Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS EU 2024 - Selected Papers from the 11th Annual Digital Forensics Research Conference Europe

Hypervisor-based data synthesis: On its potential to tackle the curse of client-side agent remnants in forensic image generation

Dennis Wolf[a,b,*], Thomas Göbel[b], Harald Baier[b]

[a] *Central Office for Information Technology in the Security Sector (ZITiS), Zamdorfer Straße 88, Munich, Bavaria, Germany*
[b] *University of the Bundeswehr Munich, Werner-Heisenberg-Weg 39, Neubiberg, Bavaria, Germany*

ARTICLE INFO

Keywords:
Forensic image generation
Data synthesis
Data generation
Data synthesis framework
Data sets
Digital corpora
Agent-less approach
Linux forensics

ABSTRACT

In the field of digital forensics, the number and heterogeneity of devices typically involved in an investigation is increasing. In order to train digital forensics practitioners and make faster progress in the development and validation of forensic tools, the demand for up-to-date data sets is high. However, manually creating data sets is a complex, tedious, and time-consuming task increasing the need for automated solutions. Existing data generation frameworks typically use components that run directly on the simulated client (e.g., a client-side agent controlled via SSH). On the one hand, this facilitates simulation by providing direct feedback from the client and the ability to use client-side libraries to access software. On the other hand, however, this approach creates unintended traces in the generated data sets that quickly reveal their synthetic origin and affect their realism and thus their relevance. To avoid such traces, this paper presents a hypervisor-based solution to eliminate such a client-side software component in a recent digital forensic data set generator, while compensating for its absence only through host-side means. To demonstrate the practicability of the proposed approach as well as the indistinguishability of the generated traces, a multi-participant scenario is performed as a proof of concept to replicate a realistic attack scenario on a Linux system from a Kali attacker machine. During the evaluation, the generated data set is compared in terms of unintended traces and realism to a data set generated by the same framework using an agent component. In this way, we demonstrate the benefits and overall usefulness of an agent-less data synthesis approach.

1. Introduction

The demand for up-to-date and publicly available forensic data sets is high. In this paper, the definition from Grajeda et al. (2017) is used, who describe Digital Forensic (DF) data sets "as a collection of related, discrete items that [have] different meanings depending on the scenario and [were] utilized for some kind of experiment or analysis". The authors point out limitations with existing data sets and the surrounding ecosystem for distribution. Many of them are not publicly available due to various reasons: some researchers lack the capability of sharing the data, others have privacy concerns, and last but not least some have not thought about sharing data sets at all. According to the evaluation of Grajeda et al. (2017), they found more missing aspects regarding the *variety*, *timeliness*, and *maintenance*.

However, data is needed for various reasons in the cybersecurity field. For example, to enable (aspiring) digital forensics practitioners to train their investigation skills and get used to different operating systems and devices with their particular features (Garfinkel, 2010), and the broad spectrum of DF artifacts (Garfinkel et al., 2009). To make education and training more effective, the provided data must be complex and realistic enough to imitate real cases.

Driven by the accumulating mass, heterogeneity and complexity of data in a DF investigation the demand to employ artificial intelligence and machine learning supported software in the forensic area is high (Mitchell, 2014; Mohsin, 2021; Garfinkel, 2012; Jarrett and Choo, 2021). This therefore increases the demand for standardized, potentially labeled, and realistic data sets further, since these systems have to consume a huge amount of labeled data to be trained in finding correlations and recognizing patterns. Besides this rather new branch of software in the DF context, classic software tools also require large masses of data to conduct excessive validation of their functionality to show their reliability and make them, e.g., admissible in court (Carrier, 2002).

Due to the reasons given above and the status quo, it becomes clear

* Corresponding author. Central Office for Information Technology in the Security Sector (ZITiS), Zamdorfer Straße 88, Munich, Bavaria, Germany.
E-mail addresses: dennis.wolf@zitis.bund.de (D. Wolf), thomas.goebel@unibw.de (T. Göbel), harald.baier@unibw.de (H. Baier).

https://doi.org/10.1016/j.fsidi.2023.301690

that it is not feasible to provide standardized and contemporary data in a large-scale manner without using computer-aided generation. This generation, as shown in Section 2 during our analysis of existing data synthesis frameworks, often utilizes client-side software to realize the actual user interactions on the client. This often leads to unintended artifact remnants during the data synthesis process, tainting the generated data set and thus decreases its realism and applicability. Some frameworks try to delete these traces subsequently. However, this happens more or less reliably. It is obvious that the initial prevention of traces is superior compared to their subsequent deletion.

1.1. Contribution

This work covers the following contributions:

- Presentation of existing concepts and solutions to compensate for client-side software during data synthesis.
- Analysis of issues and challenges when omitting client-side software and the respective compensation.
- Provision of a reference implementation and a suitable proof of concept for Linux systems to demonstrate the usefulness of our agent-less approach.
- Setting the resulting agent-less data set in comparison to a conventionally created one.

The source code of our agent-less data synthesis approach can be downloaded from: https://github.com/dasec/ForTrace/tree/agentless-fortrace.

1.2. Paper outline

The remainder of the paper is structured as follows: Section 2 discusses related work. It provides a brief overview of general means for the data sets and their generation in DF, a discussion about a recent proposal for an agent-less data synthesis solution, and a comparison of existing data synthesis frameworks. This leads us to the decision whether to develop a completely new framework or to expand existing software to meet our requirements. Section 3 discusses the necessary changes brought to the selected framework, in order to compensate for the removal of any client-side software and the sole reliance in host-side processing and control. In Section 4, we present our Linux attack scenario as a proof of concept. However, the proposed approach works completely OS-independent and a transfer to Windows or other operating systems is easily possible. Afterwards, in Section 5, the generated data set is evaluated and compared to a conventionally generated one, so both the advantages and drawbacks of our agent-less concept become clear. Finally, Section 6 concludes this work and provides an outlook to further increase the utility and usability of the framework.

2. Related work

This section covers related work in the field of DF, reaching from general possibilities for data sets, over a recent approach for the removal of client-side software, to the evaluation of existing data synthesis frameworks in order to determine a potential candidate for an agent-less implementation.

2.1. General aspects of data sets in digital forensics

The generated data sets have to contain DF artifacts, which Casey (2004) defines as “any data stored or transmitted using a computer that support or refute a theory of how an offense occurred or that address critical elements of the offense such as intent or alibi”.

Moch and Freiling (2009) stated that there are three well-known and tested approaches to manually obtain DF-relevant training images for digital investigators. Originally, they only wrote about the approaches in the context of hard disk image generation.

The **manual generation** is a widespread approach, resulting in potentially high quality data sets with DF-relevant traces that can be used in training courses or workshops. The creator of the data set knows the ground truth, assuming he/she documents the creation thoroughly and conscientiously, and is the least constrained in creating it – compared to all other approaches. Nevertheless, the creation itself is rather time-consuming, which hinders the (large-scale) creation of new data sets drastically. Once such a data set is created, it is static and cannot be changed afterwards. There are some examples of manually created images, e.g., the Digital Corpora of Garfinkel et al. (2009) which includes image and RAM dumps, or the CFReDS portal from NIST.[1]

For the second approach, the **utilization of honeypots**, a computer is attached to the Internet, to be attacked and compromised. Lin et al. (2014), for example, developed an active honeypot system and made it open-source. Since the result is based on illegal activities there are no real privacy restrictions and the data sets can be shared with the DF community. Unfortunately, the amount of generated data is limited due to high hardware requirements and the ubiquitous random component of whether the generated data is actually interesting enough. Additionally, in the context of training purposes, the collected data needs to be examined in order to reconstruct the course of events and provide a *suspected* ground truth.

The **second hand approach**, the third approach, is of limited use in the context of this work, as it is only able to yield non-volatile data. It involves the acquisition of several used hard disks, e.g., sold on the secondary market (possibly a major cost factor), and the analysis of the contained data (again time-consuming and likely not very valuable). Furthermore, privacy regulations must be considered here before any data can be shared with the DF community.

The **automatic generation** of data sets adds the fourth approach to this list, which was not covered by Moch and Freiling (2009). In recent years, several proposals have been made for its realization, as already mentioned before. The ongoing development of existing frameworks and the creation of new ones, which take up the shortcomings of previous projects and try to improve them, continues.

2.2. A previous attempt to prevent unwanted client-side data synthesis remnants

The approach presented by Schmidt et al. (2023) is based on the creation and curation of a template library and the subsequent matching of included templates with the graphical output of the VM, using OpenCV's Template Matching function.[2] Once the template is found in the image, a mouse click is performed on the determined coordinates. The authors developed multiple high-level functions and wrapped them into a Python library that can be integrated into existing data synthesis frameworks. The library itself, however, does not qualify to be called a data synthesis framework, since it does not provide enough infrastructure to actually drive simulations.

Although the approach of Schmidt et al. (2023) is relevant to our work, the following limitations prevent its use in our scope. First, it is a very tedious task to create and maintain the necessary template library, especially if multiple applications, operating systems and desktop environments should be supported by the data synthesis framework. Although this task could be facilitated by the community distributing the work, the management and sharing aspects still remain a large concern. Second, the used pattern matching algorithm is not resolution scale-invariant, i.e., all users would have to select the same screen resolution for their simulation. Third, as soon as any GUI updates change the appearance of control elements, the provided templates must be

[1] https://cfreds.nist.gov/ (visited on October 10, 2023).

[2] https://docs.opencv.org/3.4/d4/dc6/tutorial_py_template_matching.html (visited on October 4, 2023).

updated. The simulation of previous software versions requires to keep a history of templates in the generator library, which further increases its size.

As a result, we make use of a different approach and rely on keyboard shortcuts to control the VM, which is easier to implement and more reliable, as we show in Section 3.

2.3. *Existing data synthesis frameworks in digital forensics*

In what follows we assess existing data synthesis frameworks in digital forensics. Our goal is to come up with a decision whether to develop a new framework or build our approach based on an existing one. Our assessment is based on some criteria that a framework must fulfill in order to be useful for our agent-less approach:

- Provision of coherent data sets on different layers, including RAM, disk images, and network dumps
- Open-source, including all the software on which it depends
- Actively maintained
- Broad OS support

Table 1 lists six examined automatic and mostly open-source data synthesis frameworks which were evaluated based on the points given above. The number of actively maintained frameworks is rather limited. The majority is only intended to simulate Windows guests. The first framework, `ForGeOSI`, is able to synthesize data across all different layers. It uses VirtualBox to host the VMs and solely relies on its Guest Additions to drive the simulation. Its simulation capabilities are rather limited. `ForGeOSI` is written in Python 2 and is not actively maintained anymore which disqualifies the software. The further research confirms the absence of any other framework that manages the synthesis without additional client-side software. For instance `TraceGen`, from Du et al. (2021) does not rely on a dedicated client-side agent, but requires the execution of Python scripts inside the VM. Since its code is not open-source, it is not possible to determine its exact functionality and the statements above are solely based on the paper and the listings given within. The last framework given in the table is called `VMPOP` and is based on the work of Park (2018). While the authors discuss the software concept in detail, the implementation is only a PoC, rather limited, and at no level that an extension of the project could be considered.

`ForTrace` by Göbel et al. (2020) is an automatic data synthesis framework which is able to yield unique traces across all three relevant layers, i.e., network traffic, memory, and persistent storage. The framework is actively maintained and written in Python 3. A user can design scenarios using YAML files, simplifying access for non-Python-affine users. This enables the generation of data at a larger scale, only by modification of single entries in the configuration file of a scenario. `ForTrace` has a dedicated central agent component driving the simulation. Due to its modular structure, the central agent component can be detached from the rest of the software, hence we can compensate its absence. Thus, we decided to re-implement said framework with the agent-less functionality.

3. Conversion of `ForTrace` to an agent-less approach

In this section we present our agent-less approach on base of the selected framework `ForTrace`. The code base of `ForTrace`'s code base is large and built on its predecessor `hystck` (Göbel et al., 2020). Multiple classes are involved in the simulation process, where the client-side agent takes a central role. Its removal makes a restructuring and reworking of the remaining framework necessary. A new structure was introduced to `ForTrace` by aligning its components to the ones exported by *libvirt*'s Python API.[3]

[3] https://libvirt.org/api.html (visited on September 5, 2023).

3.1. *Comparison of the old and new architecture*

Fig. 1 shows abstract views of both architectures: the old one with a client-side agent, and the new one that is compensating for its absence and only present on the host. Although the architectures are different, the general functionality of `ForTrace` remains the same.

Fig. 1a depicts the current implementation of `ForTrace`. The **Framework Master** is responsible for the creation and administration of VMs, based on the provided configuration that applies to the whole simulation, including all VMs. Each VM, called a **Participant**, consists of several modules. The main module is the VM-associated **Generator**, which drives the simulation. Therefore, it may access **Application Modules**, which are providing interfaces to different applications on the client VM, and **Guest Functionalities**, accessible through the hypervisor for general control of the VM, e.g., its power state. On each VM runs the **`ForTrace` Agent**, thus highlighted in a more vibrant color, since this is the client-side component of `ForTrace`, we eliminate throughout this work. The agent receives its command via a *TCP* socket that is opened on each VM. In order to minimize the unintended traces of this connection, each VM has one monitored network interface to access the Internet and local network, and a second unmonitored one, directly bound to the Framework Master.

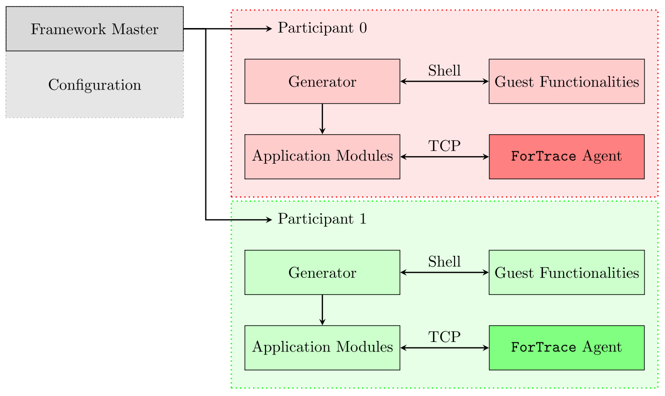
Typical problems resulting from the dependency of the agent are, for example, that the OS has to be configured so that the user logs in automatically in order to start the agent. This means that before the agent runs after the user logs in, a normal login including password input is not possible.

Fig. 1b shows the new architecture, where the central agent component is removed and all functionality is located on the host. The **Simulation Monitor** resembles large parts of `ForTrace`'s Framework Master. It is tasked with the creation of all participating domains, *libvirt*'s term for VMs, and their individually associated Generators. The Simulation Monitor holds the connection to the hypervisor, wrapped in the **VirshSession** object, the root YAML configuration, and the method that enables cross-domain communication. The latter feature is especially useful if an action on one VM should trigger another action on a different one, without tainting the generated data sets with unusual communication. Otherwise, the VMs are strictly separated from each other. Each VM has only one monitored network interface, which is connected to a *libvirt* network with Internet access, which is far more realistic than before.

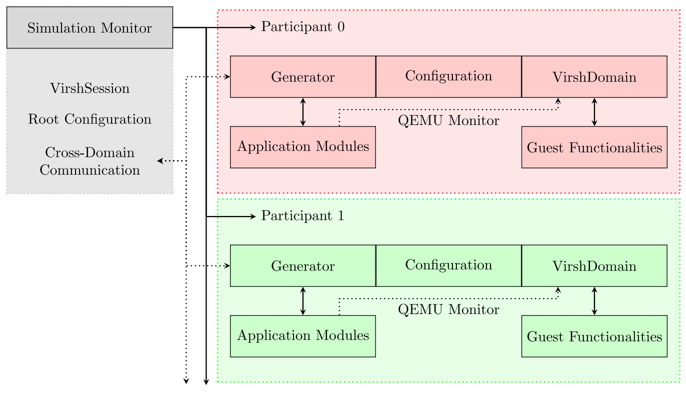
One or more Participants are tied to and controlled by the Simulation Monitor. The actual VM is wrapped inside the **VirshDomain** object of a participant. This object provides access to the Guest Functionalities, like power-management of the VM. Application Modules interact with their associated VM through the *QEMU Monitor*, a piece of software able to send complex commands – including mouse and key inputs – to QEMU. These inputs appear completely transparent to the simulated guest like a normal keyboard or mouse. The actual objects, representing the applications, are created on the host and are only intended to be used as wrappers around the QEMU Monitor. The communication between `ForTrace` and a VM via the QEMU Monitor cannot be captured with a network sniffer, since it happens entirely locally and not via TCP sockets as before. Since there is no additional software running on the guest to provide feedback to `ForTrace`, the communication to the VMs is unidirectional, meaning a command is issued to a VM, but there is no response whether it was successful. This is the main drawback of the agent-less approach, which must be mitigated in order to be able to use this approach reliably. Ideas to do so rely on the only remaining channel that provides feedback about the system state: the graphical output of a VM. It requires different steps to make it computer-readable, depending on the kind of feedback that is needed. The two approaches implemented so far are described in Section 3.2.2. Since the Generator component of `ForTrace` was completely rewritten as well and takes a central role in the new implementation, it is discussed more thoroughly in Section 3.3.

Table 1
Existing automatic data synthesis frameworks in digital forensics. The supported layers are read as follows: P = persistent, M = memory, N = network.

Framework/Publication	Layers	Virtualization software	Client-side software	OS support	Maintained	Open source
Forensig²/(Moch and Freiling, 2009, 2012)	P/–/–	KVM, QEMU	SSH-Server	Linux	×	✓
ForGeOSI/-	P/M/N	VirtualBox	VirtualBox Guest Additions	Linux/Windows	×	✓
ForGen/-	P/–/–	VirtualBox, Vagrant	Puppet, VirtualBox Guest Additions	Windows	×	✓
VMPOP/(Park, 2018)	P/–/–	VirtualBox, VMWare, KVM, QEMU	VirtualBox Guest Additions	Windows	×	✓
TraceGen/(Du et al., 2021)	P/–/N	VirtualBox	VirtualBox Guest Additions, local Python scripts	Windows	?	×
ForTrace/(Göbel et. al, 2022)	P/M/N	KVM, QEMU	ForTrace Agent	Windows	✓	✓



(a) ForTrace with a client-side agent



(b) ForTrace without a client-side agent

Fig. 1. Comparison of ForTrace architecture with and without a client-side agent.

3.2. VM interaction

ForTrace might interact with VMs in two different ways: via a console connection or through the already mentioned QEMU Monitor. Each method has its own advantages and disadvantages, which are explained in the following subsections.

3.2.1. Console interaction

By modifying the VM's configuration file, one is able to expose a guest's console to the host in the form of a pseudo TTY. This enables ForTrace to acquire the terminal and interact with it. Commands can be sent to the terminal and the output is transferred back into ForTrace, so it can be processed. For example, one can send the command `echo Hello, World!` to the VM and expect `Hello, World!` as an output. This is a huge advantage, as the communication without a client-side agent is otherwise limited and more unidirectional. With a working terminal session, this restriction is lifted for non-graphical interaction, e. g., servers like the one used in our proof of concept in Section 4.1.

After the connection is established, the module will run some initial setup commands to prepare the shell for ForTrace. Commands – elevated ones as well – can be entered, and their output is returned. As the communication happens through a *libvirt* console channel and is presented to the guest as a character device – like a regular keyboard, mouse, or serial port – it does not travel over the network, thus keeping the network dump clean and making the second control network superfluous, removing this additional trace of ForTrace from the simulation. Multiple channels can be added to a VM's configuration, which allows ForTrace to use more than one console simultaneously.

Several limitations related to the communication via *libvirt* console were discovered during the implementation and subsequent evaluation of the simulation results. These are discussed in the following.

First, in order to provide a connection for the first console channel the guest has to activate a systemd unit called serial-getty@hvc0.service. This has to be done for each console device of the VM, incrementing the unit's number. Logs related to the opening and closing of a terminal session are generated and may be found during the evaluation.

Second, when performing a live system analysis, one is able to list all active user sessions of a system. This can, for example, be achieved with Volatility's *linux_check_tty* plugin. As with the systemd service, the console type is clearly visible in the *TTY* column. Normally, the value is something like *pts/** or *tty** (* denoting a number). The term "hvc" – *hypervisor console* – can clearly be put into the context of virtualization, as such a console would not be present on a non-virtualized system. It is possible to add a console as a serial device to the hardware configuration of a VM. Then, the value in the *TTY* column is something like "ttyS0". This may be less noticeable, however, the console applications, with their current implementation, discontinue to work properly.

Third, the prompt of a console is usually not static and often displays the current working directory. With each directory change the prompt would have to be updated. If the user misspells a path, the prompt is not changed and a timeout error is thrown. To circumvent such behavior and keep the code much simpler, the prompt is changed at the beginning of each login, so it is static throughout the session. As the change is only temporary and limited to a single user, no additional privileges are required, thus the resulting traces are minimal. The change command, however, shows up in the shell history. In a post-processing step, the prompt-change command would need to be purged from the shell's history file. However, it is better to avoid the generation of traces completely, instead of removing them afterwards, since this process is error-prone and fundamentally opposite to the whole concept of this paper.

3.2.2. Graphical interaction

To enable GUI interaction with a VM, the new base class is extended with a *desktop environment* attribute. The user needs to provide the expected desktop environment in the configuration file, so that ForTrace is able to construct the correct object for interaction. This attribute represents the whole graphical part of a VM. It keeps track of all active applications and the one that is currently in focus, i.e., receiving mouse and keyboard input. All applications can send and receive data through the desktop environment object. Thus, one application is able to notify another application about a changed state (e.g., application **A** informs the desktop environment about its termination), so the latter updates the focus and the list of opened applications. With this architecture it is possible to express application dependencies, i.e., parent and child windows, whereby, for example, closing the parent window also closes all child windows. However, if the focus is lost during the simulation, the results might be unpredictable. Thus, excessive testing is necessary before integrating any new application or functionality into the framework. To mitigate the problem, it is advisable to maintain the newly developed unit test suite so that each application can be tested individually. For instance, uncaught popup windows might lead to a deadlock situation during the simulation process, since they can *swallow* later inputs.

`ForTrace`'s original approach interacting with open applications via their own specific objects was rewritten in this work. *GenericApplication* is the abstract base class for all application objects and defines the baseline of necessary attributes and methods a graphical application has to offer, e.g., closing the application window. Each application has a name and a UUID, which can be used to identify an application, even if there are multiple instances of it opened at the same time.

Along the new inheritance hierarchy, the application classes get more specialized, extending the instantiated objects with more application-specific features. For example, a text editor has to be able to save a document but is not able to browse to a given Internet address. Common features that both applications share, however, are that both can be closed and can process text input from the user. Users can extend the framework by adding new children to the application hierarchy and implementing the necessary interfaces.

The main problem of graphical interaction without an agent is the unidirectional communication channel, since the graphical part of a VM can only be observed by taking pictures and drawing conclusions about the VM's state. When using a client-side agent, it most likely uses APIs to the applications under its control, which provide feedback about received inputs. So there are multiple questions `ForTrace` has to deal with: Is an input truly processed? What is the current state of the VM's user interface? Are there any popups, warnings, etc., that have to be closed? All these questions should be answerable by the framework, so a simulation is not completely blind between start and end and might react to certain events. Currently, the framework supports two low-level solutions to remedy the situation. Both are based on the graphical channel of a VM since it can be accessed without a client-side agent.

First, the *extract_text* method is intended to be used to start an Optical Character Recognition (OCR) on the application window. Currently, this method extracts the text from the whole screen since it cannot determine the borders of individual application windows. There are multiple functions to test, whether a (similar) string is contained in the OCR result, e.g., the function shown in Listing 1 that matches a list of substrings (second parameter) with a string similarity algorithm against the extracted text. This method is more reliable than simple pattern matching like in the approach of Schmidt et al. (2023), since the content of, e.g., popup windows and cookie banners can be predicted in a more general fashion. For instance almost every cookie banner, how different they may look in general, contains certain keywords like "Accept" or "Decline". This enables `ForTrace` to react to the actual information displayed on the screen rather than to the patterns it might contain. Also this approach does not require the provision of a curated template library and can be integrated into the code more easily.

```
if text_line_contains(self.extract_text(), ['Accept',
↪  'Decline']):
    self._qs.send_key_combination("ret")
```

Listing 1. Performing OCR to catch a cookie popup window.

The second approach lies in the computation of similarity between images, which is similar to the approach of Schmidt et al. (2023). In order to be able to compare screenshots of a VM and reference images against each other, it was decided to compute the similarity between two images using the Normalized Root Mean-Square Error (NRMSE).[4] For example, this approach is used to determine whether a VM is booted and ready to receive user input, by taking pictures in fixed time intervals and comparing them against each other, thus rendering a template library superfluously. It is assumed that once the boot process has been completed, the boot screen is rather static, which results in a low NRMSE. Of course, other and more complex methods for image similarity can be integrated into the framework as well. However, as discussed in Section 2.2, this approach has several drawbacks, hence it is used only in the described case so far.

3.3. *Redesign of the* `ForTrace` *Generator*

The Generator component was rewritten to meet all new requirements. The exact purpose of this class was redefined, and thus its interface scaled down drastically. However, the general behavior is still the same: The first operation consists of loading all defined and default collections (patterns for actions) into the Generator. The next step, the setup of all needed applications, is omitted in the new Generator implementation and handled on the fly. It makes use of the new application structures explained in Sections 3.2.1 and 3.2.2, which are more flexible regarding the provisioning of applications. Thus, applications can now be opened and closed multiple times during a simulation, which enhances the realism of the generated data. The penultimate step consists of the generation of individual actions, based on the provided collections and applications. Finally, the list of generated actions is shuffled, which concludes the Generator's initialization.

The new implementation of the Generator is strictly limited to the loading of collections, generation of actions, and their execution in random order. One instance of a Generator serves a single VM. The interaction possibilities a Generator has on its list of actions are visualized in Fig. 2. The first phase – the generation – creates all to-be-simulated actions after each other, hence the list is sorted at the beginning. In the next step, all actions of the list are shuffled, after the seed provided within the configuration. Afterwards, the execution of actions (represented by the arrow above an action) might be started by calling the Generator's *execute* method, which starts the iteration over the list.

With the new implementation of `ForTrace`, more complex actions are wrapped into their own objects and can be grouped into attack- or server-interactions. Further interaction categories might be added in the future if the need arises. The Generator must not implement complex interactions with a VM in its own logic but is urged to call methods provided by these action objects, thus keeping the actual execution logic separated. Throughout the execution of actions, it falls to the Generator to respond to certain, predefined events. As an example, the action to retrieve a file from a simulated server can only be completed if there are files present on the server. Thus, an action for retrieval might send back an error event and asks the Generator to be executed again, but at a later time. The Generator has to react to this event and must reorder the event in the list of actions. Fig. 2 shows the reordering of a server interaction in the fourth line.

Attack-interactions pose a special kind of action element, since there exact number of calls is unknown. They were implemented this way, so the user does not have to know the number of calls necessary to conclude the attack. Rather this responsibility falls to the Generator, which dynamically reacts on the status of an attack-object and adds or removes calls to the list of actions, as shown in the last two lines of Fig. 2.

All actions specific to a VM's life-cycle are extracted from the

[4] https://github.com/scikit-image/scikit-image/blob/v0.21.0/skimage/metrics/simple_metrics.py/#L50-L108 (visited on September 22, 2023).

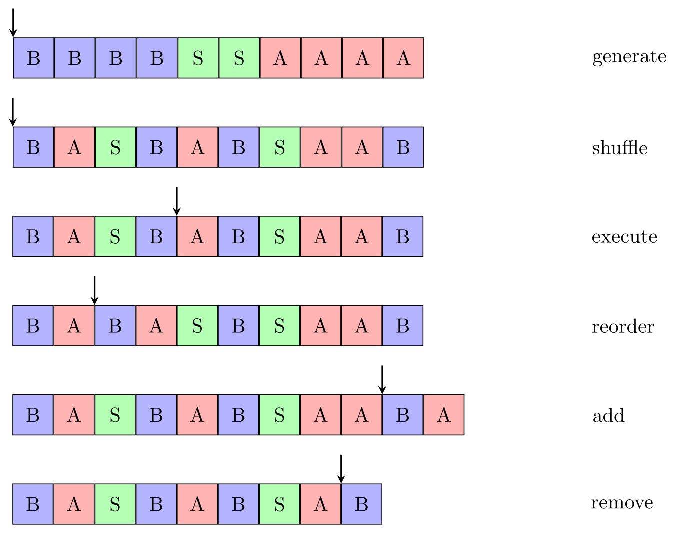


Fig. 2. Visualization of modification steps undertaken by `ForTrace`'s Generator on the list of actions. B=Browser action, A = Attack action, S=Server action.

Generator and now fall into the domain of the Simulation Monitor. Among them are: the initial boot and final shutdown of a VM, the starting and stopping of the network sniffer, and the generation of the (final) RAM and image dumps. All the described changes contribute to the Generator's encapsulation and independence of the simulated VM.

4. Linux attack scenario as a proof of concept

As a PoC, in this section we present a sample data synthesis scenario involving an attack on a file-sharing service using SFTP, starting from an attacker VM to the victim VM. As mentioned before, the agent-less approach works OS-independent, and necessary interfaces are already in place where needed, which will make the re-implementation of `ForTrace` truly capable of simulating Windows and macOS guests, besides other Linux distributions (and their desktop environments).

Since the number of frameworks capable of simulating Linux scenarios, especially using different distributions, is limited and the flexibility of the agent-less approach should be demonstrated, a scenario in this domain was developed.

Not only the SFTP server is completely simulated by `ForTrace`, but also the attacking machine. Two very general interfaces were developed to drive multi-staged attacks and generic server interactions. The latter is not limited to SFTP servers, but also capable of running other webservers as well, including non-data-sharing services, such as mail servers. Both interfaces are integrated into `ForTrace`'s Generator, described in Section 3.3, and provide callable action elements. This is a new feature introduced to `ForTrace` and capable to integrate more diverse and complex actions in scenarios.

Fig. 3 shows the general procedure undertaken by `ForTrace` when executing a scenario. Depending on the type of provided YAML configuration, the setup process merges cascading ones. A Generator is created for each VM, as described in Section 3.3. Once the Generator setup is completed, the VMs can be booted. If further setup steps are required, e. g., the installation of software, they are performed right before the scenario is started. The user can specify to perform RAM dumps at any time, and create image dumps, once the scenario is completed.

4.1. The victim – an Ubuntu SFTP server

The server runs on Ubuntu 22.04.2 LTS without any installed desktop environment, leaving the terminal the only option to receive user input. The server uses *ProFTPD* to provide a SFTP server inside the local

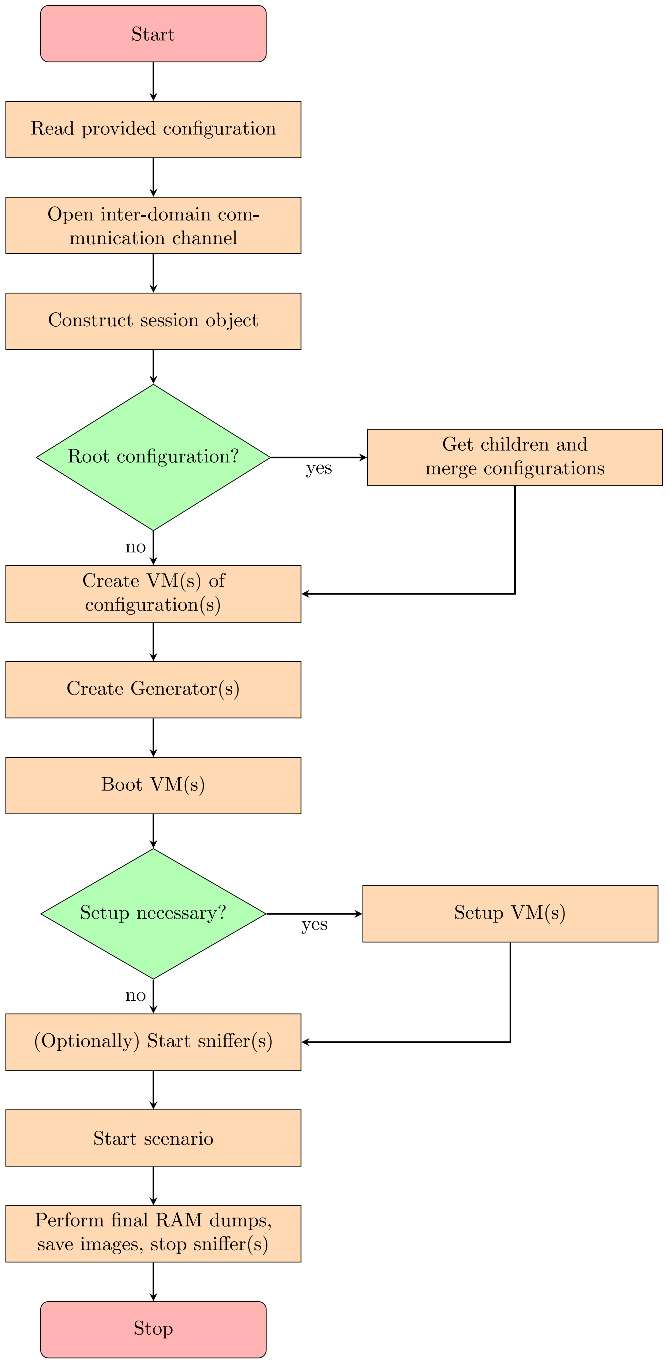

Start
Read provided configuration
Open inter-domain communication channel
Construct session object
Root configuration?
yes
Get children and merge configurations
no
Create VM(s) of configuration(s)
Create Generator(s)
Boot VM(s)
Setup necessary?
yes
Setup VM(s)
no
(Optionally) Start sniffer(s)
Start scenario
Perform final RAM dumps, save images, stop sniffer(s)
Stop

Fig. 3. Procedure for creating scenario-based image files.

network.

The attack is based on CVE-2015-3306,[5] which was published on 18th May 2015 and affects all *ProFTPD* versions up to and including 1.3.6.

To enrich the scenario, multiple users are registered on the server. Their actual number, names, passwords, and group memberships are modifiable. This is part of the default Linux server setup procedure, which takes place automatically before the simulation starts and is also a new component brought into `ForTrace`. The installation of the required software is handled automatically during the setup phase as

[5] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3306 (visited on June 21, 2023).

well.

Users interact with the server by placing private files within their respective user directories. Either a pool of files can be specified and/or `ForTrace` is instructed to generate files on demand. The latter involves the Faker library[6] to generate filenames and content (Nandakumar, 2022).

4.2. The attacker – a Kali Linux machine

The attacker uses Kali 2023.2 with GNOME as desktop environment in this scenario. For simplicity, the *Metasploit* framework is utilized for the majority of the attack. Also, `ForTrace`'s pseudo-TTY module (see Section 3.2.1) was extended to enhance the interaction with the *Metasploit* Shell, e.g., provide better output, open a *Meterpreter* session, etc. The *Metasploit*-related classes are completely OS-independent and only require a TTY-session.

A more detailed description of the whole attack can be found in Appendix A. Certain aspects of the attack can be changed through the configuration without the need to write Python code. To be able to apply it to other attacks as well, the Cyber Kill Chain, defined by Lockheed Martin (Hutchins et al., 2010), was used to subdivide the configuration so that the meaning of each option is better understood from the context. The configuration as such can influence individual aspects, but cannot change the nature of the attack itself. Fundamentally different attacks have to be implemented as Python code, even if they are welcome to make use of the configuration setup or reuse parts of this attack's implementation.

During the scenario, the attacker browses on a pool of hacking-related websites among others on a random basis. To make the scenario more realistic, a website can be parsed with the Python library *BeautifulSoup*[7] for HTML elements, to extract any associated links. It can be chosen from which elements links should be returned, e.g., all images on the website, all hyperlinks, and so on. The scraping process is performed on the host, thus independent of the client's used browser and without leaving uncommon traces on the simulated VM. Some files are then downloaded through the extracted links to enrich the scenario. `ForTrace`'s Generator adds the accessed links directly to the respective action to preserve the ground truth.

5. Evaluation of generated data

This section analyses the results of the scenario developed in Section 4. With minimal changes to the configuration, it is possible to obtain a RAM dump, a network dump, and an image dump of every involved VM. The scenario however dictates that all three layers of data are only present for the Ubuntu server. On the attacker side, there is only the image available. The generated data set, the used configuration, and generated logs are available via our Zenodo upload.[8]

5.1. Comparison of artifacts left by previous `ForTrace` version

The most recent `ForTrace` implementation regarding the support for Linux with a client-side agent was implemented and discussed by a student writing his bachelor's thesis. The author found some traces during the evaluation of the other Linux scenario performed in his work, which are listed in Table 2 with the information whether they could be omitted with the agent-less approach. The traces found are discussed below and compared to see if they are also present in the current implementation.

In order to start the agent an autostart entry needs to be created during the installation of a guest. This file, located in the main user's home directory, remains on the system and is therefore included in the generated disk image. At startup, for which the command can be extracted from the memory dump as shown in Listing 2, the agent establishes a connection to the constant IP address `192.168.102.1` on port `11000`, which is routed through the private network interface, which every VM must have. Although the connection might be hidden from the network sniffer, an analysis of the RAM dump reveals the opened socket. As there is neither a client-side agent nor an IP-based connection to the hypervisor, no such artifacts are generated with the new approach. Furthermore, a VM has no second network interface for private communication with `ForTrace`, which means that this trace is also eliminated.

```
starting agent.desktop: command=gnome-terminal -e 'bash -c
↪  "python3
↪  /home/fortrace/hystck/guest_tools/guestAgent.py; bash"'
↪  startup-id=[...]
```

Listing 2. Autostart command for the `ForTrace` agent extracted from the memory dump.

One larger part in the class for console interaction, discussed in Section 3.2.1, deals with the password prompt that is shown after a command is prefixed with *sudo*. Since the previous implementation of `ForTrace` started commands on the guest via the *subprocess* module, it had no method to interact with the detached `stdin` afterwards. Hence, the `/etc/sudoers` file was modified to disable the password prompt for the "fortrace" user completely. The modification is not reverted at the end of a simulation, thus visible in the resulting image. The initial modification of the file is documented in the system logs as well. Since this file is one of the first sources in an investigation, the modification drastically decreases the realism of the resulting data set.

Through the `qemu-install` command, issued in the automatic installation scripts, several groups are created during the VM setup and remain in the `/etc/group` file, e.g., `libvirt`, `libvirt-qemu`, `libvirt-dnsmasq`. The automated installation scripts cannot be used with the new implementation of `ForTrace`, since they have not been adjusted so far. The *libvirt* configuration needs to be changed in several sections, e.g., the video settings need to be set to "vga" (to detect the mouse), consoles to be used by `ForTrace` need to be added. If they are rewritten for convenience reasons the mentioned groups will most certainly be part of the disk image again. The commands issued during the installation, e.g., to install required Python packages, are included in the history file too. With the adapted *guestfs* component, which can already be used to transfer files to the powered-off VM, the groups and Bash history file could be modified, without leaving traces in the system logs. Anyhow, since the scripts are currently not used, neither the group file nor the Bash history is tainted by any *libvirt* or QEMU artifacts, and the installation process of new VMs is rather simple and well documented. As discussed in Section 3.2.1, the Bash history contains the command to perform a prompt change, once the user is logged in. Far less traces, compared to the situation before. The implementation of the console interaction module could be changed to handle changing prompts and keeping the history clean.

When executing Python code or importing a Python module for the first time, a directory named `__pycache__` is created in the module

Table 2
Overview of traces found by a former `ForTrace` evaluation and statement if they are omitted through the agent-less approach.

Traces	Omitted
Installation Logs	✓
Shell History	×
Password-less sudoer	✓
ForTrace Python Library	✓
Agent-Process & autostart entry	✓

[6] https://github.com/joke2k/faker (visited on May 5, 2023).
[7] https://www.crummy.com/software/BeautifulSoup/ (visited on June 21, 2023).
[8] https://zenodo.org/records/8359194.

directory, containing compiled Python source code, also called "byte code". This caching process is done for performance reasons, as this makes the loading of Python modules much faster because the compilation phase can be bypassed (Warsaw, 2009). Since ForTrace is implemented in Python, the agent is also implemented in Python and is therefore cached in the aforementioned directory. This results in multiple accesses and creations of pre-compiled byte files that are visible in an investigation of the disk image. The new implementation brings several advantages in this regard. For one, there can be no ForTrace-related byte files on the system because there was never ForTrace-related software installed. On the host side, all Python files are cached as usual, thus providing the familiar performance. Second, the Python installation on VMs, if there is any, remains devoid of any ForTrace packages and requirements. By omitting the agent, there is no need for any special software or even a Python installation on the simulated client, and the system can be configured in the most open way.

The last artifact found by the student is located in the RAM dump of a scenario. Log messages from the client-side agent process (and thus very likely the whole process) can be found. When Firefox is actively used and controlled by ForTrace the Gecko driver[9] leaves log messages in the RAM as well. Other APIs for application control act similarly, thus often tainting the data set further and decreasing its realism. In this case, too, it behaves similarly to the previously discussed artifacts: Since there is no foreign software on the VM, it cannot leave any strange logs or traces.

5.2. *Means to minimize traces left by a client-side agent*

Of course, the authors of ForTrace knew about the traces and tried to mitigate them, hence providing two methods, which should be added to the end of each scenario. Both methods are implemented for Windows only, at the time of writing. Thus, in this comparison, their capabilities in a Linux environment have to be extrapolated from the real implementation. These methods were necessary, since the traces could not be omitted with the agent component in place. However, it is clearly better to avoid traces than to remove them afterwards.

The first method *_initClean* removes artifacts within the Event Log and the Prefetch directory, generated during the configuration of the template. As for the latter, there is no real equivalent to Windows' Prefetch directory on a Linux system. The equivalent of Windows' Event Log, however, in Linux is the journal. The journal is sealed (Marson et al., 2013), so single entries cannot be removed trivially. However, what is in the journal, that reveals the ForTrace simulation? No additional software was installed on the VM so that no traces of e.g., a running agent can be generated. Still, as mentioned in Section 3.2.1, the interaction through the *libvirt* console is visible in the logs. The messages in the journal file cannot be removed or modified. This would result in a corruption of the file, leading to a failing integrity check.

The second method *_cleanUp* wipes artifacts created through ForTrace during the execution of the scenario. Again, this method is not required anymore when having no agent present on the system, as there are no Python or autostart files or ForTrace installation folders in any user directory.

Through modification of the VM configuration file, which is used by *libvirt*, KVM can be advised to hide itself; a feature available since version 1.2.8. It has to be manually added to the VM's configuration, along with the hypervisor feature policy change in the "vcpu" section (Hampton, 2018). This suppresses boot log entries regarding the virtualization of the system. Disadvantages caused by setting this option could not be determined. In any case, this option is not specific to the agent-less approach, and shall therefore only be mentioned here.

To summarize the insights gained in this section: many unintended traces can be avoided by omitting the client-side agent. The only drawback is the "hvc" string referring to the console connection and the more sophisticated host-side logic required to mitigate the drawbacks of the absence of the agent.

6. Conclusion and future work

With the removal of the agent component, many parts of ForTrace had to be rewritten. Various interactions – both graphical and non-graphical – can be simulated on Linux guests, completely transparent to them. The utilized QEMU Monitor is independent of the underlying OS, and hence works on Windows guests as well. Therefore, agent-less support for Windows is easy to implement as the integration points for Windows and macOS are already provided with our new implementation of the ForTrace framework.

Ideas to prevent certain traces created during the simulation were presented in different sections of this work. These findings are fundamentally valuable and can in any case be used for future simulations. Whether there are more unintended traces on VMs (especially on those running Windows) needs to be determined thoroughly in future research once the implementation supports other operating systems. Most of the traces, previously found during a former evaluation of ForTrace, were omitted or drastically mitigated through the removal of the agent. Of course, this decision is followed by other problems. The limited feedback of the graphical channel resembles the central one and is only partially solved through the image similarity and text recognition. In order to provide a useable and robust framework, window elements and popups need to be recognized reliably. This problem can potentially be solved better by employing machine learning or integrating automated UI testing solutions like AskUI[10] than, for example, by the OpenCV approach of Schmidt et al. (2023), because the latter would probably require providing all targeted elements (for every OS/distribution/GUI) in the form of images before the simulation can be started. This would lead to an enormous (manual) workload and breaks every time a GUI is updated. Of course the first approach, however implemented, is more complex than simple pattern matching, but will pay off in the long run.

This work showed the strengths and first of all the general usability of this novel client-side agent-less approach driving a complete Linux scenario that will be further developed in future works to extend and validate its full potential.

Appendix A. Stages of the executed proof of concept attack

Reconnaissance The attacker uses `masscan` to find all network participants in the subnet. This yields the Ubuntu Server. Next, `nmap` is run to discover the FTP service running on this server, revealing *ProFTPD* as the driving software. Then the attacker searches for *ProFTPD*-related modules and finds one suited for version 1.3.5.

Weaponization As already mentioned in Section 4.1 the Ubuntu Server runs *ProFTPD* version 1.3.5 which is vulnerable to CVE-2015-3306. *Metasploit* provides with *exploit/unix/ftp/proftpd_modcopy_exec* the associated module[11] to perform the exploit.

Delivery The module is configured to deliver a TCP reverse shell as a PHP file to the public writable web directory `/var/www/html`.

Exploitation after successful delivery of the payload, the delivered reverse shell is triggered, which then reaches out to the attacker's machine and is registered as a new session in the *Metasploit* console.

Installation The adversary investigates the crontab file (`/etc/crontab`) and locates a script that is used to make backups via `rsync`. To make sure all files can be accessed by this script it is run with root permissions. However, the permissions on the script itself are misconfigured and grant every user write access, so arbitrary commands

[9] https://github.com/mozilla/geckodriver (visited on June 13, 2023).

[10] https://www.askui.com/ (visited on December 3, 2023).

[11] https://www.rapid7.com/db/modules/exploit/unix/ftp/proftpd_modcopy_exec/ (visited on April 12, 2023).

might be executed.

By modifying the backup script, the attacker changes `/etc/sudoers` so the *www-data* user is allowed to use `sudo` without authentication.

After the cron job has run, the adversary can create a systemd unit with root privileges to open a reverse shell, so a connection can be established anytime in the future. Users are free to write their own unit file or script and include it in the attack. In this scenario the malicious service hides as a user management-related service. Once the file is uploaded to the system, it is moved into the `/etc/systemd/system` directory, and then enabled and started, so an opened multi-handler inside the *Metasploit* Console can receive the connection.

Command and Control The interaction is completely manual and realized from within the *Metasploit* Console. For convenience, the opened root shell is upgraded to a *Meterpreter* session again.

Actions and Objectives The adversary starts the collection of relevant files and extracts them via the download command of the newly opened *Meterpreter* session. These files can be specified via the configuration file, in the eponymous section. They are grouped in batches, so each batch can be executed by the Generator on a random basis.

Optionally cover-up steps – all executed in a root shell on the target – can be added via the configuration, as long as they follow the obligatory convention. Eventually, with the shutdown of the *Metasploit* framework the automatic cleanup (of *Metasploit*) is started, which removes exploit-related files from the target.

References

Carrier, B., 2002. Open Source Digital Forensics Tools.

Casey, E., 2004. Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet, second ed. ed. Academic Press, London ; San Diego, Calif. OCLC. ocm53356563.

Du, X., Hargreaves, C., Sheppard, J., Scanlon, M., 2021. TraceGen: user activity emulation for digital forensic test image generation. Forensic Sci. Int.: Digit. Invest. 38, 301133 https://doi.org/10.1016/j.fsidi.2021.301133. https://linkinghub.elsevier.com/retrieve/pii/S2666281721000317.

Garfinkel, S.L., 2010. Digital forensics research: the next 10 years. Digit. Invest. 7, S64–S73. https://doi.org/10.1016/j.diin.2010.05.009. https://www.sciencedirect.com/science/article/pii/S1742287610000368.

Garfinkel, S., 2012. Lessons learned writing digital forensics tools and managing a 30TB digital evidence corpus. Digit. Invest. 9, S80–S89. https://doi.org/10.1016/j.diin.2012.05.002. URL: https://www.sciencedirect.com/science/article/pii/S1742287612000278.

Garfinkel, S., Farrell, P., Roussev, V., Dinolt, G., 2009. Bringing science to digital forensics with standardized forensic corpora. Digit. Invest. 6, S2–S11. https://doi.org/10.1016/j.diin.2009.06.016. https://linkinghub.elsevier.com/retrieve/pii/S1742287609000346.

Göbel, T., Schäfer, T., Hachenberger, J., Türr, J., Baier, H., 2020. A novel approach for generating synthetic datasets for digital forensics. In: Peterson, G., Shenoi, S. (Eds.), Advances in Digital Forensics XVI. Springer International Publishing, Cham, pp. 73–93. https://doi.org/10.1007/978-3-030-56223-6_5.

Göbel, T., Maltan, S., Türr, J., Baier, H., Mann, F., 2022. ForTrace - a holistic forensic data set synthesis framework. Forensic Sci. Int.: Digit. Invest. 40, 301344 https://doi.org/10.1016/j.fsidi.2022.301344.

Grajeda, C., Breitinger, F., Baggili, I., 2017. Availability of datasets for digital forensics – and what is missing. Digit. Invest. 22, S94–S105. https://doi.org/10.1016/j.diin.2017.06.004. https://linkinghub.elsevier.com/retrieve/pii/S1742287617301913.

Hampton, M., 2018. Answer to "Hiding Virtual machine status from guest operating system". https://superuser.com/a/1389159.

Hutchins, E.M., Cloppert, M.J., Amin, R.M., 2010. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains.

Jarrett, A., Choo, K.K.R., 2021. The impact of automation and artificial intelligence on digital forensics. WIREs Forensic Science 3, e1418. https://onlinelibrary.wiley.com/doi/abs/10.1002/wfs2.1418, 10.1002/wfs2.1418. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wfs2.1418.

Lin, Y.D., Lee, C.Y., Wu, Y.S., Ho, P.H., Wang, F.Y., Tsai, Y.L., 2014. Active versus passive malware collection. Computer 47, 59–65. https://doi.org/10.1109/MC.2013.226 conference Name: Computer.

Marson, G.A., Poettering, B., 2013. Practical secure logging: seekable sequential key generators. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Crampton, J., Jajodia, S., Mayes, K. (Eds.), Computer Security – ESORICS 2013, vol. 8134. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 111–128. https://doi.org/10.1007/978-3-642-40203-6_7. http://link.springer.com/10.1007/978-3-642-40203-6_7. series Title: Lecture Notes in Computer Science.

Mitchell, F., 2014. The Use of Artificial Intelligence in Digital Forensics: an Introduction. Digital Evidence and Electronic Signature Law Review 7. https://doi.org/10.14296/deeslr.v7i0.1922. http://journals.sas.ac.uk/deeslr/article/view/1922.

Moch, C., Freiling, F.C., 2009. The forensic image generator generator (Forensig2). In: 2009 Fifth International Conference on IT Security Incident Management and IT Forensics, pp. 78–93. https://doi.org/10.1109/IMF.2009.8.

Moch, C., Freiling, F.C., 2012. Evaluating the forensic image generator generator. In: Gladyshev, P., Rogers, M.K. (Eds.), Digital Forensics and Cyber Crime. Springer, Berlin, Heidelberg, pp. 238–252. https://doi.org/10.1007/978-3-642-35515-8_20.

Mohsin, K., 2021. Artificial intelligence in forensic science. SSRN Electron. J. https://doi.org/10.2139/ssrn.3910244.

Nandakumar, S., 2022. Faker library in python - an intriguing expedient for data scientists. https://towardsdatascience.com/faker-library-in-python-an-intriguing-expedient-for-data-scientists-7dd06f953050.

Park, J., 2018. TREDE and VMPOP: cultivating multi-purpose datasets for digital forensics – a Windows registry corpus as an example. Digit. Invest. 26, 3–18. https://doi.org/10.1016/j.diin.2018.04.025. https://linkinghub.elsevier.com/retrieve/pii/S1742287617303614.

Schmidt, L., Kortmann, S., Hupperich, T., 2023. Improving trace synthesis by utilizing computer vision for user action emulation. Forensic Sci. Int.: Digit. Invest. 45, 301557 https://doi.org/10.1016/j.fsidi.2023.301557. https://linkinghub.elsevier.com/retrieve/pii/S2666281723000665.

Warsaw, Barry, 2009. Pep 3147 – PYC repository directories | peps. python.org. https://peps.python.org/pep-3147/.