

Article

RedHerd: Offensive Cyberspace Operations as a Service

Giovanni Pecoraro ^{*,†} , Mario D'Amico [†]  and Simon Pietro Romano 

Department of Electrical Engineering and Information Technology, University of Napoli Federico II, 80138 Naples, Italy; mariodamico@protonmail.com (M.D.); spromano@unina.it (S.P.R.)

* Correspondence: giovanni1.pecoraro@protonmail.com

† These authors contributed equally to this work.

Abstract: Nowadays, time, scope and cost constraints along with knowledge requirements and personnel training constitute blocking restrictions for effective Offensive Cyberspace Operations (OCO). This paper presents RedHerd, an open-source, collaborative and serverless orchestration framework that overcomes these limitations. RedHerd leverages the ‘as a Service’ paradigm in order to seamlessly deploy a ready-to-use infrastructure that can be also adopted for effective simulation and training purposes, by reliably reproducing a real-world cyberspace battlefield in which red and blue teams can challenge each other. We discuss both the design and implementation of the proposed solution, by focusing on its main functionality, as well as by highlighting how it perfectly fits the Open Systems Architecture design pattern, thanks to the adoption of both open standards and wide-spread open-source software components. The paper also presents a complete OCO simulation based on the usage of RedHerd to perform a fictitious attack and fully compromise an imaginary enterprise following the Cyber Kill Chain (CKC) phases.

Keywords: cyberspace operations; red teaming; adversarial simulation; penetration testing; orchestration; hacking



Citation: Pecoraro, G.; D'Amico, M.; Romano, S.P. RedHerd: Offensive Cyberspace Operations as a Service. *Signals* **2021**, *2*, 619–636. <https://doi.org/10.3390/signals2040038>

Academic Editor: Jozef Juhár

Received: 9 July 2021

Accepted: 1 September 2021

Published: 1 October 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Cyberspace at its core consists of, but is not limited to, a computerised environment that is artificially constructed and constantly under development. It can be defined as a global domain made up of interconnected communication systems, information technology and other electronic systems and networks, together with their data. Such systems include components that are both separated and independent and which process, store or transmit data [1]. It is like a shifting terrain where all actors have to face its fluidity. New vulnerabilities and opportunities constantly arise as the terrain changes. Entities are in motion; no offensive or defensive capability remains effective indefinitely and no advantage is permanent. A well-defended cyber terrain is attainable, yet it is continuously at risk. Adversary offensive activities persist because opportunity costs are low, and accesses, platforms and payloads can remain useful for extended periods of time [2]. This domain can be described in terms of three layers: physical, logical and cyber-persona. The conduct of Cyberspace Operations (COs), i.e., actions in or through cyberspace intended to preserve friendly freedom of action in cyberspace and/or to create effects to achieve commanders' objectives [1], always includes the logical layer, but may also include activities or elements from the other two layers. The desired effects of COs may exist at all layers or ultimately outside of cyberspace. COs may affect human sense and decision-making and may be used or misused to influence behaviour. Likewise, COs may also affect physical entities outside the three layers. Activities outside of cyberspace which have an effect on cyberspace are not considered COs, e.g., dropping a bomb on Communication Information Systems (CIS) [1].

Systems at the physical layer, e.g., hardware components, are bound to a geographical location. The tangible components in this layer include computers, servers, routers, hubs, switches, wiring and other equipment crucial to data storage, data processing and data

transmission. It also includes the integrated information and communications technology components of other equipment or systems like digital sensors, weapon systems, Command and Control (C2) systems and critical infrastructures [1]. Entities at the logical layer are elements manifested in code or data, such as firmware, operating systems, protocols, applications, and other software and data components. The logical layer cannot function without the physical layer and information flows through either wired networks or the electromagnetic spectrum. The logical layer, along with the physical layer, allows the cyber-persona to communicate and act [1]. The cyber-persona layer does not consist of real persons or organisations but a representation of their virtual identity. A virtual identity could be an email address, a user identifier, a social media account or an alias. Consequently, one person or one organisation can have multiple cyber-personas. Conversely, multiple people or organisations could also create just a single, shared cyber-persona [1]. In this context, the constant innovation of disruptive technologies offers all actors fresh opportunities for exploitation. In this dynamic environment, involved actors (e.g., nations or organizations) must increase their resilience, defend in depth, demand security by design and persistently contest cyberspace threats to generate continuous tactical, operational, and strategic advantages [2].

On the other hand, adversaries exploit and weaponize vulnerabilities to steal wealth and intellectual property, manipulate information, and create malicious software capable of disrupting or destroying systems. This clear intention to project power in and through cyberspace in order to achieve operational or strategic objectives reflects the idea of Offensive Cyberspace Operations (OCO). Offensive capabilities inside cyberspace range from technologically simple means that can be developed rapidly, to sophisticated tools that require a long development period. They can be leveraged to reach tactical effects or achieve strategic impacts, but the required technology level mainly depends on the aimed effects, on the hardening and on the complexity of the target environment. Moreover, preparation time in OCOs might also depend on victim maturity, intelligence gathering efforts, anonymization requirements and any mitigation measures required for controlling collateral damage. Consequently, the period between the decision to create an effect and its actual use may be significantly longer than when using traditional weapons. In relation to this, powerful infrastructure, high-performance instruments and most importantly skilled personnel are the keys for effective OCOs [2]. It is not so easy to recruit skilled cyber operators and involvement of scarce talent may need adjustment of requirements, with possible consequences on the operation plan. Skills are highly perishable, and this has an impact on the management of human resources, requiring constant attention not only to the attraction of new talents, but mainly to the education and re-training of existing personnel. As a result, the engagement in training activities is vital, and a crucial support for reducing the skills gap is the cyber range [3,4]. A cyber range is a platform for the development, delivery and use of interactive simulation environments. A simulation environment is a representation of an organisation's Information and Communication Technology (ICT), Operational Technology (OT), mobile and physical systems, applications and infrastructures, including the simulation of attacks, users and their activities and of any other Internet, public or third-party services which the fictitious environment may depend upon [5]. Cyber ranges are part of an actor's cyber capabilities and can be used for developing the potentiality of security professionals, for the research and development of cyber tools and other assets, and for the continuous delivery of cyber exercises to test those cyber capabilities [5]. It is also expected that cyber ranges will change hiring practices allowing organisations to better identify, validate and hire suitable candidates [5].

Training becomes even more crucial considering that the most dangerous cyber attacks are not randomly performed, but are complex and structured operations. There are a lot of doctrinal and technical references which identify and describe the offensive flow followed by an actor conducting hostile operations, one of the universally recognized is the Cyber Kill Chain (CKC) [6,7]. The CKC represents a systematic approach that includes a wide variety of elements, ranging from malicious payload development to lateral movement

inside a compromised environment [8]. In a complex scenario, the actions on the identified objectives could violate confidentiality, integrity, or availability of a system. Taking into account these offensive purposes, the chain phases can be identified as follows:

1. Reconnaissance: identification, selection, profiling of potential targets and vulnerability discovery via passive and active information gathering. This phase could range from technical activities to Open Source Intelligence (OSINT), passing through social engineering techniques [9] in order to extract unique and high value intelligence by harvesting public records to create a comprehensive profile of certain targets [10];
2. Weaponization: design and implementation of a malware, i.e., a Remote Access Trojan (RAT), a Meterpreter shell [11] or a different backdoor, which exploits a system vulnerability or tricks the user to act as the attacker expects. This malicious software is hidden inside unsuspected files, e.g., a PDF or a Microsoft Office document, in order to evade network and host detection;
3. Delivery: dispatching the weaponized malware to the target environment via spear phishing [12], removable media or using legitimate public facing services which have been compromised, e.g., websites or FTP servers;
4. Exploitation: triggering the malicious code execution on the target by waiting for user interaction, exploiting a specific application or the operating system directly, e.g., via a *zero-day* or other known vulnerabilities;
5. Installation: installation of a backdoor which grants a persistent access to the exploited target's system or network. This phase could involve Windows registry keys manipulation, DLL search order hijacking, DLL side loading, startup folders abuse, scheduled tasks creation or even rootkits installation;
6. Command and Control: connection of the malware previously installed, to the attacker C2 server. At this stage, the malicious implant periodically asks for instructions or sends valuable data. Generally, this communication happens asynchronously and on a predetermined interval, i.e., beaoning;
7. Actions on Objectives: achievement of the final goals, e.g., full system compromise, data integrity violation, service availability denial or contents exfiltration. This stage could include collecting, encrypting and extracting information from the victim environment. In addition, the attacker may also use the initial access as a pivot point to compromise additional systems and move laterally inside the target environment.

It is worth noting that not every attack strictly goes through all the stages of the CKC, e.g., the installation step can be skipped if the effect is immediate. As a matter of fact, time, cost and scope constraints [13] along with knowledge requirements and personnel training represent blocking issues that must be addressed to perform effective OCOs. In relation to this, the present research proposes a novel orchestration solution for offensive activities: the *RedHerd Framework* [14,15]. RedHerd is a quickly employable and easy-to-use open-source software that combines the capability to conduct real offensive operations with the possibility to realize training and simulation activities in a reproduced cyberspace battlefield in which red and blue teams can challenge each other.

This paper is organized as follows: Section 2 provides a complete description of the framework architecture, workflows, interoperability and expansion opportunities, Section 3 describes the experimental methodology and provides a structured test case, Section 4 leverages the experimental analysis to discuss the proposed solution in relation to other state-of-the-art offensive security software, while final considerations and future developments are drawn in Section 5.

Before proceeding further, the authors want to let the reader know that, despite some definitions and references have been extracted from institutional publications and papers, the vision about OCOs does not correspond in any way to the transposition of the doctrine of any particular organization, army or coalition force. The concepts of operation that are introduced are fictitious scenarios with a mere didactic purpose aimed at facilitating the understanding of the contents described. Any deductible Rules Of Engagement (ROE),

jus ad bellum, or jus in bello premise do neither intentionally align with any particular international agreement, nor reflect the authors' affiliation and orientation.

2. Proposed Framework

2.1. Framework Architecture

RedHerd is a collaborative and serverless framework for orchestrating a geographically distributed group of devices in order to conduct red teaming activities and, more in general, complex OCOs. The proposed solution integrates many community acclaimed open-source products, implemented as independent Docker containers, which are fully inter-operable and have been designed to compartmentalise features and to allow horizontal scaling if needed. A high-level representation of the RedHerd framework architecture is provided in Figure 1, where it is possible to clearly identify all of its components.

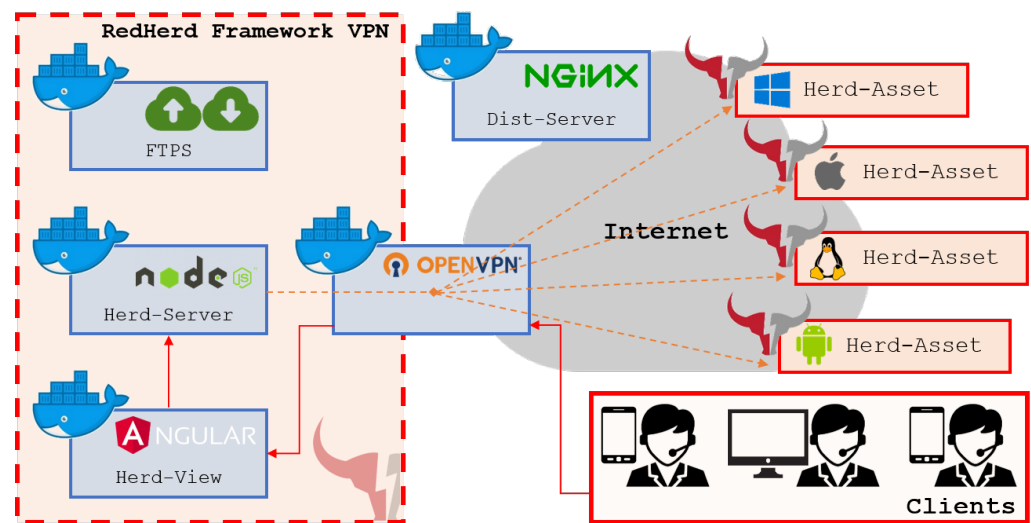


Figure 1. RedHerd Framework.

The assets are multi-platform devices (Windows, Linux, MacOS and Android) that can be orchestrated to perform cyber operations. The Herd-Server represents the Node.js [16] core of the framework and is responsible for interacting with the assets. It receives and multiplexes all the inputs from the operators thanks to an extended set of Application Programming Interfaces (API) and dispatches the output received from the assets via a Socket.IO [17] channel. Assets and operators can easily share files using an FTPS-based File-Server, which allows secure file transfer and storage. The entire framework is bounded by a Virtual Private Network (VPN) in order to guarantee Operations Security (OPSEC) by design, i.e., a systematic process by which potential adversaries can be denied information about the capabilities and intentions of organizations by identifying, controlling, and protecting generally unclassified information that specifically relates to the planning and execution of sensitive organizational activities [18]. In order to be orchestrated, all assets must connect to the OVPN-Server gateway using pre-distributed OpenVPN [19] configuration files. The Distribution-Server is the only role publicly accessible outside the VPN edge and represents an Nginx [20] web server that distributes, after authentication, all the configuration files needed by an asset attempting to join the framework. This architecture allows a high level of automation by granting minimized user interaction during the asset setup procedure. The operational network can be managed via the Herd-View which is a Progressive Web Application (PWA) [21] written in Angular that provides a user-friendly interface to monitor and task all the assets in real time.

The entire solution is cross platform and can be deployed both on premise and in a Cloud-based environment. The prototype implementation described in this paper focuses on a Debian host machine tested both locally and remotely, i.e., by using a commercial Virtual Private Server (VPS). In relation to this, a Bash script has been developed to automate the framework deployment process. Taking into account the design choice to use docker-enabled containerization as the underlying technology, an equivalent script could be easily implemented allowing the framework to be hosted on a different operating system.

2.2. Asset Joining

One of the aspects which required a considerable design and development effort was the asset setup and join procedure. The implementation of this feature has been ruled by two design drivers: high flexibility and low user interaction. The former characteristic is needed in order to grant a remarkable level of compatibility with different operating systems, while the latter is fundamental to minimize failures and reduce the skills required to add a new asset to RedHerd. The result is a manually triggered yet fully automated procedure that involves only the execution of a one-line script which is different for each compatible platform: Bash for Android and Linux (Figure 2), PowerShell for Windows (Figure 3) and Zsh for MacOS.

```
sudo bash -c "curl -k -u $USERNAME:$PASSWORD
https://$PUBLIC_ADDRESS:8443/$ASSET_FINGERPRINT/debian_asset_setup.sh > /tmp/script.sh &&
chmod +x /tmp/script.sh && /tmp/script.sh install && rm -rf /tmp/script.sh"
```

Figure 2. Asset joining one-liner for Linux.

```
$block={
[Net.ServicePointManager]::ServerCertificateValidationCallback={$true};
$wc=New-Object System.Net.WebClient; $basic=[System.Convert]::ToBase64String([System.Text.E
ncoding]::ASCII.GetBytes("$USERNAME"+" "+"$PASSWORD"));
$wc.Headers["Authorization"]="Basic $basic"; $wc.DownloadFile("https://$PUBLIC_ADDRESS:84
43/$ASSET_FINGERPRINT/windows_asset_setup.psml", "script.psml");
Import-Module .\script.psml;
Add-Asset;
Remove-Item .\script.psml;
};
powershell -ep bypass -nop -c $block
```

Figure 3. Asset joining one-liner for Windows.

It can be automatically generated by an operator who wants to add an asset and it requires some identification parameters to work properly: a username, a password and a public address to join the network. When the asset attempts to join RedHerd, a specific fingerprint is automatically calculated as the MD5 hash of the username and is used as its unique identifier. This one-liner interacts with the Distribution-Server and acts as a dropper downloading the full setup script and the related OVPN configuration file. The second stage fully configures the device in order to fulfil the framework requirements, i.e., dependencies management, certificate trusting, firewall and SSH daemon configuration. Then, the VPN connection is initiated and the APIs are used to interact with the Herd-Server and insert the new asset into the framework database. At this point, the asset is effectively part of the framework and so it is completely accessible by the operators.

2.3. Asset Tasking

The asset tasking workflow is a fundamental aspect to be underlined. The interaction between the operator and the Herd-Server, i.e., the upstream channel, is realized by a set of APIs. These APIs are provided by the Herd-Server and are based on the Representational State Transfer (REST) [22] paradigm. In addition, an authentication mechanism is implemented through JSON Web Token (JWT) [23] claims. This system is mainly used by the

Herd-View to assign the tasks, but it has to be intended as a general interface suitable for interacting with the assets through the Herd-Server.

REST is a software architectural style which defines a set of constraints suitable for creating Web services and represents a standardized communication mechanism between computer systems across the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and structured set of stateless operations. The Herd-Server responds to the REST API requests by using the JSend [24] specification in order to offer a consistent JSON (JavaScript Object Notation [25]) response format which is universally recognized and provides an easy way to consume and interact with the framework contents. CRUD operations, i.e., *create*, *read* or *retrieve*, *update*, and *delete*, are performed through their corresponding HTTP verbs:

- POST method to *create* a new item;
- GET method to *retrieve* one or more items;
- PUT method to *update* an existing item;
- DELETE method to *remove* an item.

The Herd-Server goes further, adding the opportunity to trigger the execution of a module using a dedicated REST endpoint and an ad hoc crafted POST request. This feature expands the number of functions offered by a traditional RESTful service, hence introducing the CRUD(E) operations set which integrates the previously described acronym with the *execute* verb. An entity which sends a POST request to run a module, receives a JSend response containing the *sessionid* reflected into the output flow. These results are encapsulated using a specific messaging protocol, in order to be clearly understandable and quickly consumed (see Figure 4).

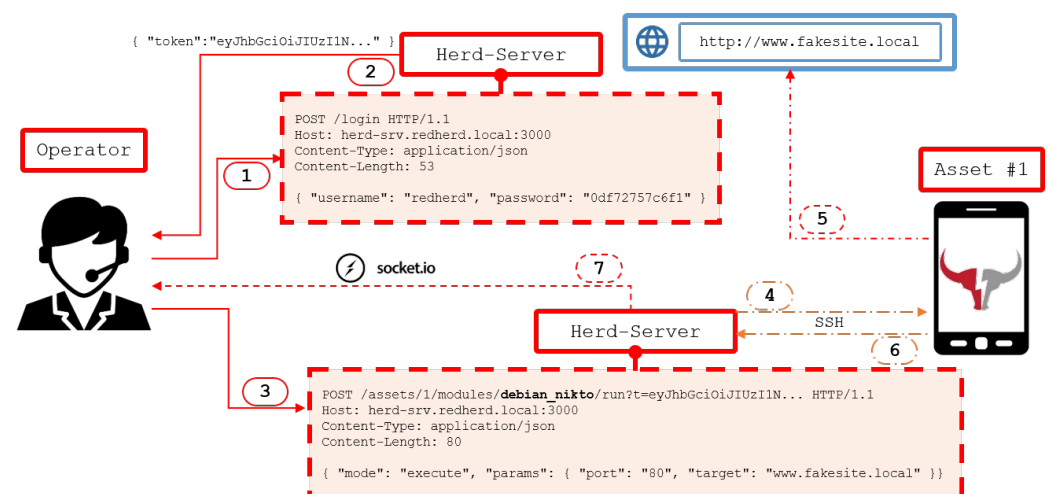


Figure 4. RedHerd tasking flow.

The direct interaction between the Herd-Server and an asset is instead based on the SSH (Secure SHell) protocol. The reason behind this choice is because it natively provides strong authentication and encryption layers and because it is a consolidated protocol offering stability as well as compatibility with the majority of the available operating systems. Last but not least, the usage of SSH leads to the agent-less implementation characterizing RedHerd. Specifically, when the Herd-Server receives a task request for an asset, it initiates an SSH connection with it contextualizing and executing the commands set needed to reach the expected result. This kind of interaction overcomes the needs of a local agent waiting for a task to accomplish and, in addition, allows for a lightweight computational effort asset-side.

2.4. Asset Output

Once the asset has been tasked, the Herd-Server starts to collect each output produced by the executed commands and broadcasts them using custom messages. As shown in Figure 5, these messages are structured on a two-layer model: the first layer (Table 1) contains transport and identification data and wraps the second one which in turn provides the informative content (Table 2).

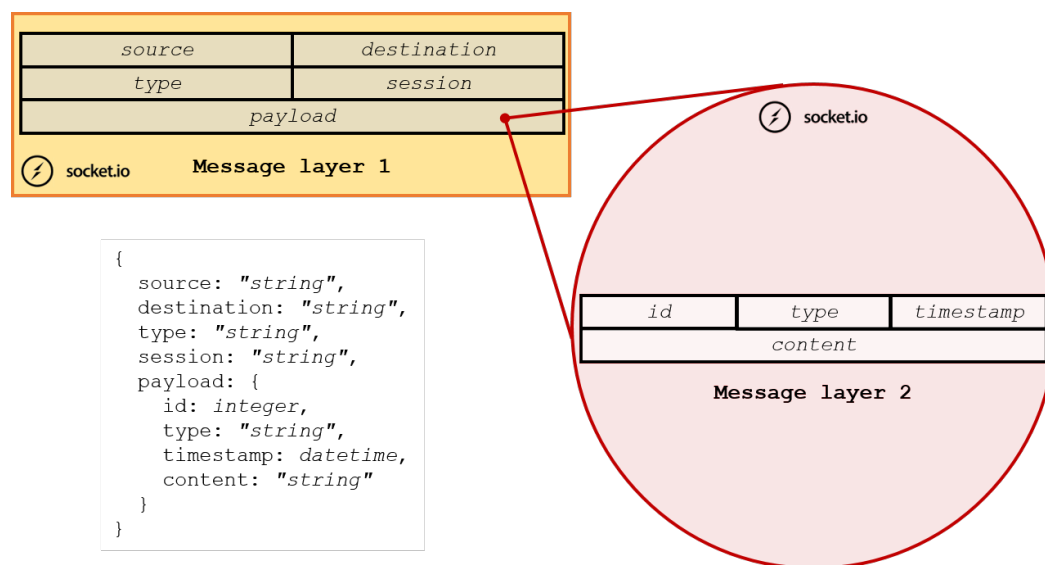


Figure 5. RedHerd message prototype.

This protocol is based on the Socket.IO technology that enables real-time, bidirectional and event-based communication. It is implemented as a JavaScript library that works on every platform, browser or device, focusing equally on reliability and speed. In addition, it primarily uses the WebSocket protocol with HTTP long-polling as a fallback option [17].

Table 1. Level 1 Socket.IO Message.

Field	Description
SOURCE	The generating entity of the message
DESTINATION	The destination entity of the message
TYPE	A message content indicator (e.g., <i>gen_res</i> for general information, <i>cmd_res</i> for command outcomes, <i>syn_res</i> for File-Server sync result)
SESSION	The <i>sessionid</i> assigned to the specific output flow
PAYLOAD	The second layer envelope

Table 2. Level 2 Socket.IO Message.

Field	Description
ID	The unique identifier assigned to the current message
TYPE	A message stream indicator (e.g., <i>stdout</i> , <i>stderr</i> , <i>extcode</i>)
TIMESTAMP	The date and time of the message creation
CONTENT	The informative value of the message

This system is the carrier upon which a custom protocol encapsulates two categories of messages: the *KeepAlive* messages, which provide a real-time status of the assets, and the informative messages containing the tasks results. This message prototype standardizes the data format generated by each asset available in RedHerd. In the same way, the REST-style APIs provide a common way to interact with them. These elements are not only fundamental components of the framework but also valuable interoperability tools. The output carrier messages are sent asynchronously using an Out-of-Band (OoB) channel which represents, from the operator point of view, the downstream communication vector. As shown, each message is tagged with a specific *sessionid* representing a unique identifier used by the requesting entity to link it with the corresponding task. When received, the data inside the message are unwrapped, parsed and presented to the operator interface in a friendly format.

2.5. Task Implementation

Previously in this paper, the term *task* has been intentionally used to indicate the execution of commands by one or more assets and could be intended as a dynamic representation of an operator intent. As a matter of fact, each task needs a static implementation in order to be executed and this implementation, called *module*, can be viewed as the code translation of an operator intent. Furthermore, it is possible to issue some tasks to the Herd-Server itself without involving an asset. This feature is implemented through *services*, which instead are the logical representation of a Herd-Server capability. The list of available services is described in Table 3.

Table 3. Services.

Service	Description
TCP_PROXY	Proxies the TCP traffic from an operator machine to an asset, and vice versa, through the Herd-Server
UDP_PROXY	Proxies the UDP traffic from an operator machine to an asset, and vice versa, through the Herd-Server
HTTP_PROXY	Proxies the HTTP traffic from an operator machine to an asset, and vice versa, through the Herd-Server
RTSP_REDIRECTOR	Redirects the RTSP video stream from an asset to an operator machine through the Herd-Server
TERMINAL	Allows an operator to access an asset console, in a fully interactive mode, using a Web browser

Therefore, if the implementation of an action performed by an asset is a module, its execution is the corresponding task. A set of modules carrying out similar tasks is grouped into a common *topic*, which is the abstract representation of one or more characteristics bringing together a number of tasks and, transitively, one or more modules. In order to adhere to a structured offensive methodology, RedHerd provides for each supported operating system, a built-in collection of topics that follows all the steps of the CKC [6], e.g., *debian_reconnaissance*, *windows_delivery*. Anyway, the framework APIs allow the definition of new topics in relation to the operational needs. As a result, the terms flexible and expandable perfectly fit the designed behavior since in RedHerd it is possible to create new modules in order to add more features and accomplish more tasks.

Every module is made up of two elements: an *info* file which contains its metadata (see Figure 6) and a JavaScript file which contains the execution logic (see Figure 7). In order to allow a rapid development, the framework provides a set of JavaScript base modules containing specific superclasses. These classes are implemented for each compatible operating system, e.g., Windows and Linux, and can be used to inherit all the required properties and methods needed to interact with the assets.

```

{
  "name": "debian_theharvester",
  "title": "theHarvester",
  "description": "Passive information gathering about a domain or a company",
  "binary": "python3",
  "author": "b4gh33r4 & peco602",
  "topic": "debian_reconnaissance",
  "version": "1.0",
  "params": [{"label": "Domain", "name": "domain", "type": "string"}],
  "tags": ["automatic"]
}

```

Figure 6. Example of module metadata.

```

class DebianTheHarvester extends LinuxModule
{
  constructor(asset, context, session, wsServer, token)
  {
    super(asset, "debian_theharvester", session, wsServer, token);
    this.domain = context.domain;
  }

  configure(whatIf = false)
  {
    let task = "sudo apt update; \
      sudo apt install python3 python3-pip -y && \
      sudo rm -rf theHarvester && \
      git clone https://github.com/laramies/theHarvester && \
      cd theHarvester && \
      sudo python3 -m pip install -r requirements/base.txt"

    this.do(task, "cmd_res", false, whatIf);
  }

  run(whatIf = false)
  {
    if (this.validate())
    {
      let task = "cd theHarvester && python3 theHarvester.py -b all -d " +
        this.domain;
      this.do(task, "cmd_res", false, whatIf);
    }
    else
    {
      this.reportAndExit(this.buildErrorMessage("Invalid input provided"));
    }
  }

  validate()
  {
    if ((this.domain) && ((Validator.validateName(this.domain) ||
      Validator.validateHostname(this.domain))))
    {
      return true;
    }
    return false;
  }
}

module.exports = DebianTheHarvester

```

Figure 7. Example of module implementation.

Specifically, each module class, deriving from one of the base classes, must have a constructor and must expose at least three methods: *run* (or one of its alternatives *interact* and *pivot*), *configure* and *validate*. The *run* method is called to start an automatic module execution, *interact* is used if the module requires an operator interaction, while *pivot* is applicable if the module needs user interaction but combined with a redirection to an external destination, e.g., a third-party website. The *configure* method is responsible for

performing all the preliminary activities, while the *validate* method verifies the parameters passed to the module in order to check for incorrect input data. By complying with the provided interface, it is extremely easy to extend and/or enhance the capabilities offered by the framework.

3. Materials and Methods

This section presents an OCO simulation consisting of a fictitious attack which aims to fully compromise a target enterprise environment following the CKC phases [6]. Table 4 lists the hardware involved during the current experiment. The Infrastructure Server is the system on which the RedHerd framework has been deployed. It hosts the core components and orchestrates the assets required for the mission accomplishment.

Table 4. Inventory of materials.

	Infrastructure Server	C2 Server	Drone
Platform	VPS	VPS	Raspberry Pi 4b
Operating System	Ubuntu 20.04	Ubuntu 20.04	Raspberry Pi OS
Processor	4 vCore	1 vCore	4 core (ARM)
Memory	8 GB	2 GB	4 GB
Storage	160 GB (SSD)	40 GB (SSD)	32 GB (SD card)

One of them is a C2 Server that essentially operates as the landing point for the malware implanted during the exploitation. It allows to manage the compromised machines, sending tasks and receiving results. The second asset is a specifically designed drone, which represents a mobile device used to reach remote locations and perform on-demand actions. The usage of a drone for this experimental phase has been mainly driven by anonymization purposes but it also demonstrates how easily RedHerd can be interfaced with non-conventional hardware. The drone is a quad-copter based on a Raspberry Pi 4 model B [26], equipped with a Navio2 Emlid flight controller. The Navio2 is an autopilot shield developed as a Linux version of Ardupilot board [27]. The quad-copter can be controlled via a 900 MHz telemetry/telecommand module and is able to perform autonomous flight missions thanks to an onboard GPS receiver. It can join the RedHerd framework via a 4G-dongle Internet connection and is provided with a Wi-Fi Alfa antenna to perform wireless attacks, as shown in Figure 8.

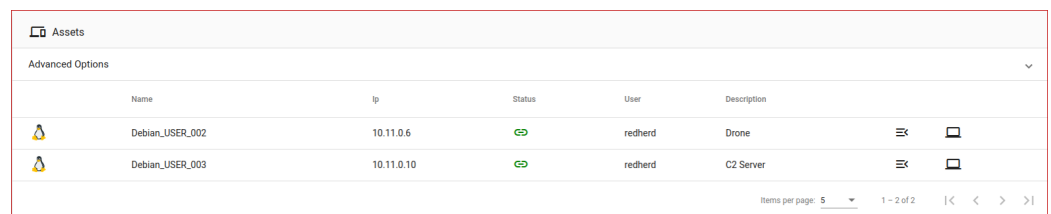










Figure 8. Top-view of the drone asset.

It is worth noting how the drone is completely based on Commercial Off-The-Shelf (COTS) products and has been assembled with a budget of almost 300 euros. Figure 9 provides a Herd-View capture showing the list of available assets, i.e., the drone and the C2 server. As the reader can observe, they have, respectively, acquired the IP addresses 10.11.0.6 and 10.11.0.10, which have been distributed after the devices joined the framework VPN. By employing the Herd-View, the operator has complete control over the available assets and can easily use them to fulfil the assigned tasks.

The end goals of this simulation are the full access to the IT environment of the *Fake Corp* imaginary enterprise and the acquisition of its sensitive data. In order to reduce the chances of cyber-attribution, i.e., the process of tracking, identifying and laying blame on the perpetrator of a cyberattack or other hacking activity, the operator can fly the drone as close as necessary to a third-party Wi-Fi Access Point (AP), gain access to it by obtaining and cracking the pre-shared key and then use a non-attributable Internet access to perform the subsequent mission steps. The Wi-Fi network WPA2 protection is easily exploitable [28] launching the RedHerd *Airgeddon* [29] module, which:

1. Deauthenticates all clients connected to the target AP, i.e., performs a so called *deauthentication attack*;
2. Captures the 4-way authentication handshake for the AP;
3. Cracks the WPA2 pre-shared key using a dictionary.



Assets						
Advanced Options						
	Name	Ip	Status	User	Description	
	Debian_USER_002	10.11.0.6		redherd	Drone	 
	Debian_USER_003	10.11.0.10		redherd	C2 Server	 

Items per page: 5 1 - 2 of 2 |< > >|

Figure 9. List of assets joined to the RedHerd Framework.

Once the operator has obtained this OPSEC compliant Internet access, he can start the preliminary phase of the engagement, i.e., the **reconnaissance**. Adversaries may execute active scans to gather detailed information about the attack surface exposed by the target or more simply perform passive information gathering without involve direct interaction with it. In this case the operator launches the *theHarvester* [30] module to scrape search engines and social networks, and retrieve as much information as possible about the target enterprise, e.g., some e-mail addresses related to its domain *fakecorp.local*. It must be emphasized that RedHerd allows a low-skilled operator to perform the task without deeply knowing the syntax and the logic of this reconnaissance tool. In fact, as show in Figure 10, the Herd-View generates the module Graphical User Interface (GUI) dynamically and in accordance with the metadata described in Figure 6, producing an HTML text input box for the *domain* string parameter. Input data are automatically validated by the Herd-Server through the *validate* method in Figure 7, in order to avoid unexpected results and block malicious intents, i.e., command injection. Once the module has completed its job, it provides the operator with the email address *helpdesk@fakecorp.local*, which is an enabling result for the subsequent operations.

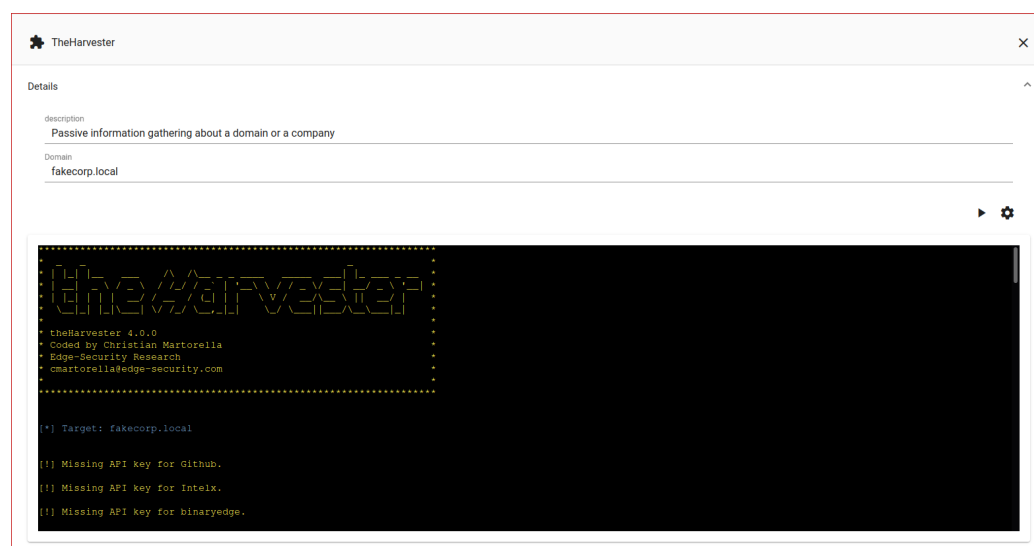


Figure 10. theHarvester module execution.

At this stage the *Social Engineer Toolkit* [31,32] covers either the **weaponization** and the **delivery** CKC phases by forging and sending a spear-phishing email with an attachment containing a malicious payload, which in turn will *call home* and land on the C2 Server. It is important to underline that, up to this point, all the activities have been performed through the initially compromised Internet access, reducing the probability of being caught. Employing the Herd-View, it is trivial to use the C2 Server asset and launch the Metasploit module in order to create a multi-handler [11] listener and wait for the exploit connection.

As soon as the target employee checks their mailbox and opens the malicious attachment, the attacker gets an interactive shell on the help-desk machine. In this way, he can perform the **installation** phase by uploading an executable file and, manipulating the Windows registry, he can implant a custom *autorun* key obtaining a persistent access (see Figure 11). The RedHerd interactive Metasploit module allows complete **command and control** over the compromised systems.

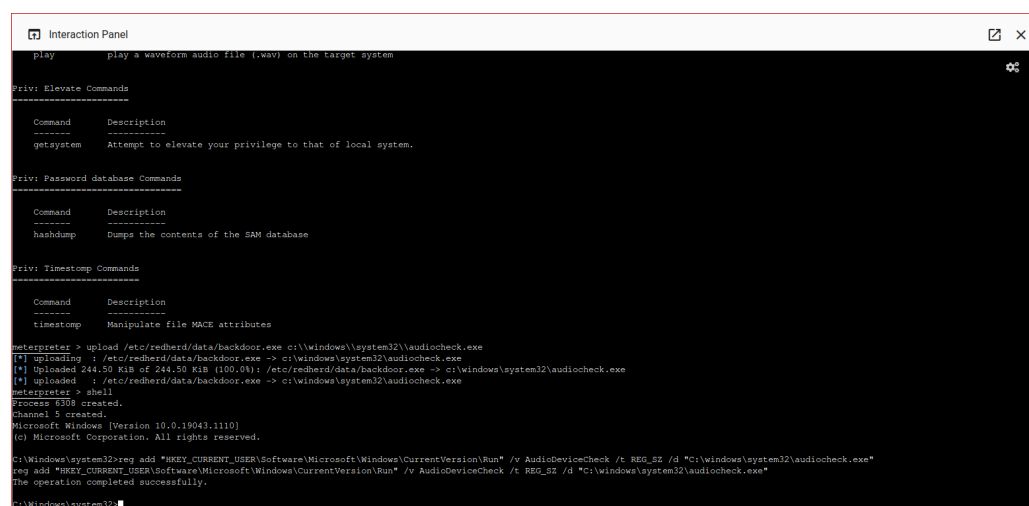


Figure 11. Malware persistence installation on the target.

Specifically, the operator can perform a wide range of **actions on objectives** but, for the purpose of this simulation, he violates the corporate data confidentiality subtracting the file named *customers_data.xlsx* (Figure 12). This exfiltration is quickly realized downloading the file on the C2 Server and making it available mission wide using the RedHerd File Manager (Figure 13).

```

Microsoft Windows [Version 10.0.19042.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
c:\users\helpdesk

C:\Windows\system32>cd c:\users\helpdesk\documents
cd c:\users\helpdesk\documents

C:\Users\helpdesk\Documents>dir
dir
Volume in drive C has no label.
Volume Serial Number is 1007-39F9

Directory of c:\Users\helpdesk\Documents
07/08/2021 12:36    <DIR>          .
07/08/2021 12:36    <DIR>          ..
07/08/2021 12:36             12,012,000 customers_data.xlsx
               1 File(s)      12,012,000 bytes
               2 Dir(s)      5,617,852,416 bytes free

C:\Users\helpdesk\Documents>exit
exit
meterpreter > download c:\users\helpdesk\documents\customers_data.xlsx /etc/redherd/data
[*] Downloading: c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 1.00 MiB of 11.46 MiB (8.73%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 2.00 MiB of 11.46 MiB (17.46%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 3.00 MiB of 11.46 MiB (26.19%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 4.00 MiB of 11.46 MiB (34.92%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 5.00 MiB of 11.46 MiB (43.65%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 6.00 MiB of 11.46 MiB (52.38%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 7.00 MiB of 11.46 MiB (61.11%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 8.00 MiB of 11.46 MiB (69.84%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 9.00 MiB of 11.46 MiB (78.57%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 10.00 MiB of 11.46 MiB (87.29%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 11.00 MiB of 11.46 MiB (96.02%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Downloaded 11.46 MiB of 11.46 MiB (100.00%): c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
[*] Download : c:\users\helpdesk\documents\customers_data.xlsx -> /etc/redherd/data/customers_data.xlsx
meterpreter >

```

Figure 12. Sensitive file download from the target.

With this final action the attackers have achieved their objectives and then they can choose to completely destroy the RedHerd instance or keep it active in order to maintain, and eventually expand, the foothold inside the target enterprise for possible future operations.



Figure 13. Herd-View File Manager.

4. Results and Discussion

The solution proposed in this paper is one of a kind, so it is quite difficult to find other state-of-the-art products that cover the same scope of the RedHerd framework. As experimentally demonstrated in Section 3, RedHerd promotes collaboration between OCO actors and embraces strong orchestration capabilities. In addition, the framework is able to cover all the CKC phases, from the reconnaissance up to the full target compromise, i.e., violating the confidentiality, integrity and availability of its data and services (see Figure 14).

For the sake of completeness, the offensive security panorama offers a wide range of tools, all of them support a set of functionalities that represents a complement and not an alternative to the proposed solution. One of the most common attack tool is the already mentioned Metasploit Framework. It is a Ruby-based, modular penetration testing platform that enables an attacker to write, test, and execute exploit code. It contains a suite of tools employable to test security vulnerabilities, enumerate networks, execute attacks, and evade detection. At its core, the Metasploit Framework is a collection of commonly used tools that provides a complete environment for penetration testing and exploit development [33]. It also offers the Meterpreter shell, which is essentially an attack platform that gets injected into the memory of a running process in order to avoid detection by Host-based Intrusion Detection System (HIDS), as well as bypass the limitations of the operating system's native command shell [11].

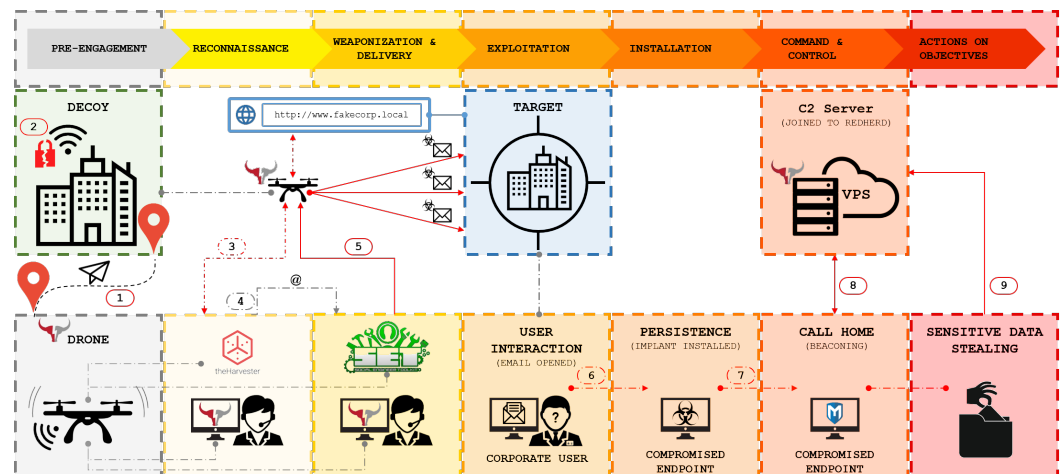


Figure 14. Attack simulation flow.

Metasploit, when combined with the Meterpreter shell, also provides some basic post-exploitation features, i.e., the capability to use an existing access to pivot from one machine to another expanding the compromised surface and adding valuable footholds for the mission prosecution [34]. Anyway, actions related to the post-exploitation phase are usually performed using specific software defined as C2 frameworks, e.g., Covenant [35], Cobalt Strike [36], PowerShell Empire [37] and many others [38]. The aim of RedHerd is not to replace these products, but to multiply their magnitude collecting them in a powerful and centralized solution where complex objectives can be achieved in reasonable time, with affordable costs and even with minimised knowledge requirements. Another important aspect to take into account is that the acquisition of code for the exploitation of common vulnerabilities is quite easy. On one hand, there are tons of public websites hosting Proofs of Concepts (PoC) and exploits, e.g., ExploitDB [39] and GitHub [40], on the other hand, in case of structured engagements, attackers inevitably need in-house script development. In relation to this, the present paper has underlined several times how trivial is the implementation of custom RedHerd modules. Developers can wrap and integrate almost any kind of script or tool into the proposed framework and then operators just have to push a button. These considerations lead to the reason behind the title of the paper: Offensive Cyberspace Operations as a Service (OCOaaS). The OCOaaS model involves a complete software solution, locally set up, remotely deployed or Cloud-based, which allows organizations to conduct OCOs employing an uncountable number of devices and low-skilled operators. It offers a layer of abstraction placed in front of the operative infrastructure and tools. In this way, the operational actors have the opportunity to focus on the task execution, while ignoring all of the collateral activities. In addition, OCOaaS requires a flexible and quickly deployable solution to conduct OCOs at a minimal cost. This paradigm expands the discussion and opens a different yet close perspective. These days, the idea behind OCOaaS would find fertile ground in a wide variety of use cases. In this regard, a mention to the emerging concept of Internet of Battle Things (IoBT) does offer an opportunity to put the spotlight on the potential application of the model and transitively on a possible RedHerd employment. The idea of IoBT encapsulates commercial IoT, cyber-battle things and human individuals. The interconnected instances of these entities represent the nodes of a network and may be considered as a set of IoBT assets. This ecosystem could occur on private or public networks, on the Cloud, or within any type of military or civilian enclave. In this scenario, Human operators, along with their cyber-personas, could participate assisted by, collaborating with or assigning tasks to IoBT devices. The complexity introduced in this context along with the various abstractions needed to consume notifications, alerts, and task results provided in real-time and in diversified formats, opens the undisputed necessity of a properly structured system for IoBT configuration, control, orchestration, mission and security management [41].

The RedHerd framework, as the practical implementation of the OCOaaS model, fulfil all the described requirements by promoting the collaboration between the involved actors, and empowering the approach with strong orchestration capabilities. It must be emphasized that the majority of these challenging aspects correspond to the purpose and motivation of the current research and represent valuable reference points which have contributed to inspire this work. Despite the high flexibility and expandability of RedHerd, this framework has also some drawbacks that are important to identify and underline. First of all, offensive tools are extremely heterogeneous, i.e., some are API-based, many have their own database and others need a standalone client, making quite demanding a complete integration within RedHerd. As an example, Metasploit embeds a PostgreSQL database to store all data collected during an engagement, e.g., IP addresses, discovered open ports, gathered credentials, while Covenant is based on SQLite. Even if RedHerd allows full control over the mentioned products, it is still not able to automatically correlate the data they collect. Moreover, public tools are updated on a daily basis so it can become quite difficult to keep all their corresponding modules constantly aligned, in particular if there are breaking changes. In order to overcome this limitation, the suggested approach is to perform frequent maintenance of the mostly used modules, e.g., port scanning, phishing, C2, and then review and prepare more targeted modules in proximity of an engagement.

5. Conclusions and Future Work

This paper introduces the OCOaaS model as a complete software solution, locally set up, remotely deployed or Cloud-based, which allows organizations to conduct Offensive Cyber Operations by offering a layer of abstraction placed on top of the operational infrastructure and tools. The RedHerd framework, a collaborative and serverless orchestration platform, represents a close implementation of the proposed model. This framework designates interoperability, expandability and ease of use as its main features. The usage of REST-style APIs and Socket.IO offers a straightforward interfacing mechanism that allows third party applications to easily communicate with the product while granting a high level of interoperability. The same idea might be applied to the task implementation which relies upon a well defined JavaScript structure offering an easy way to expand the product features by writing custom code and accomplishing an uncountable number of offensive tasks. Moreover, the friendly and dynamically generated GUI allows even low-skilled operators to actively contribute in conducting complex cyberspace operations while ignoring all the collateral activities and guaranteeing at the same time a low cost of deployment and maintenance. The experimental phase has demonstrated the capability of RedHerd to integrate and maximize the functionalities of several publicly available offensive tools to perform all the CKC stages in a complex OCO.

The achieved results are promising and open the way to future framework improvement. Tools heterogeneity could be further overtaken by introducing some kind of module output parsing and automatic RedHerd model population with juicy data, e.g., gained credentials. These data could be potentially reused during subsequent phases of the offensive engagement. In addition, the computational power of all the managed assets could be combined in order to perform more demanding tasks, e.g., Distributed Denial of Service (DDoS) and password cracking, or even stealthier actions, e.g., distributed port scanning. Last but not least, framework performance will be analyzed in relation to the number of joined assets and, in particular, in case of stress conditions, i.e., multiple modules performing resource intensive tasks, in order to better determine hardware requirements and find undiscovered bottlenecks.

Author Contributions: Conceptualization, G.P. and M.D.; Methodology, G.P. and M.D.; Software, G.P. and M.D.; Supervision, S.P.R.; Writing—original draft, G.P. and M.D.; Writing—review and editing, S.P.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

AP	Access Point
API	Application Programming Interfaces
ARM	Advanced RISC Machine
C2	Command and Control
CKC	Cyber Kill Chain
CIS	Communication Information Systems
CO	Cyberspace Operations
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
GUI	Graphical User Interface
HIDS	Host-based Intrusion Detection System
HTML	HyperText Markup Language
ICT	Information and Communication Technology
IoBT	Internet of Battle Things
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
JWT	JSON Web Token
OCOaaS	Offensive Cyberspace Operations as a Service
OCO	Offensive Cyberspace Operations
OoB	Out-of-Band
OPSEC	Operations Security
OS	Operating System
OSINT	Open Source Intelligence
OT	Operational Technology
PoC	Proof of Concept
PWA	Progressive Web Application
RAM	Random Access Memory
RAT	Remote Access Trojan
REST	Representational State Transfer
RISC	Reduced Instruction Set Computer
ROE	Rules Of Engagement
RTSP	Real Time Streaming Protocol
SD	Secure Digital
SSD	Solid State Drive
SSH	Secure SHell
TTP	Tactics, Techniques and Procedures
VPN	Virtual Private Network
VPS	Virtual Private Server
WPA2	Wireless Protected Access 2

References

1. North Atlantic Treaty Organization. *AJP 3-20, Allied Joint Doctrine for Cyberspace Operations*; NATO Standardization Office (NSO): Brussels, Belgium, 2020.
2. Dalmjin, A.; Banse, V.; Lumiste, L.; Teixeira, J.; Balci, A. *Cyber Commanders' Handbook*; NATO CCDCOE Publications: Tallinn, Estonia, 2020.
3. National Institute of Standards and Technology. *The Cyber Range: A Guide (Draft)*; NIST: Gaithersburg, MD, USA, 2020.

4. Grigoriadis, A.; Darra, E.; Kavallieros, D.; Chaskos, E.; Kolokotronis, N.; Bellekens, X. Technology Development for Security Practitioners. In *Security Informatics and Law Enforcement*; Chapter Cyber Ranges: The New Training Era in the Cybersecurity and Digital Forensics World; Springer: Cham, Switzerland, 2021; pp. 97–117. [\[CrossRef\]](#)
5. European Cyber Security Organization. *Understanding Cyber Ranges: From Hype to Reality*; ECS: Brussels, Belgium, 2020.
6. Hutchins, E.; Cloppert, M.; Amin, R. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. *Lead. Issues Inf. Warf. Secur. Res.* **2011**, *1*, 80.
7. Bahrami, P.N.; Dehghantanha, A.; Dargahi, T.; Parizi, R.M.; Choo, K.R.; Javadi, H.H.S. Cyber Kill Chain-Based Taxonomy of Advanced Persistent Threat Actors: Analogy of Tactics, Techniques, and Procedures. *J. Inf. Process. Syst.* **2019**, *15*, 865–889. [\[CrossRef\]](#)
8. Quintero-Bonilla, S.; del Rey, A.M. A New Proposal on the Advanced Persistent Threat: A Survey. *Appl. Sci.* **2020**, *10*, 3874. [\[CrossRef\]](#)
9. Krombholz, K.; Hobel, H.; Huber, M.; Weippl, E. Advanced social engineering attacks. *J. Inf. Secur. Appl.* **2015**, *22*, 113–122. [\[CrossRef\]](#)
10. Tabatabaei, F.; Wells, D. Open Source Intelligence Investigation: From Strategy to Implementation. In *Advanced Sciences and Technologies for Security Applications*; Chapter OSINT in the Context of Cyber-Security; Springer: Cham, Switzerland, 2016; pp. 213–231. [\[CrossRef\]](#)
11. Maynor, D.; Mookhey, K. *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*; Elsevier: Amsterdam, The Netherlands, 2007. [\[CrossRef\]](#)
12. Aleroud, A.; Zhou, L. Phishing environments, techniques, and countermeasures: A survey. *Comput. Secur.* **2017**, *68*, 160–196. [\[CrossRef\]](#)
13. Atkinson, R. Project management: Cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *Int. J. Proj. Manag.* **1999**, *17*, 337–342. [\[CrossRef\]](#)
14. D’Amico, M.; Pecoraro, G.; Romano, S.P. RedHerd Framework. Available online: <https://github.com/redherd-project/redherd-framework.git> (accessed on 6 August 2021).
15. D’Amico, M.; Pecoraro, G.; Romano, S.P. RedHerd Documentation. Available online: <https://redherd.readthedocs.io/en/latest/> (accessed on 6 August 2021).
16. OpenJS Foundation Node.js. Available online: <https://nodejs.org> (accessed on 26 May 2021).
17. Rauch, G. Socket.IO. Available online: <https://socket.io/> (accessed on 26 May 2021).
18. Joint Task Force Transformation Initiative Interagency Working Group. *Security and Privacy Controls for Information Systems and Organizations*; Technical Report NIST Special Publication (SP) 800-53, Rev. 5, Includes Updates as of 10 December 2020; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2020.
19. OpenVPN. Available online: <https://openvpn.net> (accessed on 30 May 2021).
20. Nginx. Available online: <https://www.nginx.com/> (accessed on 26 May 2021).
21. Angular. Available online: <https://angular.io> (accessed on 26 May 2021).
22. Fielding, R.T. Representational State Transfer (REST). Available online: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (accessed on 24 May 2021).
23. Jones, M.; Bradley, J.; Sakimura, N. JSON Web Token (JWT). RFC 7519, RFC Editor. 2015. Available online: <https://www.rfc-editor.org/rfc/rfc7519.txt> (accessed on 7 August 2021).
24. JSend. Available online: <https://github.com/omniti-labs/jsend> (accessed on 24 May 2021).
25. Bray, T. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259, RFC Editor. 2017. Available online: <https://www.rfc-editor.org/rfc/rfc8259.txt> (accessed on 7 August 2021).
26. Raspberry Pi 4. Available online: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/> (accessed on 7 August 2021).
27. Navio2 Board. Available online: <https://navio2.emlid.com/> (accessed on 7 August 2021).
28. Tutorial: How to Crack WPA/WPA2. Available online: https://www.aircrack-ng.org/doku.php?id=cracking_wpa (accessed on 7 August 2021).
29. Airgeddon. Available online: <https://github.com/v1s1t0r1sh3r3/airgeddon> (accessed on 7 August 2021).
30. Martorella, C. theHarvester. Available online: <https://github.com/laramies/theHarvester> (accessed on 7 August 2021).
31. Engebretson, P. The Basics of Hacking and Penetration Testing (Second Edition). In *Ethical Hacking and Penetration Testing Made Easy*; Chapter Social Engineering; Syngress: Rockland, MA, USA, 2013; pp. 127–140. [\[CrossRef\]](#)
32. Kennedy, D. The Social-Engineer Toolkit (SET). Available online: <https://github.com/trustedsec/social-engineer-toolkit> (accessed on 7 August 2021).
33. Metasploit. Available online: <https://www.rapid7.com/products/metasploit> (accessed on 30 May 2021).
34. Andress, J.; Linn, R. *Coding for Penetration Testers*; Elsevier: Amsterdam, The Netherlands, 2012. [\[CrossRef\]](#)
35. Cobb, R. Covenant. Available online: <https://github.com/cobbr/Covenant> (accessed on 30 May 2021).
36. Cobalt Strike. Available online: <https://www.cobaltstrike.com/> (accessed on 30 May 2021).
37. PowerShell Empire. Available online: <https://www.powershellempire.com/> (accessed on 30 May 2021).
38. Orchilles, J.; Bort, B.; Mashinchi, A. The C2 Matrix. Available online: <https://www.thec2matrix.com/matrix> (accessed on 26 May 2021).
39. Exploit Database. Available online: <https://www.exploit-db.com> (accessed on 30 May 2021).

-
40. GitHub. Available online: <https://github.com> (accessed on 30 May 2021).
 41. Erdelyi, B.; Ahiskali, M.B. Management and Orchestration of Autonomous Cyber Things. Available online: <https://apps.dtic.mil/sti/pdfs/AD1106124.pdf> (accessed on 26 May 2021).