Article

Not peer-reviewed version

# GoibhniUWE: A Lightweight and Modular Container-Based Cyber Range

Alan Mills *, Jonathan White, Phil Legg *

Posted Date: 21 June 2024

doi: 10.20944/preprints202406.1504.v1

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

 doi:10.20944/preprints202406.1504.v1



*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, UK.
* 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 Threat (APT). There is a need to consider how testbed environments such as cyber ranges can be readily deployed to improve examination of attack characteristics and the assessment of defences. Whilst cyber ranges are not new, they can often be computational expensive, require significant 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 cyber attack. Adopting a modular approach, users can seamlessly switch out existing, containerised vulnerable services, 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 (IDS), 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 Advance 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 2 processors and memory usage of less than 4.5 GB.

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

## 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 vulnerable application set. This has given rise to the idea of a 'cyber range' - 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 around the creation of VMs for providing suitable cyber security exercises. This has been popularised further by the introduction of cloud services to deploy VMs, such as TryHackMe and HackTheBox. 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 defensive tools built in, 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 4 VMs, this can be a time-consuming and resource-intensive deployment, requiring at least 16GB of RAM and 55GB of free hard-disk space to run locally. It is also a platform which is no longer supported, highlighting the issues around ongoing support and maintenance for such systems. A similar example is the Splunk

 doi:10.20944/preprints202406.1504.v1



Attack Range [2], which is setup 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 very resource intensive and seems to be targeted at cloud based deployments, with multiple open issues for local deployment, some of which are over 6 months old, as of writing[1].

In our work, we propose *GoibhniUWE* (pronounced *go-bin-you-ee*), a platform that extends the concept of a cyber range to offer a much more modular and scalable approach through the use of 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. Furthermore, the introduction of containerisation enables researchers and practitioners to study additional infrastructure concepts such as containerisation 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. Related Works

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

### *2.1. VM based Cyber Ranges*

In [4] Yamin and Katt created a software-based cyber range which aims to provide realistic cyberthreat scenarios in cyber 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 CVE frameworks for the use of vulnerability injection. Furthermore, the platform is designed on the concept of a CTF challenge, and therefore 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.* [5] 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 then 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.* [6] 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) allow for ease of deployment.

Beuran *et.al* [7] also adopted a cloud-based approach to cyber range development in demonstrating their Cybersecurity Training and Operation Network Environment (CyTrONE) cybersecurity training framework. Building upon the Cyber Range Instantiation System (CyRIS) [8], 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.

---

[1] https://github.com/splunk/attack_range/issues

 doi:10.20944/preprints202406.1504.v1



### 2.2. Container Based and Hybrid Cyber Ranges

In [9], 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 4 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 [10] 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 [11] target server. Though some logging was provided via SNORT [12] 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 by Chouliaras *et al.* [13] the authors present their cyber range platform (UNIWA), based on Docker using Infrastructure as Code (IaC) methodology. UNIWA itself consists of 6 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 carried out on systems with 32 GB and 16 GB of dedicated RAM and still showed delays during scenario deployment when stress tested. The tested scenarios themselves made no mention of logging or IDS deployment which could be utilised by students for analysis or identification of attacks.

### 2.3. Frameworks and Overview

In [14] 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 [15]. 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 light weight deployment. Furthermore, it aligns with the proposed CRDF framework [14] to provide a platform that enables on-demand, layered learning within an academic context. By leveraging existing Open Source (OS) resources such as Vulhub [16] 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

 doi:10.20944/preprints202406.1504.v1



world CVEs or common mis-configurations, with minimal setup required by the end user, while still allowing for full customisation of the deployment itself.

## 3. 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 - Ease of deployment: The cyber range should be easy to deploy 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 [14]; 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 and easier deployment, with minimal performance overhead (*R*1). In addition, it also enables an 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.

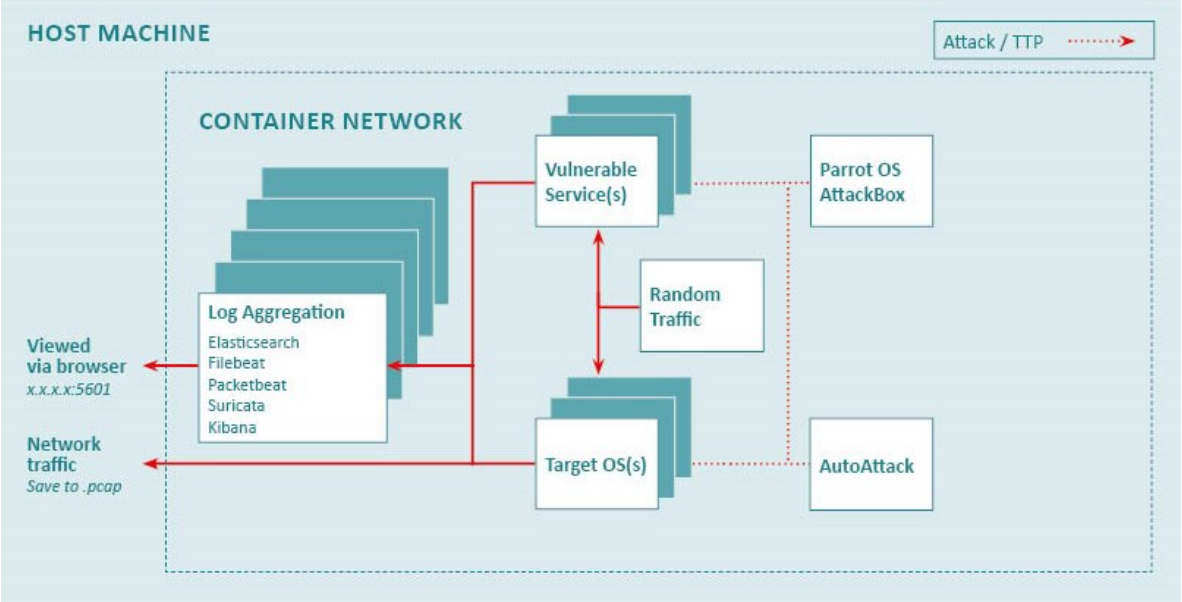



**Figure 1.** Container-based architecture of *GoibhniUWE*

Designed to facilitate easy 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 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

 doi:10.20944/preprints202406.1504.v1



network, with the option to have these setup to mimic external or internal services. End users are able to setup 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. 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), adding additional "malicious" traffic to 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 carried out at multiple points to ensure maximum coverage of actions taken. Network traffic is captured using tcpdump, which is deployed on the host, and PacketBeat deployed as a container. A Suricata container configured with default rules is used as an Intrustion 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 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 simulate attack to use web browsers and other UI tools that would not be available to the AttackBox container without introducing further complexity, while avoiding a "secondary" attacking IP and ensures consistency of events within the traffic capture. In this way, the 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, utilising the Django web application framework. This manages the creation of the container network, if it doesn't 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 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.

 doi:10.20944/preprints202406.1504.v1



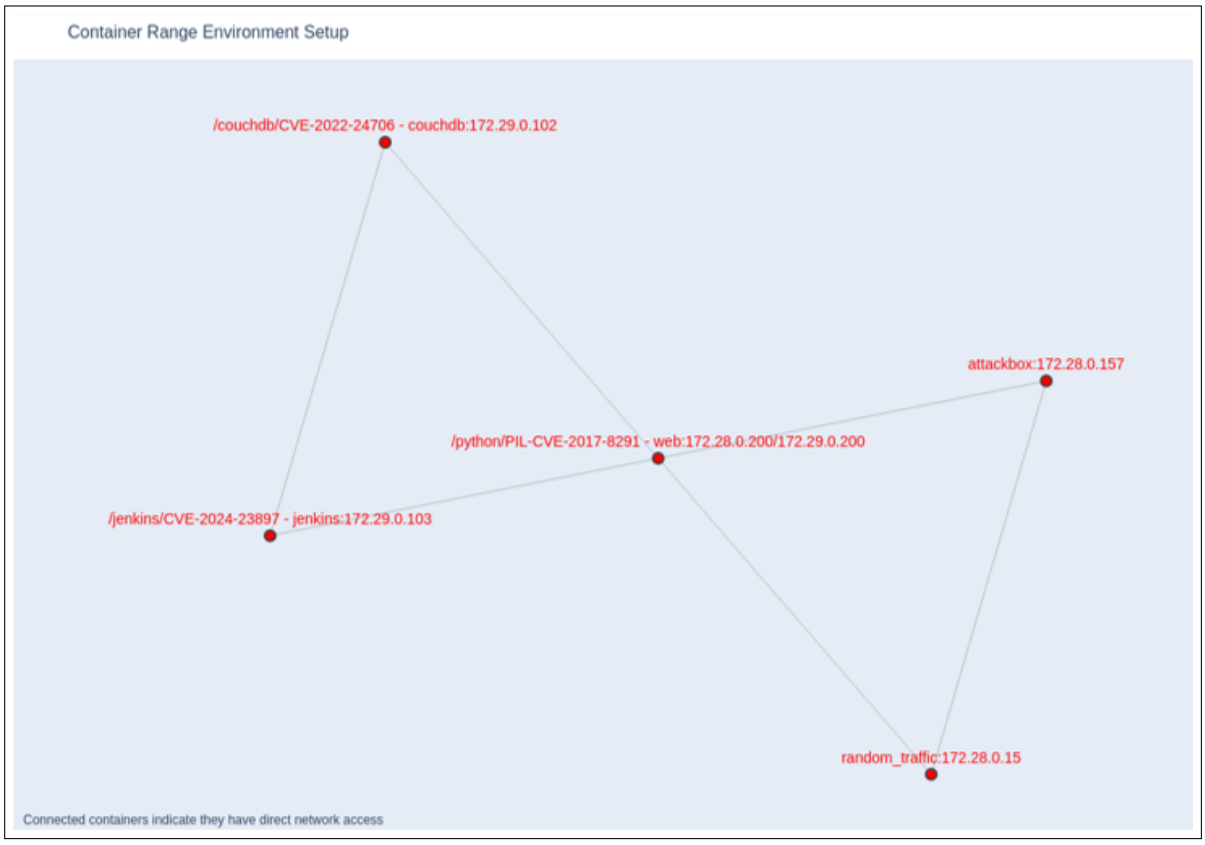



**Figure 2.** *GoibhniUWE* UI - Infrastructure diagram

These vulnerable services are pre-populated using the Vulhub [16] repository and are mapped against their associated Common Weakness Enumeration (CWE) [17], 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.

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 making the container itself privileged (or any combination of all 3). To facilitate analysis and training in the detection of such escape methods process monitoring of the "target" instance can also be enabled.

 doi:10.20944/preprints202406.1504.v1



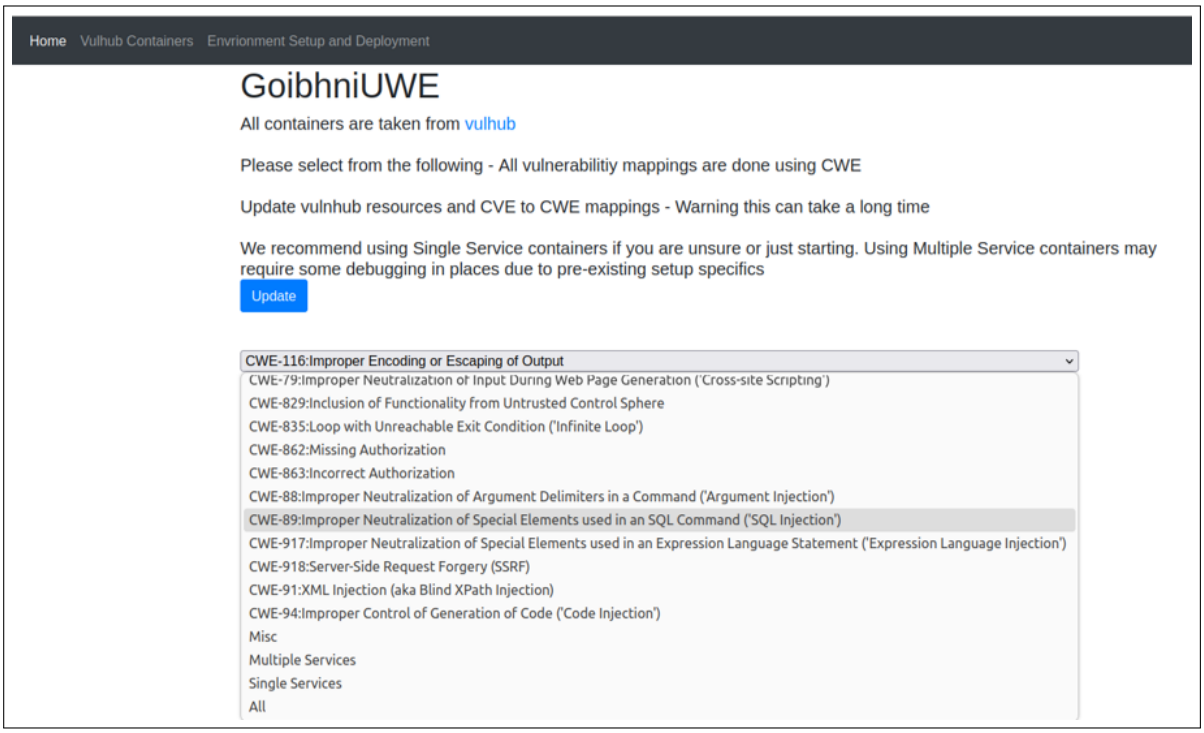



**Figure 3.** *GoibhniUWE* UI - Initial Vulhub categorisation selection

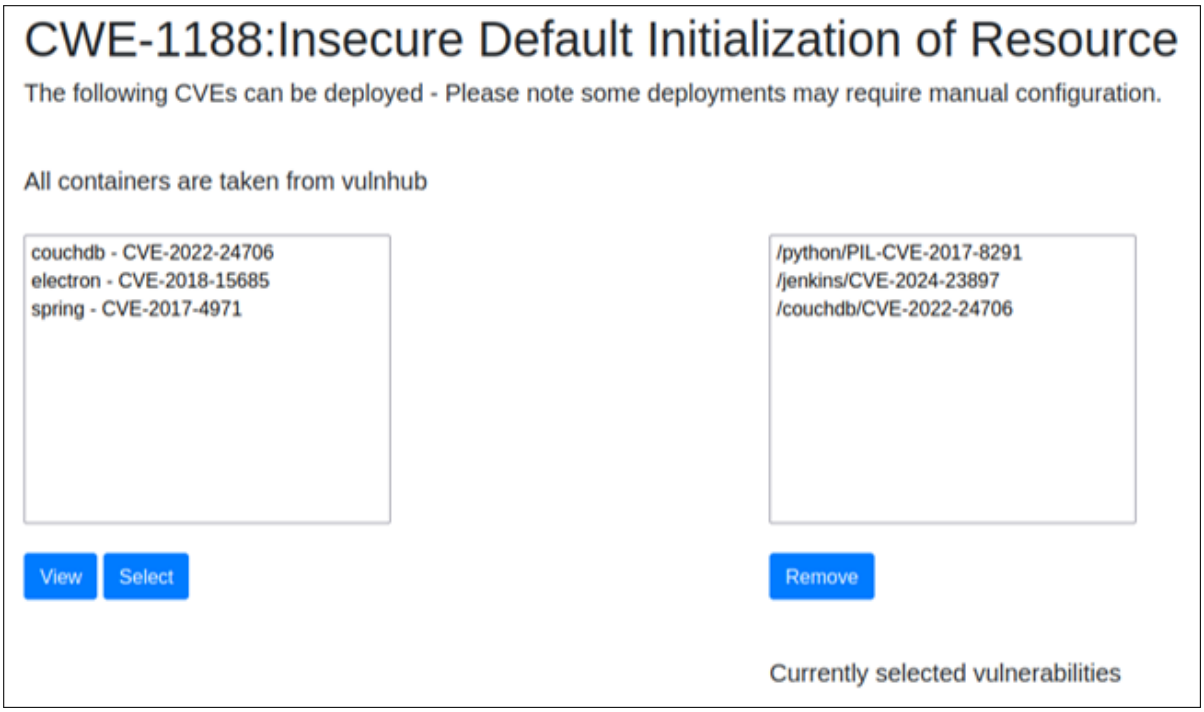



**Figure 4.** *GoibhniUWE* UI - CVE and service selection

## 4. Experimentation

To provide a comparative analysis against existing cyber ranges *GoibhniUWE* was deployed within a Ubuntu 20.0.4 (LTS) VMware VM. The VM was set to run using 8GB of memory with 4 processors to provide a running environment that could be widely utilised, setting a low barrier of entry for use of our cyber range and addressing considerations of "workspace requirements" for on-demand learning and deployment [14].

 doi:10.20944/preprints202406.1504.v1



We follow the work of Nakata and Otsuka [10] to establish a baseline comparison for our experimentation. In their work, they proposed the CyExec* platform and compared this against SecGen [18], another VM-based container range designed to generate randomly insecure environments. In their experimentation Nakata and Otsuka ran 1 - 10 simultaneous instances of container, host, and hypervisor based cyber ranges, highlighting the effectiveness of the container based cyber range [10]. The metrics they used for performance were; startup time, memory usage, CPU usage and storage usage. We focus on the first 3 metrics in our experimentation and results. As *GoibhniUWE* uses existing OS containers and is reliant on the Vulhub GitHub [16], elements of the storage are beyond our control, however it is worth noting that the core files and scripts, not including resources download from Vulhub, totals 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 seconds 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:

- **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 hours and the deployed services were being targeted by up to 12 external attackers at a time.

All conditions except the CTF were run 5 times and the reported CPU, memory and startup time results are the average across all 5 runs. In all conditions the deployment was actively used, with vulnerable services targeted / exploited and where appropriate the SIEM was checked and logs analysed.

During one run of the T3 scenario we also captured stats around logging. We queried the indices stats endpoint every 10 seconds to get 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 [19]), creating a scenario that allowed for 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 done to mimic a realistic infrastructure deployment and allow for the collection of complex, APT level, attack data that spans 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, here 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 response.

 doi:10.20944/preprints202406.1504.v1



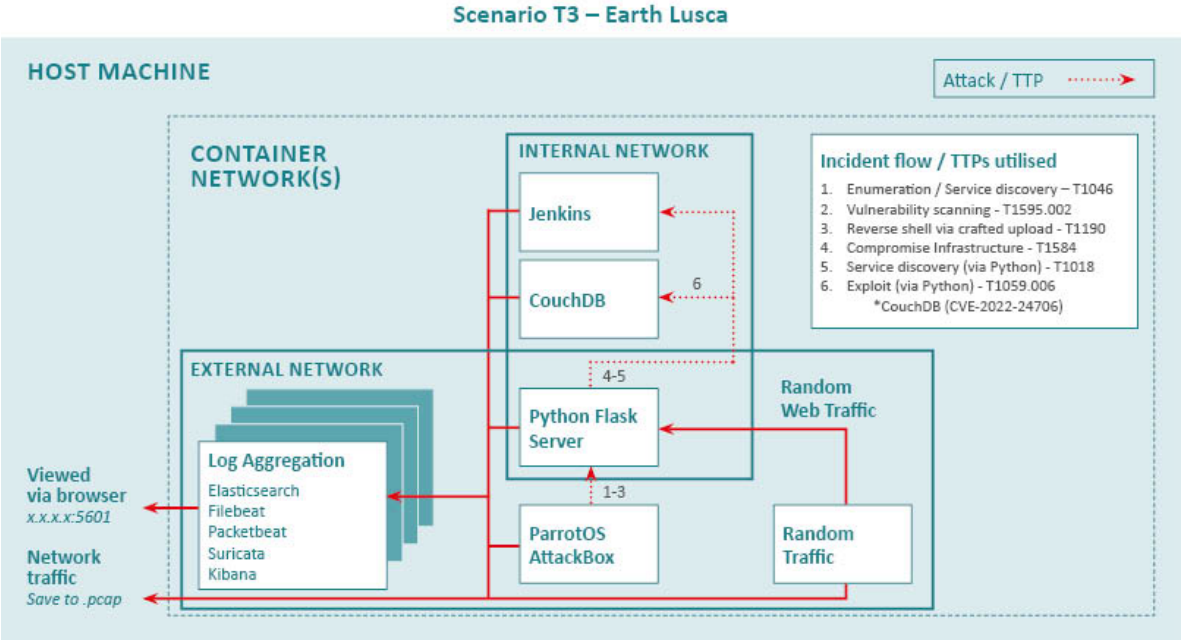



**Figure 5.** Overview of T3 - Based on known TTPs of APT Earth Lusca

**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) |

## 5. Results and Discussion

Table 2 shows the results of running *GoibhniUWE* under the different experimental conditions. Even during continual, targeted use the average CPU load is below 50% (2 processors) and the memory usage never exceeds 5.1 GB, which is the highest reported memory usage for *CyExec** [10].

The startup time, while lower than the peak startup for comparable ranges (such as times reported in [10]) is still on average significantly higher when utilising the full logging capability. Without this the startup time is consistent with that of *CyExec**.

**Table 2.** Sysstat logs and startup times per scenario.

| Scenario | CPU Usage (%) | Memory Usage (GB) | Startup Time |
|---|---|---|---|
| T1 | 11.34 | 3.18 | 19s |
| T2 | 29.87 | 3.21 | 11m34s |
| T3 | 27.64 | 3.28 | 5m43s |
| T4 | 28.96 | 5.06 | 5m48s |
| T5 | 45.66 | 4.22 | 20s |

 doi:10.20944/preprints202406.1504.v1



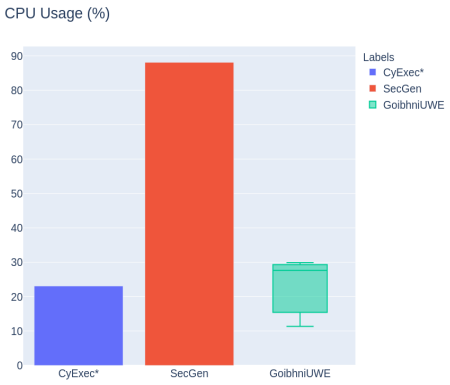
Table 3 shows the results from the log aggregation stats, gathered during the running of a T3 scenario. This shows that during the T3 scenario the Elasticsearch stack was ingesting an average of 642 logs every 10 seconds. At it's highest point(s) the Elasticsearch stack was handling 1000s of logs, while at it's 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 a low computational usage.

**Table 3.** Log aggregation stats (logs ingested every 10 seconds) - Based on T3.

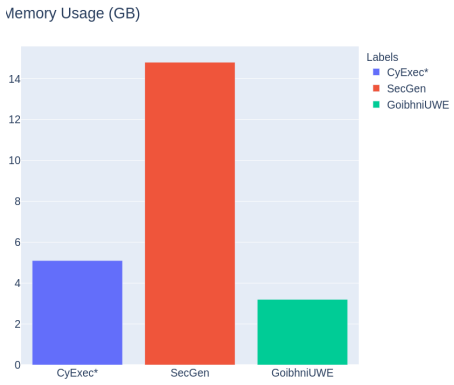
| Logging Source and Index | Average | Min | Max |
|---|---|---|---|
| Filebeats | 118 | 3122 | 10 |
| Packetbeats | 524 | 9169 | 26 |

One issue that we found with comparing our work to the experimentation carried out by Nakata and Otsuka [10], 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* have been taken from [10] and are based on their reporting of running "10 scenarios", which is the highest reporting point in the study and has been mapped to our running environment as discussed above. Figure 6a shows a box plot for the different experimental conditions *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 is less than 7% higher than the 10 "scenarios" reported for *Cyexec** and memory usage is 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*.

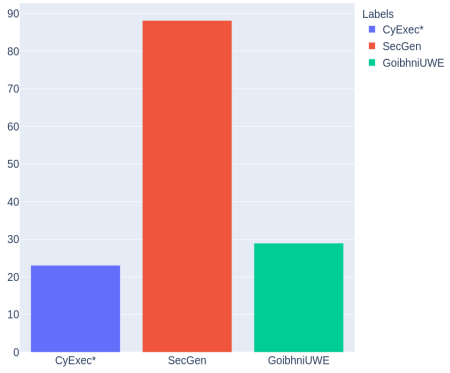



(b) Mem usage: T1-T3 on *GoibhniUWE*.

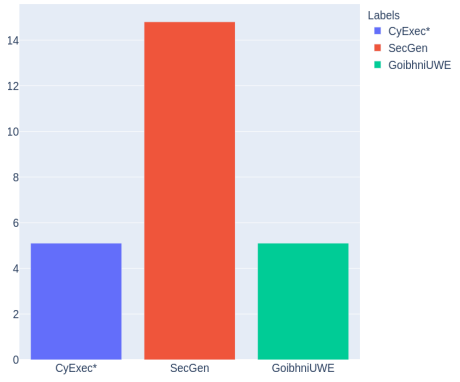
**Figure 6.** CPU usage (a) and Mem usage (b) comparison - With 10 "scenarios" on *Cyexec** and *SecGen* and 10 containers on *GoibhniUWE*

 doi:10.20944/preprints202406.1504.v1






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




(b) Mem usage: T1-T3 on *GoibhniUWE*.

**Figure 7.** CPU usage (a) and Mem usage (b) comparison - With 10 "scenarios" on *Cyexec** and *SecGen* and 16 containers on *GoibhniUWE*

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 to be unsuccessful with the host VM crashing on two consecutive attempts. To resolve this the CPU and memory allocation for the VM were increased to 8 processors and 12GB. 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 comparing against our lightweight container-based configuration.

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

- *R*1 - Ease of deployment: Even during continuous, active usage the deployment averaged less 2 CPU processors and 5.1GB of memory.
- *R*2 - Modular: Running conditions and deployments could 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

### *5.1. 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 straight forward, 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

 doi:10.20944/preprints202406.1504.v1



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[2].
- 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[3].

## 6. Conclusions and Further Work

In this paper we presented *GoibhniUWE*, a modular container based cyber range which is designed for 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, IDS and running processes, we are able to provide real time analysis (via the Elasticstack) as well as 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 1000s of logs a second during engagement modelling, with minimal impact on computational usage. With a deployment running 16 container instances and full logging averaging %28.96 CPU usage (in a 4 processor environment). Even with a continuously active, multi scenario (11 vulnerable services) CTF deployment run over 8 hours *GoibhniUWE* averaged a use of less then 2 CPU processors and less then 4.5GB of memory.

Our modular approach, using existing OS vulnerable services, with a minimal resource deployment helps *GoibhniUWE* addresses issues highlighted in a recent Cyber Range Design Framework (CRDF) [14], 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 data generated from these attack scenarios can be used to create synthetic datasets to help further research within intelligent threat detection and response.

We are making *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 can be found at https://github.com/uwe-cyber/GoibhniUWE.

**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 University of the West of England.

**Institutional Review Board Statement:** Not applicable.

**Data Availability Statement:** All data from this study, including source code, is made publicly-available at https://github.com/uwe-cyber/GoibhniUWE

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

## References

1. DetectionLab. Introduction :: DetectionLab. https://www.detectionlab.network/, 2023. Date accessed: $6^{th}$ Feburary 2024.

[2] https://github.com/redcanaryco/atomic-red-team
[3] https://github.com/elasticsearch-dump/elasticsearch-dump

 doi:10.20944/preprints202406.1504.v1




2. Splunk Threat Research Team. Attack Range v3.0 | Splunk. https://www.splunk.com/en_us/blog/security/attack-range-v3-0.html, 2023. Date accessed: 6$^{th}$ Feburary 2024.
3. Atomic Red Team. Explore Atomic Red Team. https://atomicredteam.io/, 2023. Date accessed: 6$^{th}$ March 2024.
4. Yamin, M.M.; Katt, B. Modeling and executing cyber security exercise scenarios in cyber ranges. *Computers & Security* **2022**, *116*, 102635.
5. 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. Proceedings of the European Interdisciplinary Cybersecurity Conference, 2020, pp. 1–6.
6. Vykopal, J.; Čeleda, P.; Seda, P.; Švábenskỳ, V.; Tovarňák, D. Scalable learning environments for teaching cybersecurity hands-on. 2021 IEEE Frontiers in Education Conference (FIE). IEEE, 2021, pp. 1–9.
7. Beuran, R.; Tang, D.; Pham, C.; Chinen, K.i.; Tan, Y.; Shinoda, Y. Integrated framework for hands-on cybersecurity training: CyTrONE. *Computers & Security* **2018**, *78*, 43–59.
8. Pham, C.; Tang, D.; Chinen, K.i.; Beuran, R. Cyris: A cyber range instantiation system for facilitating security training. Proceedings of the Seventh Symposium on Information and Communication Technology, 2016, pp. 251–258.
9. Oh, S.K.; Stickney, N.; Hawthorne, D.; Matthews, S.J. Teaching Web-Attacks on a Raspberry Pi Cyber Range. Proceedings of the 21st Annual Conference on Information Technology Education; Association for Computing Machinery: New York, NY, USA, 2020; SIGITE '20, p. 324–329.
10. Nakata, R.; Otsuka, A. CyExec*: A High-Performance Container-Based Cyber Range With Scenario Randomization. *IEEE Access* **2021**, *9*, 109095–109114.
11. Rapid7. Metasploitable 2. https://docs.rapid7.com/metasploit/metasploitable-2/. Date accessed: 4$^{th}$ November 2023.
12. Cisco. Snort. https://www.snort.org/. Date accessed: 4$^{th}$ November 2023.
13. Chouliaras, N.; Kantzavelou, I.; Maglaras, L.; Pantziou, G.; Ferrag, M.A. A novel autonomous container-based platform for cybersecurity training and research. *PeerJ Computer Science* **2023**, *9*, e1574.
14. Katsantonis, M.; Manikas, A.; Mavridis, I.; Gritzalis, D. Cyber range design framework for cyber security education and training. *International Journal of Information Security* **2023**, pp. 1–23.
15. 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.
16. vulhub. Pre-Built Vulnerable Environments Based on Docker-Compose. https://github.com/vulhub/vulhub, 2023. Date accessed: 6$^{th}$ November 2023.
17. MITRE Corporation. CWE - New to CWE. https://cwe.mitre.org/about/new_to_cwe.html, 2023. Date accessed: 6$^{th}$ Feburary 2024.
18. 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. 2017 USENIX Workshop on Advances in Security Education (ASE 17). USENIX Association, 2017.
19. Corporation, M. Earth Lusca, TAG 22, Group 1006 | MITRE- ATT&CK. https://attack.mitre.org/groups/G1006/, 2022. Date accessed: 6$^{th}$ March 2023.