoring of the "target" instance can also be enabled.

5. Experimentation

To provide a comparative analysis against existing cyber ranges, GoibhniUWE was deployed within a Ubuntu 20.04 (Long-Term Support) VMware VM. The VM was set to run using 8 GB of memory with four single-core processors to provide a running environment that could be widely utilised, setting a low barrier of entry for the use of our cyber range

and addressing considerations of "workspace requirements" for on-demand learning and deployment [20].

We follow the work of Nakata and Otsuka [16] to establish a baseline comparison for our experimentation. In their work, they proposed the CyExec* platform and compared this against SecGen [23], another VM-based container range designed to generate randomly insecure environments. In their experimentation, Nakata and Otsuka ran between 1 and 10 simultaneous instances of container-, host-, and hypervisor-based cyber ranges, highlighting the effectiveness of the container-based cyber range [16]. The metrics they used for performance were startup time, memory usage, CPU usage, and storage usage. We focus on the first three metrics in our experimentation and results. As GoibhniUWE uses existing OS containers and is reliant on Vulhub GitHub [5], elements of storage are beyond our control; however, it is worth noting that the core files and scripts, not including resources downloaded from Vulhub, total less than 39MB.

Metrics for the CPU and memory usage were collected by using the "sysstat" functionality, which was run on a poll every 10 s and re-directed to a logging file. We ran GoibhniUWE under multiple conditions and recorded CPU and memory usage during startup, platform usage, and shutdown. The tested conditions were as follows:

- **T1 (10 containers—no pull or logging):** 9 vulnerable service containers with the required AttackBox running. All container images were on the host system.
- **T2 (10 containers—pull and full logging):** 4 vulnerable service containers with the required AttackBox and full logging. All container images were downloaded during scenario deployment.
- **T3 (10 containers—full logging and random traffic):** 3 vulnerable service containers with the required AttackBox, RandomTraffic and full logging. All container images were on the host system.
- **T4 (16 containers—full logging):** 10 vulnerable service containers with the required AttackBox and full logging. All container images were on the host system.
- **T5 (12 containers—CTF):** 11 vulnerable services and the required AttackBox. No logging and all container images were on the host system. The CTF ran for 8 h, and the deployed services were being targeted by up to 12 external attackers at a time.

All conditions except the CTF were run five times, and the reported CPU, memory, and startup time results were averaged across all five runs. In all conditions, the deployment was actively used, with vulnerable services targeted/exploited; moreover, where appropriate, the SIEM was checked, and the logs were analysed.

During one run of the T3 scenarios, we also captured stats around logging. We queried the indices' stat endpoints every 10 s to obtain the total number of indexed documents and looked at the increment between each query. The T3 scenario was based around an existing APT (Earth Lucsa [24]), creating a scenario that allowed for the use of their known TTPs, as laid out in Table 1. The services were deployed on both "external" and "internal" networks, with the Python Flask server running on the "external" network and both CouchDB and Jenkins running on the "internal" network, as shown in Figure 5. This was completed to mimic realistic infrastructure deployment and allow for the collection of complex, APT-level, attack data that span multiple networks, including random web traffic for noise. The modular nature of GoibhniUWE allows for the creation of multiple such deployment instances, which can all be tailored to different attack complexities or specific APT groups and their associated TTPs. For the purposes of this work, we illustrate how a complex APT scenario can easily be replicated using the GoibhniUWE platform. The nature of the logging data from this example and similar APT-focused deployments are part of ongoing research into intelligent threat detection and responsiveness.

Table 1. MITRE ATT&CK TTP and Action Mapping—Earth Lusca.

MITRE ATT&CK TTP	Action	Target
Active Scanning: Vulnerability Scanning-T1595.002	Use of Nmap, dirb, and Nikto	Python Flask server
Exploit Public-Facing Application-T1190	Reverse shell via crafted upload	Python Flask server (Ghostscript)
Compromise Infrastructure-T1584 & Remote System Discovery-T1018	Use of compromised Flask server to launch further reconnaissance and attacks	Internal services (CouchDB and Jenkins)
Command and Scripting Interpreter: Python-Port scanning-T1059.006	Crafted Python scripts for port scanning and exploitation	CouchDB (CVE-2022-24706)

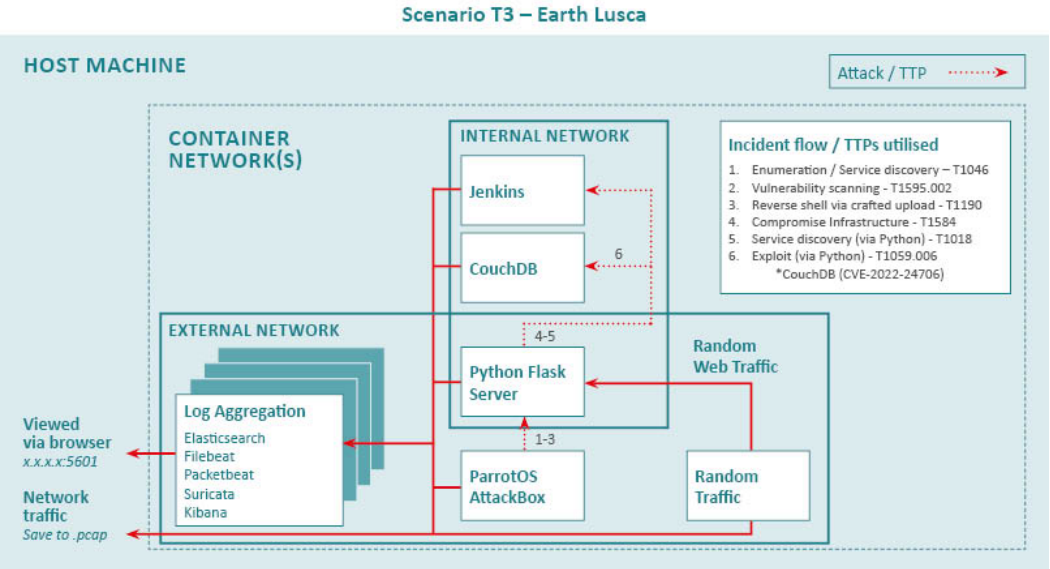


Figure 5. Overview of T3—based on known TTPs of APT Earth Lusca.

6. Results and Discussion

Table 2 shows the results of running GoibhniUWE under the different experimental conditions. The highest average CPU load was below 50%, and the memory usage never exceeded 5.1 GB, which is the highest reported memory usage for CyExec* [16]. The different scenarios showcase the ability to support a large number of containers, complete with logging (T4, 16 containers including active logging) and a long-running (T5 ran for 8 h), continually targeted use, with attacks from multiple machines. Importantly, GoibhniUWE remained stable throughout, highlighting stability and flexibility in deployment scenarios, with all deployed containers remaining active and in use throughout all scenarios.

Table 2. Sysstat logs and startup times per scenario.

Scenario	CPU Usage (%)	Memory Usage (GB)	Startup Time
T1	11.34	3.18	19 s
T2	29.87	3.21	11 m 34 s
T3	27.64	3.28	5 m 43 s
T4	28.96	5.06	5 m 48 s
T5	45.66	4.22	20 s

The startup time, while lower than the peak startup for comparable ranges (such as times reported in [16]) is, on average, significantly higher when utilising the full logging capability. Without this, the startup time is consistent with that of CyExec*.

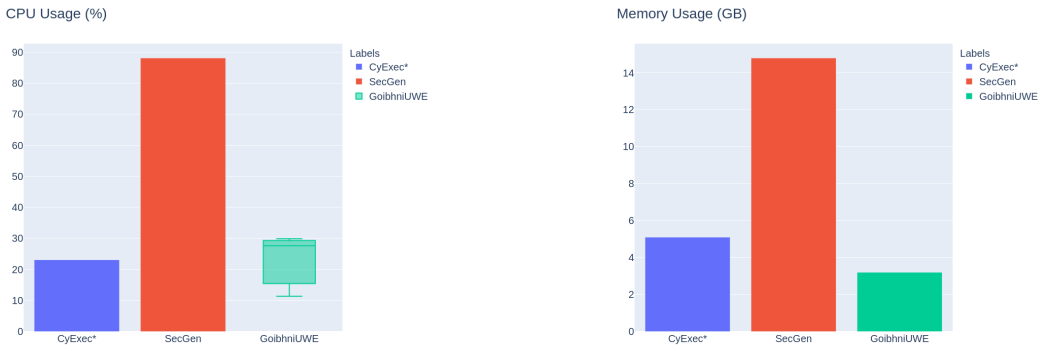
Table 3 shows the results from the log aggregation stats, gathered during the running of a T3 scenario. This shows that during the T3 scenario, the Elasticsearch stack was ingesting an average of 642 logs every 10 s. At its highest point(s), the Elasticsearch stack was handling 1000 s of logs, while at its lowest, it was still handling around 3 logs a second. This highlights the significant amount of logging data captured and available when using GoibhniUWE with full logging, not accounting for the independent network traffic capture. Due to preset limitations and configuration within the deployed Elasticsearch stack, this logging can be carried out while still maintaining low computational usage.

Table 3. Log aggregation stats (logs ingested every 10 s)—based on T3.

Logging Source and Index	Average	Min	Max
Filebeat	118	10	3122
Packetbeat	524	26	9169

One issue that we found with comparing our work to the experimentation carried out by Nakata and Otsuka [16] is that the hardware used is not discussed. Therefore, while we have results which indicate CPU usage as a percentage, it is unclear what this is of (4, 8, 12 processors?). We have mapped these percentages to our minimal running environment to provide a baseline comparison that does not unfairly represent CyEexc* itself.

Figures 6 and 7 show the CPU and memory usage comparison between CyExec*, SecGen, and GoibhniUWE. The CPU and memory usage for both CyExec* and SecGen were taken from [16] and are based on their reporting of running "10 scenarios", which is the highest reporting point in the study and was mapped to our running environment, as discussed above. Figure 6a shows a box plot for the different experimental conditions that GoibhniUWE was run under. The highest reporting point for this comparison (Figure 6a) includes CPU usage during a fresh pull of all images used and full logging for completeness of reporting (T2). Even when running 10 vulnerable services with full logging (a total of 16 container instances-T4), the CPU usage was less than 7% higher than the 10 "scenarios" reported for CyExec*, and memory usage was the same for both. We have chosen to run this many container instances so that our reporting can be equated to having 10 scenarios (vulnerable instances) as a reporting point, while also using the full logging which is a key feature of the GoibhniUWE platform. This provides a comparison point which shows there is a minimal resource consumption increase for an arguably higher workload within the GoibhniUWE platform.



(**a**) CPU usage: T1–T3 on GoibhniUWE.

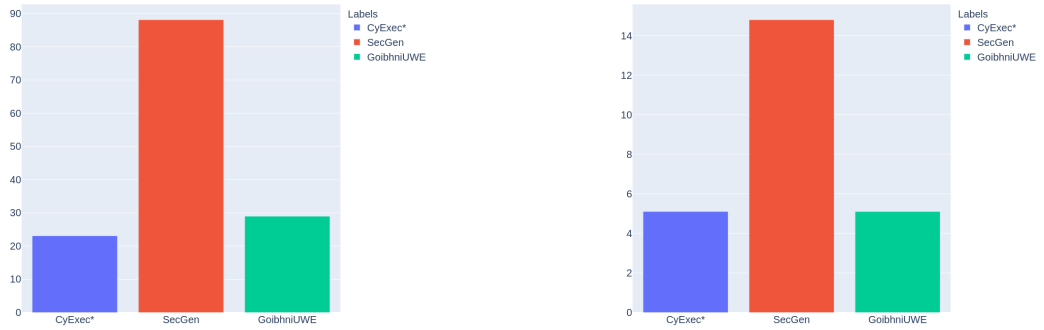
(**b**) Memory usage: T1–T3 on GoibhniUWE.

Figure 6. CPU usage (**a**) and memory usage (**b**) comparison—with 10 "scenarios" on CyExec* and SecGen and 10 containers on GoibhniUWE.

An additional comparison with the Splunk Attack Range [2], a VM-based cyber range with a focus on logging and "detection development" as a comparison point for a cyber range with a similar focus on logging as GoibhniUWE, was planned. However, attempting to run this under the same conditions as GoibhniUWE proved unsuccessful, with the host VM crashing on two consecutive attempts. To resolve this, the CPU and memory allocation for the VM were increased to eight processors and 12 GB. Even under these conditions, the Splunk Attack Range could only be run under minimal conditions (no Windows Server). At this stage, it was apparent that the significant computational resource required for deploying the Splunk Attack Range negated the rationale for a comparison with our lightweight container-based configuration.

Our experimentation shows that we have met the initial design and system requirements laid out for GoibhniUWE:

- *R*1—Minimal deployment: Even during continuous active usage, the deployment averaged less than 50% CPU usage and 5.1GB of memory.
- *R*2—Modular: Running conditions and deployments can be easily altered in a repeatable fashion.
- *R*3—Real-time: Real-time monitoring and logging from multiple sources could be enabled and utilised.
- *R*4—Scalability: The number of vulnerable services could be increased, and deployments could be actively targeted by multiple external attackers.



(**a**) CPU usage: T4 on GoibhniUWE.

(**b**) Memory usage: T4 on GoibhniUWE.

Figure 7. CPU usage (**a**) and memory usage (**b**) comparison—with 10 "scenarios" on CyExec* and SecGen, and 16 containers on GoibhniUWE.

Limitations

While the cyber range has demonstrable applications and helps to meet current gaps in modular, behavioural analysis for end-to-end engagement research, there are still some areas which can be improved upon and will be the subject of further work:

- Host OS. Currently, GoibhniUWE is limited to deployment on Linux-based hosts and, therefore, Linux-based target architectures. The use of containerisation should make the transition to Windows and MacOS hosts relatively straightforward; however, this has not been fully explored to date.
- Manual engagements and attack modelling. While the setup of the target environment and container architecture itself is automated, the role of the attacker is a manual process which is carried out by the end user. Though this does offer a degree of flexibility for experienced users, it does leave a gap in the ranges offering, as some VM-based cyber ranges do provide automated attacks (or simulations) through tools such as AtomicRedTeam [3].
- Log exporting. Currently, the traffic capture and IDS logs are saved locally to the host machine (IDS logs are also consumed by the Elastic stack). However, the collated Elastic logging, which includes all container logs and IDS logging, needs to be manually exported via elasticdump [25].

7. Conclusions and Further Work

In this paper, we presented GoibhniUWE, a modular container-based cyber range which is designed for the behavioural analysis of end-to-end engagements. By including multiple sources of monitoring and logging within the range itself, system logs, service logs, network captures, IDSs, and running processes, we can provide real-time analysis (via the Elastic stack) and data that can be easily exported and investigated in-depth after an engagement has been modelled.

The modular nature of the range allows end users to easily change out vulnerable services, modifying or removing services within the target deployment. These changes can be made prior to the setup of a target architecture or even during the live modelling of an engagement.

We have kept the GoibhniUWE system lightweight, able to handle 1000 s of logs a second during engagement modelling, with a minimal impact on computational usage, with a deployment running 16 container instances and full logging averaging 28.96% CPU usage (in a 4 single-core processor environment). Even with a continuously active, a multi-scenario (11 vulnerable services) CTF deployment run over 8 h GoibhniUWE averaged a CPU usage score of less than 50% and less than 4.5 GB of memory.

Our modular approach, using existing OS vulnerable services, with a minimal resource deployment helps GoibhniUWE address issues highlighted in a recent Cyber Range Design Framework (CRDF) [20], namely the need for ongoing, layered learning and the consideration of "workspace requirements" for on-demand learning and deployment.

GoibhniUWE also has demonstrable research use cases, having already been utilised to model complex, APT-level attacks spanning multiple services and networks. Due to inbuilt logging within the platform, the data generated from these attack scenarios can be used to create synthetic datasets to help further research within intelligent threat detection and responsiveness.

We have made GoibhniUWE available as an open-source project to help facilitate further research in this area, as well as to serve as an educational tool for community use. All relevant material, including the GoibhniUWE cyber range itself, can be found at https://github.com/uwe-cyber/GoibhniUWE (accessed on 23 August 2024).

Author Contributions: Conceptualization, A.M., J.W. and P.L.; methodology, A.M.; software, A.M.; validation, A.M., J.W. and P.L.; formal analysis, A.M.; investigation, A.M.; resources, A.M.; data curation, A.M.; writing—original draft preparation, A.M., J.W. and P.L.; writing—review and editing, A.M., J.W. and P.L.; visualization, A.M.; supervision, P.L.; project administration, P.L.; funding acquisition, P.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the internal Expanding Research Excellence funding scheme at the University of the West of England.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: All data from this study, including source code, were made publicly available at https://github.com/uwe-cyber/GoibhniUWE (accessed on 23 August 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. CWEs Available via Vulhub and GoibiniUWE

Table A1. CWEs available.

CWE
CWE-19: Data Processing Errors
CWE-20: Improper Input Validation
CWE-22: Improper Limitation of a Pathname to a Restricted Directory (Path Traversal)
CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component (Injection)
CWE-78: Improper Neutralization of Special Elements used in an OS Command (OS Command Injection)
CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)
CWE-88: Improper Neutralization of Argument Delimiters in a Command (Argument Injection)
CWE-89: Improper Neutralization of Special Elements used in an SQL Command (SQL Injection)
CWE-91: XML Injection (aka Blind XPath Injection)
CWE-94: Improper Control of Generation of Code (Code Injection)
CWE-116: Improper Encoding or Escaping of Output
CWE-125: Out-of-bound Reads
CWE-184: Incomplete List of Disallowed Inputs
CWE-190: Integer Overflow or Wraparound
CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
CWE-269: Improper Privilege Management
CWE-276: Incorrect Default Permissions
CWE-284: Improper Access Control
CWE-287: Improper Authentication
CWE-288: Authentication Bypass Using an Alternate Path or Channel
CWE-290: Authentication Bypass by Spoofing
CWE-306: Missing Authentication for Critical Function
CWE-307: Improper Restriction of Excessive Authentication Attempts
CWE-319: Cleartext Transmission of Sensitive Information
CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization (Race Condition)
CWE-384: Session Fixation
CWE-434: Unrestricted Upload of File with Dangerous Type
CWE-502: Deserialization of Untrusted Data
CWE-502: Deserialization of Untrusted Data
CWE-552: Files or Directories Accessible to External Parties
CWE-601: URL Redirection to Untrusted Site (Open Redirect)
CWE-611: Improper Restriction of XML External Entity Reference
CWE-704: Incorrect Type Conversion or Cast
CWE-755: Improper Handling of Exceptional Conditions
CWE-787: Out-of-bound Writes
CWE-829: Inclusion of Functionality from Untrusted Control Sphere
CWE-835: Loop with Unreachable Exit Condition (Infinite Loop)
CWE-862: Missing Authorization
CWE-863: Incorrect Authorization
CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement (Expression Language Injection)
CWE-918: Server-Side Request Forgery (SSRF)
CWE-1188: Insecure Default Initialization of Resources

References

1. DetectionLab. Introduction: DetectionLab. 2023. Available online: https://www.detectionlab.network/ (accessed on 6 February 2024).
2. Splunk Threat Research Team. Attack Range v3.0 | Splunk. 2023. Available online: https://www.splunk.com/en_us/blog/security/attack-range-v3-0.html (accessed on 6 February 2024).
3. Atomic Red Team. Explore Atomic Red Team. 2023. Available online: https://atomicredteam.io/ (accessed on 6 March 2024).
4. Splunk. Issues-splunk/attack_range. Available online: https://github.com/splunk/attack_range/issues (accessed on 23 August 2024).
5. Vulhub. Pre-Built Vulnerable Environments Based on Docker-Compose. 2023. Available online: https://github.com/vulhub/vulhub (accessed on 6 November 2023).

6. Vulhub-log4j. vulhub/log4j/CVE-2021-44228. Available online: https://github.com/vulhub/vulhub/blob/master/log4j/CVE-2021-44228/README.md (accessed on 23 August 2024).
7. ANY.RUN. How to Create a Sandbox Environment (for Malware Analysis). 2024. Available online: https://any.run/cybersecurity-blog/how-to-create-a-sandbox/ (accessed on 9 August 2024).
8. Aqua. VM vs. Container. 2024. Available online: https://www.aquasec.com/cloud-native-academy/docker-container/vm-vs-container/ (accessed on 9 August 2024).
9. Statista. Adoption of Container Technologies 2021. 2024. Available online: https://www.statista.com/statistics/1104543/worldwide-container-technology-use/ (accessed on 9 August 2024).
10. Yamin, M.M.; Katt, B. Modeling and executing cyber security exercise scenarios in cyber ranges. *Comput. Secur.* **2022**, *116*, 102635. [CrossRef]
11. Leitner, M.; Frank, M.; Hotwagner, W.; Langner, G.; Maurhart, O.; Pahi, T.; Reuter, L.; Skopik, F.; Smith, P.; Warum, M. AIT cyber range: Flexible cyber security environment for exercises, training and research. In Proceedings of the European Interdisciplinary Cybersecurity Conference, Rennes, France, 18 November 2020; pp. 1–6.
12. Vykopal, J.; Čeleda, P.; Seda, P.; Švábenskỳ, V.; Tovarňák, D. Scalable learning environments for teaching cybersecurity hands-on. In Proceedings of the 2021 IEEE Frontiers in Education Conference (FIE), Lincoln, NE, USA, 13–16 October 2021; pp. 1–9.
13. Beuran, R.; Tang, D.; Pham, C.; Chinen, K.i.; Tan, Y.; Shinoda, Y. Integrated framework for hands-on cybersecurity training: CyTrONE. *Comput. Secur.* **2018**, *78*, 43–59. [CrossRef]
14. Pham, C.; Tang, D.; Chinen, K.I.; Beuran, R. Cyris: A cyber range instantiation system for facilitating security training. In Proceedings of the Seventh Symposium on Information and Communication Technology, Ho Chi Minh, Vietnam, 8–9 December 2016; pp. 251–258.
15. Oh, S.K.; Stickney, N.; Hawthorne, D.; Matthews, S.J. Teaching Web-Attacks on a Raspberry Pi Cyber Range. In Proceedings of the 21st Annual Conference on Information Technology Education, New York, NY, USA, 7–9 October 2020; SIGITE '20, p. 324–329.
16. Nakata, R.; Otsuka, A. CyExec*: A High-Performance Container-Based Cyber Range With Scenario Randomization. *IEEE Access* **2021**, *9*, 109095–109114. [CrossRef]
17. Rapid7. Metasploitable 2. Available online: https://docs.rapid7.com/metasploit/metasploitable-2/ (accessed on 4 November 2023).
18. Cisco. Snort. Available online: https://www.snort.org/ (accessed on 4 November 2023).
19. Chouliaras, N.; Kantzavelou, I.; Maglaras, L.; Pantziou, G.; Ferrag, M.A. A novel autonomous container-based platform for cybersecurity training and research. *PeerJ Comput. Sci.* **2023**, *9*, e1574. [CrossRef] [PubMed]
20. Katsantonis, M.; Manikas, A.; Mavridis, I.; Gritzalis, D. Cyber range design framework for cyber security education and training. *Int. J. Inf. Secur.* **2023**, *22*, 1005–1027. [CrossRef]
21. Ukwandu, E.; Farah, M.A.B.; Hindy, H.; Brosset, D.; Kavallieros, D.; Atkinson, R.; Tachtatzis, C.; Bures, M.; Andonovic, I.; Bellekens, X. A review of cyber-ranges and test-beds: Current and future trends. *Sensors* **2020**, *20*, 7148. [CrossRef] [PubMed]
22. MITRE Corporation. CWE-New to CWE. 2023. Available online: https://cwe.mitre.org/about/new_to_cwe.html (accessed on 6 February 2024).
23. Schreuders, Z.C.; Shaw, T.; Shan-A-Khuda, M.; Ravichandran, G.; Keighley, J.; Ordean, M. Security Scenario Generator (SecGen): A Framework for Generating Randomly Vulnerable Rich-scenario VMs for Learning Computer Security and Hosting CTF Events. In Proceedings of the 2017 USENIX Workshop on Advances in Security Education (ASE 17), Vancouver, BC, Canada, 15 August 2017; USENIX Association: Berkeley, CA, USA, 2017.
24. Corporation, M. Earth Lusca, TAG 22, Group 1006 | MITRE-ATT&CK. 2022. Available online: https://attack.mitre.org/groups/G1006/ (accessed on 6 March 2023).
25. Elasticsearch-dump. Import and Export Tools for Elasticsearch & Opensearch . Available online: https://github.com/elasticsearch-dump/elasticsearch-dump (accessed on 23 August 2024).