Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS 2022 EU - Selected Papers of the Ninth Annual DFRWS Europe Conference

ForTrace - A holistic forensic data set synthesis framework

Thomas Göbel [a, *], Stephan Maltan [a], Jan Türr [b], Harald Baier [a], Florian Mann [c]

[a] *Research Institute CODE, Universität der Bundeswehr München, Germany*
[b] *Department of Computer Science, Technical University Darmstadt, Germany*
[c] *IT Security Infrastructures Lab, Friedrich-Alexander University Erlangen-Nürnberg, Germany*

A R T I C L E I N F O

Article history:

Keywords:
Forensic image generation
Data synthesis
User simulation
Forensic data set
Digital forensic corpora
Forensic education
Forensic tool testing

A B S T R A C T

Digital forensic experts are confronted with a wide variety of investigation objectives, e.g., to deal with an infected IT system. The same holds for digital forensic tools. Mostly different sources of digital traces have to be inspected including persistent storage devices (e.g., SSDs, SD cards, USB drives), volatile main memory snapshots, and network captures, respectively. In order to train experts and tools and keep their knowledge and capabilities up-to-date, a capacious amount of realistic, timely training data is necessary. However, due to different reasons like privacy, secrecy, or intellectual property rights there is a large gap in digital forensic training data. In recent years different synthesis frameworks to generate realistic digital forensic data sets have been proposed. However, none of these frameworks provides a *holistic* approach to generate realistic digital forensic relevant traces of different sources. In this paper we introduce `ForTrace`, a holistic framework for the simultaneous generation of persistent, volatile and network traces. Our approach is based on the data synthesis framework `hystck`. We explain our extension of `hystck` by defining properties of a holistic data set synthesis framework and by discussing different forensically relevant scenarios and their implementation in `ForTrace`. We then successfully evaluate `ForTrace` with respect to diverse realistic and complex scenarios. `ForTrace` is open source and may be adapted or extended with respect to individual needs.

1. Introduction

The advancing digitisation leads to numerous challenges and threats for the security of companies and private individuals. To overcome these challenges, well-trained forensic experts are needed who are able to reconstruct the exact sequence of actions on modern devices and operating systems after a crime and find the traces left behind by the criminals. In order to keep up with the complexity and variety of today's forensic investigations, forensic experts typically need a large amount of training data covering a broad spectrum of criminal activities to train their skills and validate their digital forensic tools. In general, the more forensically relevant scenarios (like a data breach, a ransomware, or a phishing scenario) are covered by the training data sets, the better. The need for digital forensic images and the importance of such data corpora for digital forensic education, training, research, tool and process development and validation has been pointed out several times in the past (Garfinkel, 2007; Garfinkel et al., 2009; Woods et al., 2011; Yannikos et al., 2014; Grajeda et al., 2017). In recent years, several repositories for digital forensic data sets have been published, including disk images, mobile device images, memory dumps, and network packet captures. The most prominent platforms among them are (1) *Digital Corpora*[1] (Garfinkel et al., 2009), (2) the *Computer Forensic Reference Data Sets (CFReDS)* project[2] at NIST (NIST, 2021), and (3) the *Datasets For Cyber Forensics* platform[3] (Grajeda et al., 2017). Although many of the data sets found on these platforms are treated as standardised digital forensic corpora, they often have drawbacks, such as insufficient diversity, poor timeliness, unrealistic or missing background noise (i.e. no regular usage patterns like email or browser content), and most importantly, unknown ground truth.

Therefore, in addition to the mere existence, i.e., the quantity of such data sets, the quality of the data sets undoubtedly also plays an important role. For example, it is important to have an appropriate

* Corresponding author.
E-mail addresses: thomas.goebel@unibw.de (T. Göbel), stephan.maltan@unibw.de (S. Maltan), jan.tuerr@stud.tu-darmstadt.de (J. Türr), harald.baier@unibw.de (H. Baier), florian.mann@fau.de (F. Mann).

[1] https://digitalcorpora.org.
[2] https://cfreds.nist.gov.
[3] https://datasets.fbreitinger.de.

https://doi.org/10.1016/j.fsidi.2022.301344

number of traces at different layers available within an image. This is because sophisticated forensic analysis is no longer typically performed with a single data source, but requires clever correlation of different digital traces based on a recurring attack pattern (Amato et al., 2017). In such a multi-source forensic investigation not only the digital traces on the persistent storage are analysed, but also further traces, like those available in the volatile main memory or in the network traffic. We believe that it is essential to have correlated data from multiple data sources available in modern training materials to provide the most realistic experience for trainees. Therefore, there is a need for a framework that simultaneously generates holistic data, i.e., correlated persistent, volatile and network traces from the same device, including a realistic and sufficient amount of background noise for a given scenario.

Another aspect that highlights the importance of digital forensic data sets, and as such especially up-to-date data sets, is the increasing use of machine learning techniques in digital forensic analysis. Recent work in the field of digital forensics is increasingly using machine learning and in particular also deep learning, e.g., for malware classification (Le et al., 2018), for automatic Android malware detection and family attribution (Karbab et al., 2018), for the detection of online sexual predatory chats (Ngejane et al., 2021), and last but not least for child pornography classification (Dalins et al., 2018). Qadir and Noor (2021), however, conclude that the biggest challenge in machine learning-based digital forensics actually is the optimal selection and use of data sets. They mention that the crime scene is constantly evolving, so it can be difficult for the machine learning algorithm to classify the new developments accurately. In fact, most of the popular machine learning algorithms are based on supervised learning, which means that these algorithms require a large amount of ground truth training data anyway for proper classification.

To keep machine learning models up-to-date and train forensic practitioners on newer data sets with actual relevant and recent digital traces (e.g., including recent malware or ransomware samples) as well as a known ground truth, the forensic community clearly needs a framework to create such data sets as quickly, flexibly, and easily as possible. However, due to different reasons like privacy, secrecy, or intellectual property rights, there actually is a large gap in publicly available data sets in the field of digital forensics. Grajeda et al. (2017) published an article on the availability of data sets in digital forensics. The authors analysed 715 conference and journal papers from 2010 to 2015 in terms of the utilisation of data sets and if they were published. They found that (1) many researchers create their data sets manually, (2) data sets are mostly not shared, and (3) there is a lack of standardised, labelled data sets available. Abt and Baier (2014) previously published a similar paper in which they reviewed 106 network security papers from 2009 to 2013 and reached nearly the same conclusions.

The lack of such standardised, labelled data sets indeed leads to poor quality control of forensic tools and processes due to missing validation capabilities, poor metrics in machine learning (e.g., weak classification skills of a machine learning algorithm due to bad or insufficient training and test data), and last but not least, poor opportunities in teaching digital forensic students and analysts. Moreover, with a lack in publicly available data sets, the digital forensic community often faces issues like low reproducibility, comparability and peer validated research (Abt and Baier, 2014).

Since the manual creation of high-quality training data for digital forensics is tedious, time-consuming and error-prone (Woods et al., 2011), various data synthesis methods have been proposed in recent years to automatically generate realistic digital forensic data sets. A popular way to create such data sets is to use a framework that simulates real user behaviour to automatically create forensically relevant artifacts. However, to the best of our knowledge, there is currently no publicly available framework capable of generating traces at the persistent, volatile memory and network layer simultaneously.

In this paper we introduce `ForTrace`, a holistic synthesis framework for generating forensic data sets to support the digital forensic field with respect to forensic training, tool development and evaluation. The acronym `ForTrace` is composed of the two terms *Forensics* and *Traces*. This specific naming is inspired by the term *Fortress* and is meant to remind us that finding the correct digital traces can be just as difficult as the conquest of a fortress. `ForTrace` adopts a holistic approach to generate forensically relevant data sets, enabling the simultaneous generation of persistent, volatile and network traces. The holistic synthesis is put into practice by firstly creating a model of a realistic system, the so-called scenario, secondly simulating the user's actions and behaviour automatically within a virtual machine (VM), thereby storing the forensically relevant traces in the resulting disk and memory dumps and network captures. These images can then be used as labelled data sets for a forensic investigation, as training data for machine learning, or for a sophisticated forensic tool development and evaluation. By properly simulating human–computer interactions, the resulting artifacts should be indistinguishable from regular user interactions at best. This behaviour is validated by testing various realistic and complex scenarios. In fact, the modular structure of `ForTrace` enables accurate approximation of human–computer interactions for a highly customisable data generation, as well as easy synthesis of large amounts of relevant data (i.e., incriminating traces), as well as non-relevant data (i.e., background noise), using the built-in `ForTrace` *Generator* module. The term 'customisable' refers to the framework's ability to simulate different scenarios depending on the combination of modules chosen in the respective configuration file. This allows the user to configure the synthesis of different scenarios as desired, e.g. in some scenarios s/he could only create permanent traces, while others would leave traces at all levels. In addition, the `ForTrace` *Reporter* component logs the ground truth, i.e., it records in detail all actions performed during the synthesis of a scenario.

`ForTrace` is fully open source and available to the digital forensics community. The latest version of the framework's source code can be downloaded from our GitHub repository: https://github.com/dasec/ForTrace. The official documentation of the framework with detailed installation instructions, details on all currently available features and information on how to add custom code to simulate your own scenarios with any protocols and applications can also be found in the `ForTrace` repository.

The remainder of this paper is structured as follows: Section 2 presents related work focusing on the synthesis of digital forensic evidence. Then, in Section 3 we list properties of a holistic data set framework that we used during the software design of `ForTrace`. In addition, the related work is assessed with respect to these properties. Section 4 shows the current framework architecture and the exact functionality of `ForTrace`, exploring its exact synthesis capabilities based on the *Generator* and *Reporter* components. In Section 5, the data synthesis process with `ForTrace` is demonstrated in more detail using two diverse realistic and complex scenarios. We then evaluate the actual quality of the digital traces generated by the automated user simulation using common digital forensic tools. Finally, in Section 6 we summarise our work and point out open tasks that will extend our data set synthesis framework in the future. We also mention the many opportunities available to the digital forensics community for future work in this area.

2. Related work

In this Section we distinguish between well-known, manually created forensic images and image generators that are used to create forensic images automatically.

2.1. Manually generated images

In digital forensics it is a widespread approach for instructors to spend a lot of time manually preparing disk images, network captures, memory dumps, and other relevant forensic material that can be used, for example, as forensic challenges for training purposes (Garfinkel, 2007). For this purpose, a forensically relevant scenario is often replayed within a VM. Among the most popular publicly shared forensic images are those of Carrier (2021), CFReDS (NIST, 2021), DFRWS (2021), Digital Corpora (Garfinkel, 2007) including the Real Data Corpus[4] (Garfinkel, 2012), Hadi (2021) and UNHcFREG (Grajeda et al., 2017).

However, creating realistic forensic corpora that is plausible, consistent and useful is indeed a complex task that requires extensive planning to achieve the desired outcome (Woods et al., 2011). Manually generated images are often very simplistic due to the manual effort required to perform enough actions in a given scenario. Relevant background noise, that would normally occur in real data sets, is typically missing in synthesised data sets.

Another major drawback of manually created images is the lack of adaptability. Once the image is created, it is static in that it cannot be easily adjusted without recreating the entire image (Göbel et al., 2020). However, a recurring use of static images, especially in education, is mostly not possible. Therefore, this work aims to provide a solution that allows the forensic community to create forensic images as dynamically as possible. For example, modifying the result of an automatically generated image by simply adjusting the configurable parameters of `ForTrace` before data synthesis (e.g., to determine in advance which traces should be present in the image later on) would greatly simplify the reuse of the generated image.

2.2. Automatic forensic image generators

Several attempts have been made in the past to develop tools and frameworks to automate the data synthesis process in digital forensics. We will compare their strengths and weaknesses and thereby explain what is missing and what was our main intention in developing `ForTrace` alongside the existing tools.

The `Forensic Image Generator Generator` (Forensig^2) (Moch and Freiling, 2009, 2012) opens up the possibility to produce generators for forensic images for forensic analysis exercises. The input to Forensig^2 consists of a script written by the instructor. The output consists of a file system image to be analysed by students and an automatically generated human readable report defining the ground truth. A clear advantage of Forensig^2 is that the traces are created in a Qemu-based VM and that it can be extended with own scripts written in Python. This makes it possible to freely configure the system. However, this framework is outdated and no more maintained. Its original website is no longer accessible.

The `Computer Forensic Test Image Generator` (ForGe) (Visti et al., 2015) is a framework capable of creating forensic test images by hiding previously defined target files in a NTFS filesystem besides regular files. It provides a user interface and takes instructions in form of database entries. Its output contains images and information sheets. Although the tool is available on GitHub, it has not been further developed since 2015. Considering our intended holistic approach, it does not provide network captures and memory dumps.

The `Virtual Machine POPulation system (VMPOP)` framework (Park, 2018a) is also designed for populating operating systems or applications, and extracting forensically interesting data from populated systems. `VMPOP` assists the development of system-generated data along with a virtualisation system and elaborate automated VM control. As a practical proof-of-concept Park demonstrates how to automate the generation of a Windows Registry corpus using PowerShell. This work partly influenced us in terms of adding PowerShell as one of the control components to `ForTrace` in order to simplify the generation of Windows artifacts.

`EviPlant` (Scanlon et al., 2017) focuses on the optimisation of forensic scenarios distribution by first providing a base disk image for students and then deliver scenarios as evidence packages, which only contain the difference between the base image and the instructor's modified image. This has the advantage that large files do not have to be sent numerous times, which in particular is of interest for teaching purposes. The last commit to the project was in August 2020, updating the code to be compatible with Python 3. Due to its differential design, `EviPlant` does not provide holistic data, such as network captures and memory dumps.

A more recent work that, besides creating traces on the hard disk, also captures network traces is the `TraceGen` (Du et al., 2021) framework. It focuses on creating disk images with weeks and months of simulated user interaction defined by a user provided script file within a short time. However, the approach is currently handled as a prototype and its source code is not publicly available. To combine the advantages of manual and automatic trace generation, APIs such as *pywinauto* are used to simulate user interactions in the GUI. For example, copying a file via Windows Explorer creates more/other traces than simply copying a file via the command line. (e.g., Thumbnails, Prefetch, etc.). Since this approach currently is only a proof-of-concept, it reads as if all available scenarios are in Python scripts or all individual actions are in lists. This does not yet result in a high level of user-friendliness while generating images, especially for less technically experienced users.

`hystck` (Göbel et al., 2020) is a framework that generates traces on hard disk and network layer. This can either be done partially automated with Python scripts or fully automated via YAML configuration files. Through the support of automated data synthesis, it is possible to create a wide variety of traces with little effort, which can be distributed efficiently due to the division into template and differential images. However, considering our intended holistic approach, `hystck` also lacks in providing a memory dump.

In addition to the aforementioned frameworks, which are all based on peer-reviewed papers, two other tools were found during our literature research. First, `ForGen` (Keighley, 2021), a Ruby based generator for forensic images, creates on the one hand forensic disk images for training purposes, on the other hand individual test sheets that can be worked on by the students. These test sheets can be automatically compared to a mark sheet afterwards which is also created during the generation. The other tool is called `ForGeOSI` which is based on a bachelor thesis and uses a wrapper for *pyvbox*, a Python implementation of the Virtualbox COM API, to control the user behaviour (like keystrokes and browser activity within Windows and Linux VMs). Further it creates a log of the actions performed during an execution. Unfortunately, it seems like both tools are no longer maintained by their authors. Also, there is only little documentation for both tools, so they should be treated with appropriate caution.

In Table 1 we summarise the previously mentioned synthesis tools and provide their name and, if available, their peer-reviewed publication, their release date and date of last update, their programming language, and the link to their Git repository if their

[4] https://wiki.nps.edu/display/DEEP/Data+Sets.

Table 1
Overview of existing data synthesis frameworks.

Framework/Publication	Release	Last update	Language	Code available	Image type
Forensig²/(Moch and Freiling, 2009, 2012)	2009	Never	Python[a]	×	P/-/-
ForGeOSI/-	2014	05/2014	Python 2	✓/(Fragg, 2014)	P/-/-
ForGe/(Visti et al., 2015)	2015	03/2015	Python 2	✓/(Visti, 2015)	P/-/-
ForGen/-	2016	02/2017	Ruby	✓/(Keighley, 2017)	P/-/-
EviPlant/(Scanlon et al., 2017)	2017	08/2020	Python 3	✓/(Du, 2020)	P/-/-
VMPOP/(Park, 2018a)	2018	03/2018	Python 3	✓/(Park, 2018b)	P/-/-
hystck/(Göbel et al., 2020)	2020	04/2021	Python 3	✓/(da/sec, 2021)	P/-/N
TraceGen/(Du et al., 2021)	2021	Never	Python[a]	×	P/-/N

[a] No information about the exact Python version.

Table 2
Properties of existing data synthesis frameworks.

Framework/Publication	OS	Hol	Uniq	Lab	Tim	Act
Forensig² (Moch and Freiling, 2009, 2012)	×	×	N/A	✓	×	×
ForGeOSI	✓	×	×	✓	×	×
ForGe (Visti et al., 2015)	✓	×	×	✓	×	×
ForGen	✓	×	×	✓	×	×
EviPlant (Scanlon et al., 2017)	✓	×	×	✓	✓	✓
VMPOP (Park, 2018a)	✓	×	×	✓	✓	×
hystck (Göbel et al., 2020)	✓	×	✓	✓	✓	✓
TraceGen (Du et al., 2021)	×	×	N/A	✓	✓	N/A

source code is publicly available. The last column specifies the image type that the frameworks can create (P = persistent; M = memory; N = network). One major drawback of the existing frameworks clearly is that there currently is no framework that is able to generate traces on all layers simultaneously. Only `hystck` and `TraceGen` are capable of generating traces at least on two levels (persistent and network), the latter, however, not being publicly available. However, as mentioned earlier, it is usually only through the availability of traces on all three layers that the forensic trainee as well as the practitioner is able to gain a much more comprehensive view of a criminal case. For example, in a malware case, important information such as the actual infection vector or further relevant artifacts that are only in memory would be completely missing. However, those could be the only information that finally helps to solve the case.

3. Properties of a holistic synthesis framework

Based on our discussion in Section 2 about related work and its different strengths and weaknesses, we next gather properties that we consider as reasonable for a holistic data set synthesis framework and that form the basis for our design of `ForTrace`. We then shortly explain our choice in favour of `hystck`.

Grajeda et al. (2017) formulated three critical features of a digital forensic data set to ensure high-quality results. First *availability* means that the data set is publicly distributed by common channels like a website or a software repository to ensure reproducibility. Next *quantity* addresses the aspect that the data set contains a reasonable amount of data. Finally *quality* considers the property that the data set guarantees accurate results due to labels and similarity to real-world data.

We transfer these properties to a data set synthesis framework by extending and formulating more fine-grained properties:

OS *Free and open source availability*: the framework including its source code must be downloadable without registration and fees from a public web resource (e.g., a website or a Git repository).
Hol *Holistic quality*: the framework provides digital forensic traces on different layers including at least persistent, volatile and network traces.
Rel *Realistic quality*: it is hard for a digital forensic user to distinguish the generated traces from real traces.
Templ *Templates*: the framework comes with realistic templates representing forensic relevant use cases, e.g., malware, data breach, data hiding scenarios. Furthermore test-scripts should be provided for a scenario-based evaluation.
Conf *Customisation*: the framework must be easily configurable with respect to the chosen scenario, e.g., through a text-based configuration file or a GUI.
Uniq *Uniqueness of data set*: based on the chosen configuration, the generated data set differs from further data sets generated based on the same configuration since the framework introduces a certain degree of randomness in terms of background noise.
Lab *Ground truth*: the framework outputs a labelled data set, i.e., besides the data set it supplies the respective ground truth with respect to the configured scenario.
Use *Usability*: the framework may be used by an 'ordinary' experienced digital forensic user.
Tim *Timeliness*: the framework generates a data set with respect to contemporary hard- and software.
Act *Activity*: the framework is maintained and used in the digital forensic community.

In Table 2 we assess the presented frameworks of Section 2 with respect to the most important properties of a data synthesis framework. If an assessment is possible the mark indicates if a particular property is respected by the framework or not, respectively. If N/A is assigned, this is because sparse information is available due to the lack of accessible source code, so no definitive statement can be made. The result of the evaluation is that `hystck` is the best choice as a framework basis to be extended to a holistic synthesis framework, as it addresses most of the properties and already generates traces on at least two layers, persistent and network, without focusing on memory artifacts so far.

4. `ForTrace` structure and functionality

We now give details on the modular structure of our data synthesis framework `ForTrace` in this section and provide an overview of its available data synthesis features and additional assistance for the evaluation process.

4.1. Basic framework functionality

The basis of `ForTrace` and its functionality is the simulation of human-computer interaction through the use of a client-server architecture. The server-side host component creates and manages a client-side guest component by using open-source software products, such as *KVM*, *Qemu* and *libvirt*. To synthesise data, the host and client communicate over a separate private network. This ensures separation between the artifacts being generated and the management traffic. To ensure generation of realistic traces on persistent, memory, and network layers, `ForTrace` simulates human-computer-interaction by employing so-called *User Interaction Models*. The framework generates this model on the host-side and executes the respective actions on the guest components to realise a specific and realistic scenario.

The current focus during the publication of `ForTrace` is on using a Linux host machine and Windows 10 guests[5]. Since the `hystck` framework was originally developed in Python 2, the core framework has now been fully migrated to Python 3 compatibility, and since then all our additions via `ForTrace` have been developed using the latest Python 3 version. Python was chosen as it is a largely platform-independent language suitable for our use of both Windows and Linux-based operating systems.

To both allow users to synthesise data quickly and flexibly and to make modifications to the framework, such as adding new modules without having to manually reinstall all parts of the framework, `ForTrace` includes fully automated installation scripts for both Linux hosts and Windows guests (Property: Usability). These scripts install all necessary packages, dependencies and third-party software on both the host and guest side. The host side script also sets up the virtual environment including both the public and private networks. The user only needs to perform few steps with all of them outlined in the `ForTrace` documentation and in the wiki page of the GitHub repository. Furthermore, all relevant configuration files to be considered when installing and using `ForTrace` are documented in the repository, including relevant installation packets, network interfaces and relevant path configurations, etc. (Property: Customisation).

4.2. Framework architecture

As mentioned in the previous section, `ForTrace` uses a basic client-server (or guest-host) architecture and is therefore split into two parts. The basic structure of the framework is depicted in Fig. 1. An important aspect of the client-server architecture are the two separate networks that are configured during the installation process. To synthesise network traffic and have general access to the Internet, a public network is created. Since host and guest exchange a variety of messages during the execution of a scenario, a second private network without Internet connection, is configured. This removes all `ForTrace`-related artifacts from the network dump and allows the user to configure additional services for more convenient testing, e.g., the user can perform file transfers between multiple VMs without affecting the actual network data captured.

The host-side component is called **Framework Master** (represented in code by 2 of the 3 core classes *Virtual Machine Monitor (VMM)* and *Guest*). The **Framework Master** is responsible for creating and managing the guest components as well as initiating the communication over the private network. It also contains the user-created scenarios that are supposed to be executed on the guest component.

The *VMM* class is mainly responsible for organisational tasks. It prepares the environment by cloning the prepared template. Due to licence restrictions, it is not possible to provide Windows guests that are pre-configured. However, `ForTrace` provides detailed description how to set up the templates as well as scripts that fully automate the template installation process. Next, it makes sure the communication path with the guest component is correct by checking the created guest's MAC address. When running a test script, all of these actions will be carried out by the *create_guest()* function.

The other core host-side *Guest* class is mainly responsibly for the actual communication between both components of the framework. It establishes the initial connection with the guest component and then submits the chosen scenario. Within this user-created scenario it is determined which of the `ForTrace` modules will be employed to manipulate specific applications and

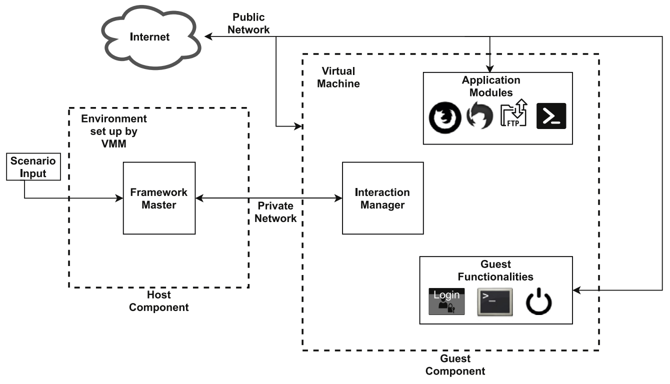


Fig. 1. Basic framework architecture.

functionalities or, if a more direct approach by utilising functions embedded in the *Guest* class are called, to control elementary Windows functions and CLI inputs. It is possible to combine both approaches without restrictions.

The guest component consists of one or multiple VMs, cloned from a prepared template each time a new scenario is executed in `ForTrace`. This ensures that new scenarios are not tainted with artifacts from previous attempts (Property: Uniqueness). They are also responsible for finding and establishing the communication over a predetermined port on the private network as well as for executing the right commands corresponding to the **Framework Master** scenario. Ideally, the component will also send information about the current state of the scenario's execution back to the host component. Therefore, this component is called **Interaction Manager**.

It is represented by the third core class called *Agent*. On Windows, the agent is executed via batch script every time the guest VM starts - this mitigates the need for manual interference in the execution of the chosen scenario. The *Agent* class will first establish the private network communication channel. When the connection is confirmed, the scenario constructed on the host-side is converted into necessary actions which are then executed. Many of these executed actions will be followed by a current status report that is sent back to the **Framework Master**.

Fig. 2 conveys a typical `ForTrace` workflow. After preparing and installing the framework on the host machine and creating a template for the guest VM, a user can run the chosen scenario. The *VMM* class will clone the template and boot the cloned VM. Subsequently, the *Guest* class will load the *User Interaction Model*. When the guest VM is running and the automatic user login has passed, the *Guest* and *Agent* classes will continuously attempt to establish the connection between the two components over the private network. As soon as the connection is established, the scenario is transferred to the client, transformed into executable actions and executed by the **Interaction Manager**. To test the correct execution of different scenarios and validate their various artifact generations, test-scripts for each `ForTrace` feature are provided together with its source-code. Anyone can run these test-scripts on their self-created templates with current OS version without having licensing issues. When the entire scenario is finished, `ForTrace` begins shutting down the guest component. If the user has chosen to extract one or multiple memory dumps, the dumps will be created at the chosen time during the scenario to ensure that the desired artifacts can be found later in the image. The memory dumps are then stored in a specified location before the machine is shut down. The network traffic on the public interface is always captured using `tcpdump` and is stored in the location specified in the configuration file. Unless a deletion of the

[5] Testing on Windows 11 is currently ongoing and full support will be provided as part of a timely update in the GitHub repository.

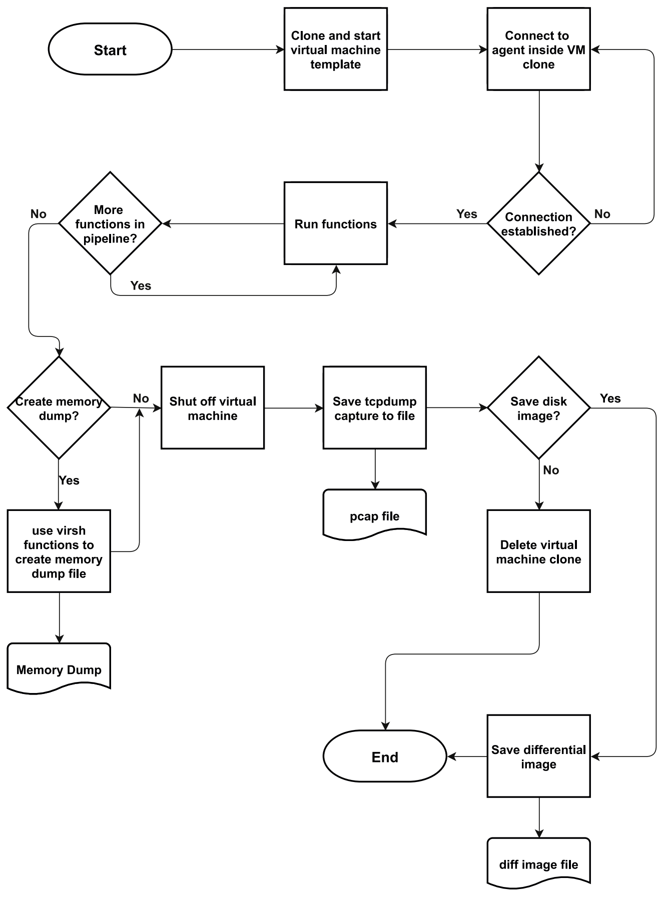

Start
Clone and start virtual machine template
Connect to agent inside VM clone
Connection established?
Yes
No
Run functions
More functions in pipeline?
No
Yes
Create memory dump?
No
Yes
use virsh functions to create memory dump file
Memory Dump
Shut off virtual machine
Save tcpdump capture to file
pcap file
Save disk image?
Yes
No
Delete virtual machine clone
End
Save differential image
diff image file

Fig. 2. Standard `ForTrace` workflow.

guest VM is explicitly instructed, the differential persistent image is of course also saved by default including the traces of all actions performed.

4.3. *Data synthesis features*

To allow the execution of a wide spectrum of different scenarios, `ForTrace` is able to simulate different kinds of user behaviour and application usage. A complete overview of all currently supported data synthesis features can be seen in Table 3 in Appendix A.

The goal of all scenarios is to synthesise traces in at least one layer - better, however, in all three layers, network traffic, memory, and persistent storage (Property: Holistic quality). While the network traffic of the public network is continuously recorded, persistent storage and memory dumps are gradually formed by simulating specific user actions. In order to have a more realistic experience later during the forensic analysis, larger time-spans can be simulated by changing the system date and time of the VMs, allowing the user to simulate multiple days of system use during a single session (Property: Realistic quality). The time is changed via the hypervisor between different user sessions while the guest is offline, so that it looks to the system (and later to the analyst) as if time has simply passed since the last logon. This approach prevents unwanted messages in the Event Log that would occur if the clock would be changed directly in Windows.

The *Generator* component included in `ForTrace` enables the generation of large amounts of traces without the necessity of coding complex Python scripts. It almost completely removes the requirement for advanced Python programming knowledge when using the `ForTrace` framework. Therefore, a *YAML* file is fed to the *Generator*, in which applications, actions, and distinctions between benign and malicious data and actions to be performed, can be defined. The *YAML* tags determine which modules and functions should be called, while also aggregating all necessary parameters for these functions. The following execution of these actions is identical to the execution of specific Python scenario scripts, but instead of writing the scripts manually, the synthesis is now performed fully automated and can easily be adjusted as desired via the *YAML* file (Property: Customisation). In addition, the benign part of the *YAML* file creates a haystack of 'normal' human-computer interaction (i.e. important background noise) in the resulting data sets (Property: Realistic quality). To create a changing amount of prevalent artifacts, an additional 'random' flag can be set in the configuration file so that a large number of randomly selected different websites are visited or randomly generated emails are sent (Property: Uniqueness).

The following list provides an overview and explanation of the most prominent `ForTrace` modules. For a complete overview of all currently implemented data synthesis features, please refer to Table 3 in Appendix A.

- **Elementary functions:** Many elementary OS functions are implemented as part of the aforementioned core classes *Agent* and *Guest*. These functions include, but are not limited to shutting down the VM, basic file and directory operations, manipulation of the system clock and CLI functionality. The latter, for example, implicitly allows SSH communication and therefore additional file transfer protocols such as SFTP, too.
- **Multi-user capability:** Multiple users can be created and deleted on the guest during the scenario execution. Further, it is possible to change the user who is executing the desired operations in an automated way (i.e. log a user in and out). This allows the execution of more complex multi-user scenarios, like shared workstations and using the impact of one user's operations on the other user.
- **Anti-forensic capabilities:** To control the amount of artifacts created by relevant operations during a scenario (so-called needles), `ForTrace` provides different features to either prevent the creation of typical Windows artifacts in advance or delete these artifacts afterwards. Avoiding the creation of artifacts is mostly done by changing related Windows Registry keys to disable the artifact-creating services. This approach, for example, allows to disable well known artifact sources on Windows, like the Windows Thumbcaches or Prefetch data. If the creation of artifacts can not be suppressed, `ForTrace` also offers a way to wipe already created artifacts by either deleting related Registry keys, like the UserAssist key, or by secure deleting files containing the relevant artifacts, like Prefetch or Thumbcache files.
- **PowerShell support:** PowerShell is an important and powerful tool in modern Windows administration with growing popularity that has even led to Linux PowerShell integration. Due to its vast capabilities, a PowerShell module has been added to `ForTrace`. This module enables the execution of elementary and advanced system, file, and application functions, such as searching the system for keywords, unattended (un)installation and execution of software, deleting files and directories, connecting to a network share, (un)mounting USB devices, etc.
- **Browser simulation:** Currently, this module's realisation is assisted by Mozilla's own *marionette driver*. The module is capable of simulating the most common user habits, such as browsing websites, login to an account (e.g., on Facebook,

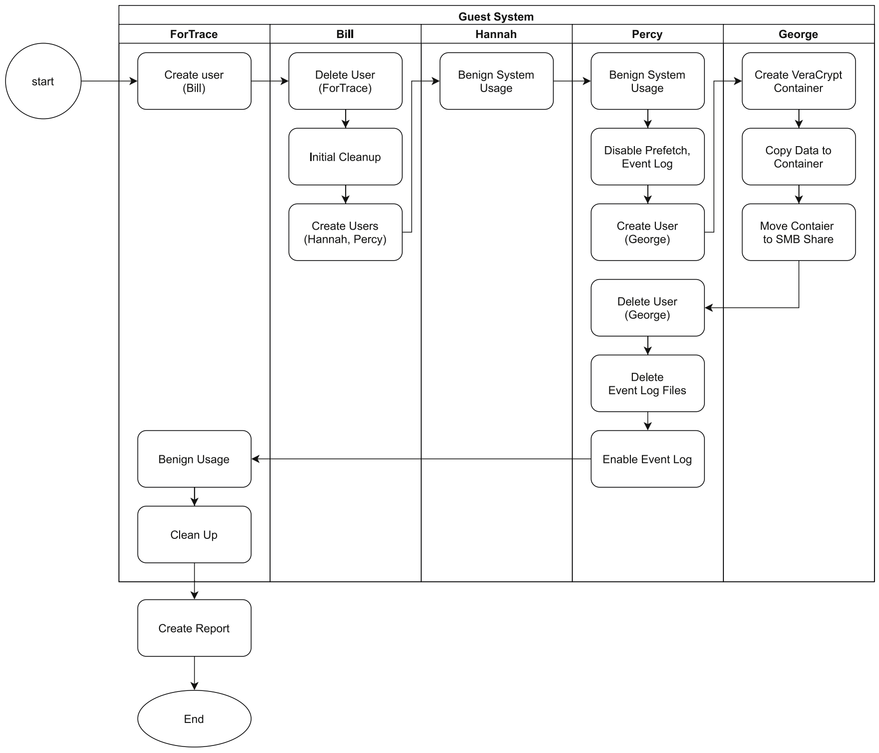


Fig. 3. Sample scenario: Exfiltrating data on a multi-user workstation.

Instagram, etc.), downloading data as well as clicking specific elements. Both *marionette driver* and the `ForTrace` Firefox module employ multiple ways of navigating a website, e.g., using a website element's xpath or id. Further browser support is currently being developed for Chrome and Microsoft Edge.

- **Email simulation:** This module is capable of simulating the most common user actions, such as login to an email account, sending and receiving emails with or without attachments in Thunderbird. Additionally, the module allows the user to artificially fill up its mailbox to simulate previous traffic and create a more immersive and cohesive experience, which could be beneficial in an educational environment for students (Property: Realistic quality). Further email client support is currently being developed for Microsoft Outlook.
- **Encypted container support:** This module currently enables some elementary VeraCrypt functionalities such a creating, mounting and dismounting encrypted containers as well as transferring data into and out of the containers.
- **File transfer support:** This module enables basic file transfer methods, such as transferring files or directories via the network and over the SMB, NFS, FTP, or SFTP protocols. To configure the required server component, this can be realised by either installing the corresponding service directly on the host component or by simply employing an additional service VM, which is available within `ForTrace` by default and hosts various services required by the individual scenarios (Property: Customisation).
- **Printer support:** This module implements print queues for a digital network printer. Enabling document printing thereby allows the generation of appropriate printer traffic in the network. Similar to the file transfer module, this requires the user to set up a digital printer using a service like CUPS either on the host component or use the printing service out-of-the-box on the supplied service VM.
- **Malware synthesis:** This module is able to generate typical traces left by common malware variants. It is split into two components: The C&C server component, that is fully configurable, and the malware client component. The malware is able to execute commands sent by the server. It hooks itself to the system using a variety of persistence mechanisms and carries out various malicious actions, like process hollowing or malicious code injection (compare with our validation in Section 5.2).

4.4. Report of relevant actions performed

`ForTrace` aims to generate large amounts of traces, especially for education purposes and tool testing. A major part of this generation is the creation of a ground truth for every scenario run. The framework manages this by maintaining all relevant executed operations and the guest-side timestamps related to them. After the executed scenario is finished, it creates a report in form of a XML document, containing all logged information, sorted by the related module as well as in a chronological timeline (Property: Ground truth). In addition, `ForTrace` also contains a component that parses the XML document into a user-friendly HTTP view.

In addition to the explicit report of all executed actions, `ForTrace` also records all network traffic generated during the runtime of each guest by using `tcpdump`. Besides the scenario-specific traffic, these captures do also include all kind of Windows-related network traffic (e.g., realistic background noise, such as Windows updates, firewall or anti-virus traffic, etc.). The *pcap* file is stored in the path specified in one of the configuration files.

As already mentioned, `ForTrace` also allows for memory extraction. The framework uses *KVM* functionalities for the memory acquisition. If a memory dump is required, this can be explicitly mentioned either in the test script or in the *Generator*. As `virsh` is capable of creating other dump files, it is easily possible to expand the framework's data extraction functionalities.

5. Validation of the `ForTrace` framework

As stated before, `ForTrace` aims at simulating realistic system usage in order to create a holistic data set including all relevant forensic artifacts. In the following section two different forensic scenarios are presented, as well as their realisation by using the `ForTrace` framework. All of the following scenarios are executed using custom Python test-scripts, all of which are included in the `ForTrace` repository. Since `ForTrace` is designed and developed on base of the holistic data synthesis properties presented in Section 3, further scenarios may easily be put into practice. Some examples of such eligible forensically relevant scenarios in the context of `ForTrace` are discussed in Table 4 in Appendix C (Property: Templates). For simplicity, we call the attacker *Mallory* and the victim *Alice*. The table also lists respective data sources (e.g., Windows Registry, RAM dump, etc.) where a forensic investigator would actually find the relevant artifacts after an incident. Given the diversity of places where relevant artifacts typically are stored, it should get clear why it nowadays is important to keep track of multiple data sources to fully solve a complex digital forensic case.

5.1. Validation of Windows artifacts

The first scenario demonstrates the capabilities of the framework regarding *multi-user scenarios* and *anti-forensic methods*. It demonstrates a shared workstation with benign and malicious users, where data is extracted with the help of a VeraCrypt container and a SMB share. Fig. 3 shows relevant actions performed during the scenario, where a system administrator creates different users, of which each is performing an individual set of operations. One of these users exfiltrates critical data, by first creating another system, as well as disabling different critical services, such as the Windows Event Log and the creation of Prefetch files, to drastically reduce the amount of created artifacts. The new user uses a VeraCrypt container to exfiltrate the critical data in a subtle way. After this, the malicious user is deleted again and its user data is deleted securely. This is done to wipe as many artifacts as possible that point to the existence of this user. Finally, the disabled services are re-enabled, so it will be less suspicious in the following forensic analysis.

Listing 1 is a small snippet of the Python script used to execute this scenario. This code example shows the deactivation of multiple Windows services as well as the creation of a new user, who then mounts and copies files to a VeraCrypt container which is then exfiltrated to a network share. While it is making use of multiple modules, such as *Anti Forensics*, *VeraCrypt* and *User Management*, all modules are called using the *ScenarioHelper* utility class. This class has two major functionalities:

- All modules and functions are aggregated into a single class and therefore a single import, allowing greater flexibility when writing multi-module test scripts for `ForTrace`.
- The *ScenarioHelper* adds more and clearer information to the `ForTrace` *Reporter*, making it easier to retrace and comprehend the scenario afterwards. An excerpt from the report for this scenario can be seen in Listing 2.

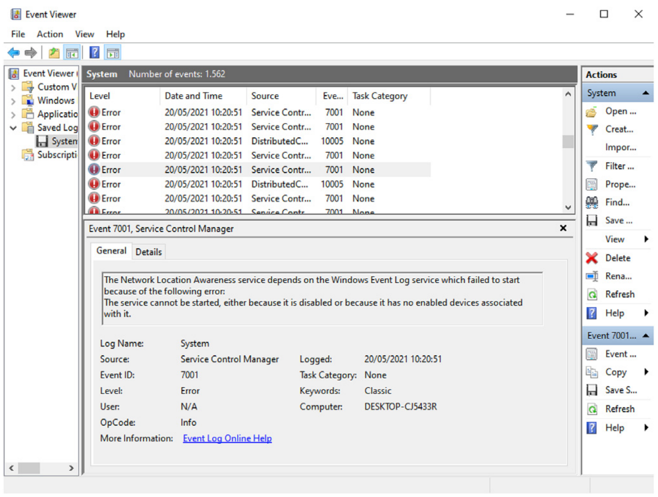
Listing 1. Multi-user scenario script example code.

```
[...]

####### Session 3 - Percy

# anti-forensics
sc.disablePrefetch("1", "needle")
sc.disableEventLog("1", "needle")

# add malicious user
sc.addUser("George", "enj_Jwaa", "needle")
sc.changeUser("George", "enj_Jwaa", "needle")

# reboot
sc.reboot()

####### Session 4 - George

[...]

 # Copy files to the VC container
 sc.mountContainer(cont_p, cont_pass, "X", "needle")
 sc.copyToContainer(george_document1_t, "X", "needle")
 sc.copyToContainer(george_picture1_t, "X", "needle")
 wait(5,15)

 # Unmount the container
 sc.dismountContainer("X", "needle")
 wait()

 # exfiltrate data via "public" SMB share

 [...]

 sc.smbCopy(cont_p, smb_tar, smb_user, smb_pass, "needle")

 [...]
```

Listing 2. Excerpt from the report on the multi-user scenario.

```
 [...]
<management>
        <user>[19/08/2021 10:57:47 UTC+2] User Bill deleted
            user: hystck; File deletion: secure </user>
        <user>[19/08/2021 11:00:29 UTC+2] User Bill created
            new user: Hannah; pass: V4PIfkkRZ </user>
        <user>[19/08/2021 11:00:49 UTC+2] User Bill created
            new user: Percy; pass: fkkRZIkm </user>
        <user>[19/08/2021 11:02:02 UTC+2] User context will
            be changed from Bill to: Hannah on next reboot <
            /user>
        <user>[20/08/2021 08:30:53 UTC+2] User context will
            be changed from Hannah to: Percy on next reboot
            </user>
        <user>[20/08/2021 10:17:56 UTC+2] User Percy created
            new user: George; pass: enj_Jwaa </user>
        <user>[20/08/2021 10:17:58 UTC+2] User context will
            be changed from Percy to: George on next reboot
            </user>
        <user>[20/08/2021 10:23:04 UTC+2] User context will
            be changed from George to: Percy on next reboot
            </user>
        <user>[20/08/2021 18:00:25 UTC+2] User Percy deleted
            user: George; File deletion: secure </user>
        <user>[22/08/2021 07:17:45 UTC+2] User Bill deleted
            user: Hannah; File deletion: keep </user>
</management>
 [...]
```

The evaluation of the artifacts created by this scenario reveals a few interesting characteristics. While deleting Event Log entries and disabling the Event Log obfuscates the actual malicious activity, the presence of malicious activity will, however, not be completely hidden as security auditing is still running and error messages in



System   Number of events: 1.562

```
Level    Date and Time        Source           Eve...   Task Category
Error    20/05/2021 10:20:51  Service Contr...  7001    None
Error    20/05/2021 10:20:51  Service Contr...  7001    None
Error    20/05/2021 10:20:51  DistributedC...  10005    None
Error    20/05/2021 10:20:51  Service Contr...  7001    None
Error    20/05/2021 10:20:51  Service Contr...  7001    None
Error    20/05/2021 10:20:51  DistributedC...  10005    None
Error    20/05/2021 10:20:51  Service Contr...  7001    None
```

Event 7001, Service Control Manager

The Network Location Awareness service depends on the Windows Event Log service which failed to start because of the following error:
The service cannot be started, either because it is disabled or because it has no enabled devices associated with it.

Log Name:	System		
Source:	Service Control Manager	Logged:	20/05/2021 10:20:51
Event ID:	7001	Task Category:	None
Level:	Error	Keywords:	Classic
User:	N/A	Computer:	DESKTOP-CJ5433R
OpCode:	Info		
More Information:	Event Log Online Help		

Fig. 4. Suspicious Event Log entry after disabling service.

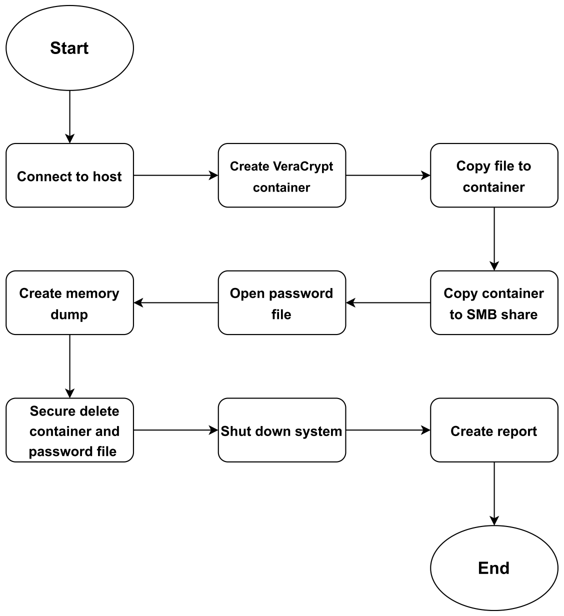
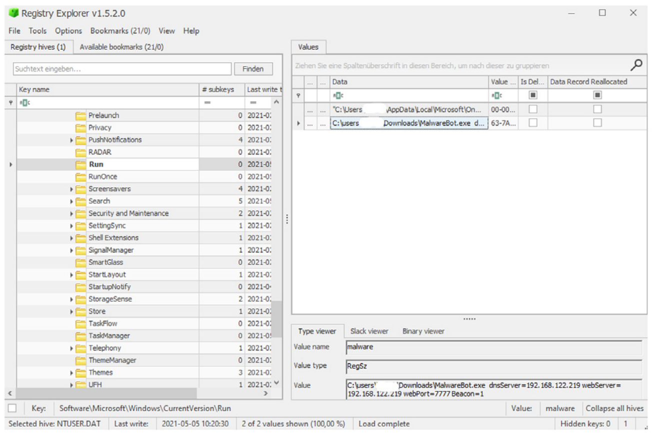


Fig. 5. Adaption of the multi-user scenario with additional memory acquisition.

the system log will hint the analyst at disabled services. This can be seen in Fig. 4. A more effective way to hide malicious activity can be achieved with `ForTrace` by simply disabling the *Event Log*, secure deleting the corresponding files and then re-enabling the service.

Fig. 5 depicts a small change to the final steps of the previously explained multi-user scenario to get an idea of how easily and flexibly an existing scenario can be changed by the dynamic configuration file in `ForTrace`. The simulated user creates a VereCrypt container to hide confidential or illegal material. The container is then moved from the local system to a SMB share. While the malicious user inspects a password file, a memory dump is created for later inspection. Despite secure deletion of all persistent traces, such as the password file and the VeraCrypt container, much of the information can still be recovered using the network dump and the memory dump provided by `ForTrace`, as long as the forensic analyst uses the appropriate tools. For example, during our evaluation of this scenario we were able to reconstruct the password we were looking for from RAM using Volatility. In this way, the scenarios in ForTrace can be easily adapted to provide


Registry Explorer v1.5.2.0
File Tools Options Bookmarks (21/0) View Help
Registry hives (1) Available bookmarks (21/0)
Suchtext eingeben... Finden
Values
Ziehen Sie eine Spaltenüberschrift in diesen Bereich, um nach dieser zu gruppieren
Type viewer Slack viewer Binary viewer
Collapse all hives

Key name	# subkeys	Last write t
Prelaunch	0	2021-0
Privacy	0	2021-0
PushNotifications	4	2021-0
RADAR	0	2021-0
Run	0	2021-0
RunOnce	0	2021-0
Screensavers	4	2021-0
Search	5	2021-0
Security and Maintenance	2	2021-0
SettingSync	1	2021-0
Shell Extensions	1	2021-0
SignalManager	1	2021-0
SmartGlass	0	2021-0
StartLayout	1	2021-0
StartupNotify	0	2021-0
StorageSense	2	2021-0
Store	1	2021-0
TaskFlow	0	2021-0
TaskManager	0	2021-0
Telephony	1	2021-0
ThemeManager	0	2021-0
Themes	3	2021-0
UFH	1	2021-0

...	...	Data	Value ...	Is Del...	Data Record Reallocated
...	...	"C:\Users[redacted]\AppData\Local\Microsoft\On...	00-00...	☐	☐
...	...	C:\users[redacted]Downloads\MalwareBot.exe d...	63-7A...	☐	☐

Value name	malware
Value type	RegSz
Value	C:\users[redacted]Downloads\MalwareBot.exe dnsServer=192.168.122.219 webServer=192.168.122.219 webPort=7777 Beacon=1

Key: Software\Microsoft\Windows\CurrentVersion\Run   Value: malware

Selected hive: NTUSER.DAT   Last write: 2021-05-05 10:20:30   2 of 2 values shown (100,00 %)   Load complete   Hidden keys: 0   1

Fig. 6. Malware persistency found in the Registry.

different levels of difficulty for the subsequent forensic analysis, i.e., some of the scenarios can already be solved at the persistent layer, while others definitely require a memory or network dump (Property: Customisation).

5.2. Malware synthesis

The second validation scenario demonstrates the ability of `ForTrace` to distribute and control malware and give an overview of the synthesised artifacts. Fig. 7 in the Appendix B depicts a general workflow of `ForTrace`'s ability to synthesise traces of malware distribution and execution. The first step is the choice of an infection vector / dropper for the malware. Currently, four different methods are supported. The malware can either be distributed via download, by using `curl` or the Firefox browser, via an email attachment or via a phishing email with an embedded malicious office macro that can also be used to drop the malware onto the system. An example office macro can be seen in Listing 3.

Listing 3. Office macro code snippet.

```
Sub AutoOpen()
Dim WinHttpReq As Object
Set WinHttpReq = CreateObject("Microsoft.XMLHTTP")
WinHttpReq.Open "GET", "http://192.168.103.111:8080/
    MalwareBot.exe",
    False, "", ""
WinHttpReq.send

If WinHttpReq.Status = 200 Then
    Set oStream = CreateObject("ADODB.Stream")
    oStream.Open
    oStream.Type = 1
    oStream.Write WinHttpReq.responseBody
    oStream.SaveToFile "MalwareBot.exe", 2
    oStream.Close
End If
    Call Shell ("cmd.exe /c" & "C:\users\ForTrace\Downloads\
        MalwareBot.exe
    dnsServer =192.168.103.111 webServer =192.168.103.111
        webPort =7777
    Beacon =3", vbNormalFocus )

End Sub
```

Next, the malware's persistence mechanism needs to be chosen. Again, there are multiple methods supported by `ForTrace`. For example, it is possible to use *search order hijacking* which would modify existing Windows DLLs with malicious code, running the malware at boot time or creating a new Windows service, resulting in a new Registry entry. The method chosen for the following example is the direct creation of a Registry entry, that contains the path of the executable, resulting in the malware being executed at boot time. An example Registry entry extracted and verified using *Registry Explorer* can be seen in Fig. 6. Another artifact left by this persistence mechanism is found in the system's memory. When a Windows system is booted, it executes *winlogon.exe* which in turn will launch *userinit.exe*. This process will start any programs marked for launch at boot time. Any suspicious programs like the malware used will also be present here and can therefore be looked up in the Registry by the forensic analyst to find the used persistence mechanism.

Due to the variety of possibilities when executing the malware synthesis functions of `ForTrace`, many different types of hard disk traces can be synthesised. Due to lack of space, we cannot cover them all in detail here. To exemplary demonstrate the feature, the *Generator* mentioned in Section 4.3 is used to simulate a phishing attack on the supply chain of a company. The example *YAML file* can be seen in Listing 4. A simulated attack on the supply chain is chosen since directly attacking critical infrastructures is becoming continuously harder since the protection of end systems has been significantly improved. In this scenario, further code is downloaded to collect information about the system, saving it in files to be exfiltrated later. These files are marked as hidden before they are deleted to make it harder to trace them. The infection is simulated by generating a phishing mail, with multiple mails with similar attachments placed in the systems outbox folder, suggesting that the initial infection spread to further users via email. For certain file operations, additional traces can be found in the system's memory if it is acquired at the right time. For this scenario, this specifically applies to the download of the additional code as well the upload of the gathered system information.

Listing 4. Phishing attack simulation.

```
[...]
collections:
        c- malware -1:
        type: malware
        commands: /home/fortrace/examples/generator/
             collections/malware3.txt
        email: /home/fortrace/examples/data/email_lure.xml
settings:
        host_nfs_path: /data/fortrace_data
        guest_nfs_path: Z:\\
applications:
        malware-1:
                type: malware
                service -vm: 192.168.103.111
                webServer: 192.168.103.111
                dnsServer: 192.168.103.111
                webPort: 7777
                service-port: 8080
                beacon: 3
                name:lure.docm
needles:
        h-malware-2:
                application: malware-1
                collection: c-malware-1
scripts: [...]
```

5.3. `ForTrace` *artifacts*

One of the key challenges of all synthetic data sets is to reduce the number of artifacts created by the data synthesis framework to a minimum without deleting relevant traces that may be required for the forensic analysis of the executed scenario. When evaluating different scenarios, a few different types of artifacts can be discovered:

- **File system artifacts**: The framework leaves a few persistent artifacts, first and foremost the actual installation folder including example scripts and the entire code base as well as a link inside the Windows startup/autostart folder to a batch script used to start the agent. But there are a few more delicate and not so obvious artifacts, too. Since the `ForTrace` is installed on the system, there are traces in the Python installation directories. Additionally, some artifacts can be found in the *Prefetch* directory, e.g., if a module uses `regedit` or `psexec`.
- **Registry artifacts**: The most important Registry artifact is created by the *changeUser()* function. The Registry key *HKLM\Microsoft\Windows NT\CurrentVersion\Winlogon* reveals the username and password of the last active user in plain text.
- **Event Log artifacts**: The persistent Event Log artifacts are among the most pervasive. As demonstrated in section 5.1 and Fig. 4, even if the Event Log is disabled, logging and security auditing occurs. Additionally, if the Event Log is not cleared, there may be events hinting at the template's installation and possibly inconsistent system times if they were changed during the scenario.

To reduce the created artifacts, two steps have been implemented into `ForTrace` and usually should be added in each scenario. First, the default user should be deleted at the beginning of the scenario. Afterwards the guest function *initClean()* should be used to remove artifacts within the *Event Log* and the *Prefetch* directory, which were created during the template preparation and the beginning of the scenario. Second, the guest function *cleanUp()* should be called at the end of a scenario. It removes the `ForTrace` installation directory for every user created during the scenario, except of the active user which still needs the framework. The cleanup process can be called in manual mode allowing the deletion of the previously mentioned remaining artifacts. A step-by-step manual is provided within the framework documentation. Additionally, there is a list of commands that can assist the user in manually cleaning unwanted synthesis artifacts using different tools like CCleaner. Future work on `ForTrace` will address the prevention as well as clean up processes of additional synthesis artifacts in greater detail.

6. Conclusion and future work

Diverse and extensive data sets of high quality are essential for training both forensic practitioners and machine learning algorithms. This has long been an unsolved problem in the forensics community, as existing data sets often have significant drawbacks and manually synthesising data sets is a tedious task. More recently, attempts have been made to develop initial tools that reduce the time required for this task and assist in the generation of forensically relevant data. Since many existing tools are either focused on specific types of data synthesis, lack the ability to create traces on multiple layers, or are not publicly available, `ForTrace` was developed as an open-source, modular and holistic data synthesis framework for the digital forensic field. `ForTrace` is able to synthesise correlated memory, disk and network traces in different ways, allowing the creation of different data sets through the use of forensically relevant and realistic scenarios, while simulating the human—computer interaction through the use of a client-server architecture. The framework already supports a large number of ways to manipulate a Windows system through diverse anti-forensic measures and the synthesis of typical malware behaviour. It further allows for seamless integration of new functions due to its modular framework architecture. Meeting the properties specified at the beginning of this paper, the framework facilitates forensic trace generation on Windows VMs through a variety of ready-to-use test scripts or through freely configurable YAML scripts. To generate suitable background noise, the *Generator* component can be used, which automatically generates a realistic amount of irrelevant background artifacts, such as random websites visited, content downloaded, emails sent and received, thus distracting the forensic examiner from the actual incriminating data.

With its modular structure, it is a continuous task to extend the capabilities of the framework and keep the existing functions up to date. Possible new features under development include, for example, the creation of network and file transfer traces through the development of modules capable of synthesising the use of modern voice and text communication tools such as Skype, WhatsApp, Facebook Messenger, Microsoft Teams or Discord. In addition to the completion of new modules, the expansion of existing modules is another important task. Besides adding more mail and browser applications, such as Outlook, Edge and Chrome, a special focus is on extending the already existing PowerShell module, as most tasks on Windows can actually be automated via PowerShell, easily providing a variety of forensically relevant artifacts. Whenever we develop new features, we always try to have as few dependencies on app-specific APIs as possible. Also, only when it is not possible otherwise, we fall back on GUI automation. Therefore, PowerShell could also help to develop a generic approach to interact with many apps in the VMs independently of app-specific APIs.

The more functions we add to `ForTrace`, the more important a graphical user interface becomes. Therefore, in the near future, we intend to develop a suitable front-end that will allow the user of the framework to synthesise forensic images not only on the command line, but also with a GUI. Another more general extension of the framework to support multiple guest (and possibly host) operating systems (e.g., Linux, macOS) is also among the tasks currently under consideration. At the time of writing, there already exists a set of basic features that can perform similar tasks for Ubuntu guest VMs as we have shown here for Windows. In addition, we have started to develop an Android interface for `ForTrace` in order to simulate different scenarios on a mobile operating system as well and thus to simplify the generation of mobile forensic images. This task is of particular interest, as there is an even greater lack of available images in the field of mobile forensics as in the field of disk, memory, or network forensics (Ceballos Delgado et al.,2021; Gonçalves et al., 2022).

Another very important task is the accurate detection and reduction of `ForTrace`-specific synthesis artifacts as mentioned in Section 5.3. While there are some initial considerations presented in the paper, this part is not detailed enough and definitely will be the focus of future work. Therefore, tasks such as exploring ways to prevent or effectively detect and remove framework-related artifacts in an automated and less intrusive way, as is currently done with calling functions to remove certain traces, are a high priority. In addition to persistent storage, these considerations must also take into account main memory and network traces.

In its current state, the `ForTrace` framework has several modules that users can combine together to simulate different scenarios. This enables it to synthesise realistic data at scale, using a variety of Windows-internal functions and applications, ubiquitous network protocols, third-party applications, and simulated malware samples. As a major unique selling point, `ForTrace` not only creates persistent disk images, but also simultaneously captures volatile network traffic and dumps memory data during and after running a user-scripted scenario.

Appendix A `ForTrace` data synthesis function overview

Table 3
Available data synthesis functions in `ForTrace` related to Windows systems.

Module	Available functions/user actions
Guest/Agent/VMM (core modules)	Create/Close a VM
	Establish connection to VMs
	Start/Shutdown/Restart
	Execute modules
	Execute arbitrary commands via CLI/Linux Bash (e.g., SSH)
	Set OS date and time
	Send keystrokes
	Create network traffic dump (automated)
	Create memory dump
File System	Copy/Move/Delete files/folders
	Change directory
	Empty recycle bin
	Secure delete files/folders (SDelete)
File Transfer	Transfer files between guest and SMB share
	Transfer files between guest and (S)FTP share
	Transfer files between guest and NFS share
User Management	Add/Delete/Change local accounts
	Logon/logoff of the desired user
PowerShell	Install/Uninstall a program
	Launch/Terminate a program
	Enable/Disable UAC
	Open Windows Explorer
	Search for a keyword
	Attach/Detach USB devices
	Connect to/Mount/Unmount a network drive
	Basic file and folder manipulation
Printer	Set up (software) network printer
	Print files
Anti Forensics	Disable/Delete Event Log entries
	Disable Hibernation file
	Disable Page file
	Disable/Empty Recycle bin
	Disable/Delete Prefetch files
	Disable/Delete Recent files
	Disable/Delete Thumbcache
	Disable/Delete MRUs/User Assist
	Disable/Delete File History or Volume Shadow Copy
	Clear Jump lists
	Set/Manipulate/Delete arbitrary Registry keys
Malware Synthesis	Set up environment (Web server, DNS server, C&C server)
	Deliver malware (via dropper, email, download)
	Use persistence mechanisms (search order hijacking, service creation, Registry manipulation)
	Execute various commands (e.g., upload, download)
Firefox	Open/Close application
	Browse to one/multiple (specific or random) websites
	Perform downloads and ”right click save as” operations
	Click elements via ID or xpath
	Perform logins (Facebook, etc.)

(continued on next page)

Table 3 (*continued*)

Module	Available functions/user actions
Thunderbird	Open/Close application
	Add IMAP/POP3 accounts
	Send/Receive emails w/or w/o attachments
	Fill up mailbox file artificially
VeraCrypt	Create an encrypted container
	Mount/Unmount encrypted container
	Transfer data to encrypted container
Pidgin	Instant Messaging via IRC, Jabber, Bonjour, etc.

Appendix B Malware synthesis workflow

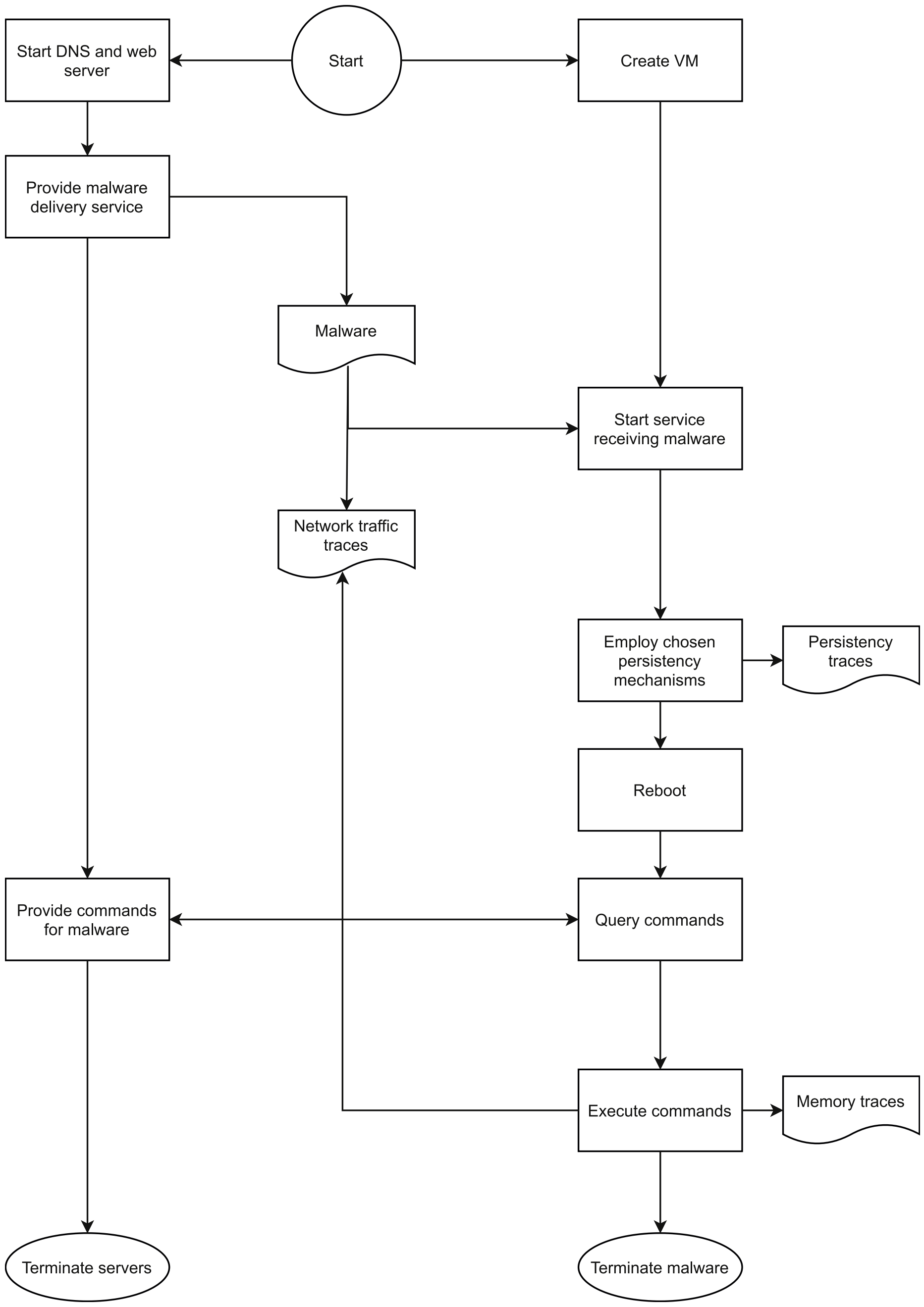


Fig. 7. General workflow of a malware synthesis scenario.

Appendix C Exemplary `ForTrace` scenarios

Table 4
Exemplary forensically relevant scenarios in the scope of `ForTrace`.

Scenario	Attacker Mallory	Victim Alice	Related forensic artifact sources
Data breach (internal actor)	Employee *Mallory* steals confidential internal company data from a network drive and copies it to a USB flash drive.	–	Registry, Event Log, File System
Data breach (external actor)	External attacker *Mallory* distributes malicious code. The malware selects all relevant documents by file type and transfers them to an external storage.	Company employee *Alice* is tricked into opening an email attachment containing the malicious code.	Email Database, Event Log, RAM, Network Traffic
Preservation of evidence	*Mallory* harasses co-workers in private Skype chat sessions at the office.	*Alice* files a complaint with her superiors who order a forensic examination of the affected computers and Skype accounts to gather evidence for possible disciplinary action.	Skype Database, RAM, Network Traffic
Ransomware	*Mallory* sends zipped ransomware attachments hidden in an Office macro. The ransomware creates an encryption key and encrypts specified files by name or extension once it is executed.	*Alice* opens the email and extracts the attachment, executing the ransomware by opening the Office file. Each time Alice logs in, she gets a message that her files were encrypted and a ransom is to be payed.	File System, Email Database, RAM (decryption key), Network Traffic
Phishing	*Mallory* sends emails with an attached HTML document. The HTML document is a website that is stored in temporary storage and opened locally to circumvent potential red flags like missing certificates.	*Alice* opens the email and HTML document. The websites displays a login page, but when the victim enters her credentials, they are sent to the attacker *Mallory* who uses them to further propagate the phishing email via *Alice's* email address.	Browser Database, Email Database, Network Traffic, File System, RAM
Password theft/Trail obfuscation	*Mallory* extracts local passwords with a tool like *mimikatz*. If a victim has chosen to save their credentials after logging into a service, the password can be read in plain text. These credentials can then be used by *Mallory* to log into the victim's account and steal data.	*Alice* connects to a network share and chooses to save her credentials for future logins.	Registry (UserAssist, MountedDevices, etc.), Prefetch/Recent Files, Event Log, Network Traffic
Drug trafficking	Drug dealer *Mallory* stores a customer list and supplier contacts on his computer in an encrypted VeraCrypt container. He uses DuckDuckGo to find out about hiding information in other files via steganography. The details of drug exchanges (e.g., location, order) are then hidden in a picture. Previously, however, he inadvertently searched for the location via Google Maps and stored a screenshot of the location locally.	–	File System, Thumbcaches, Email Database, Browser database, RAM
Data hiding	*Mallory* owns criminally relevant files and tries to bypass forensic tools from recognising these files using for example hash values or contents. He tries to hide the files using internal file system structures like slack and reserved areas, storing the files in encrypted containers, and deleting some of the files securely.	–	Thumbcaches, File System, Registry, RAM (container content, encryption keys)
Data prevention	*Mallory* uses the Windows Registry to reduce traces left on his system. Therefore, he disables the creation of prominent artifacts like Thumbcaches and Prefetch files. He then executes and moves files to an encrypted container on an external drive. After extracting the data, he re-enables the previously disabled services.	–	Registry, Jump Lists, Event Log, RAM

References

Abt, S., Baier, H., 2014. Are we missing labels? a study of the availability of ground-truth in network security research. In: 2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security. BADGERS), pp. 40–55. https://doi.org/10.1109/BADGERS.2014.11.

Amato, F., Cozzolino, G., Mazzeo, A., Mazzocca, N., 2017. Correlation of digital evidences in forensic investigation through semantic technologies. In: 2017 31st International Conference on Advanced Information Networking and Applications Workshops. WAINA), pp. 668–673. https://doi.org/10.1109/WAINA.2017.4.

Carrier, B., 2021. Digital forensics tool testing images. URL: http://dftt.sourceforge.net. (Accessed 12 October 2021).

Ceballos Delgado, A.A., Glisson, W.B., Grispos, G., Choo, K.-K.R., 2021. FADE: A forensic image generator for android device education. Wiley Interdisciplinary Reviews: Forensic Science e1432. https://doi.org/10.1002/wfs2.1432.

da/sec, 2021. Biometrics & Internet Security Research Group, hystck. URL. https://github.com/dasec/hystck. (Accessed 13 October 2021).

Dalins, J., Tyshetskiy, Y., Wilson, C., Carman, M.J., Boudry, D., 2018. Laying foundations for effective machine learning in law enforcement. majura – a labelling schema for child exploitation materials. Digit. Invest. 26, 40–54. https://doi.org/10.1016/j.diin.2018.05.004. URL.

DFRWS, 2021. Forensic challenges. URL: https://dfrws.org/forensic-challenges/. (Accessed 12 October 2021).

Du, X., 2020. Eviplant. URL: https://github.com/XiaoyuDu/eviplant. (Accessed 13 October 2021).

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, 2666-2817. https://www.sciencedirect.com/science/article/pii/S2666281721000317. Forensic disk image creation, Evidence planting, User emulation, Tool testing and validation, Forensic education.

Fragg, M., 2014. Forgeosi. URL. https://github.com/maxfragg/ForGeOSI. (Accessed 11 October 2021).

Garfinkel, S., 2007. Forensic Corpora: a Challenge for Forensic Research. Electronic Evidence Information Center, pp. 1–10.

Garfinkel, S., 2012. Lessons learned writing digital forensics tools and managing a 30tb digital evidence corpus, Digital Investigation. In: the Proceedings of the Twelfth Annual DFRWS Conference, 9, pp. S80–S89. https://doi.org/10.1016/j.diin.2012.05.002.

Garfinkel, S., Farrell, P., Roussev, V., Dinolt, G., 2009. Bringing science to digital forensics with standardized forensic corpora, Digital Investigation. In: the Proceedings of the Ninth Annual DFRWS Conference, 6, pp. S2–S11. https://doi.org/10.1016/j.diin.2009.06.016.

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.

Gonçalves, P., Attenberger, A., Baier, H., 2022. Actual data distribution in mobile devices and the need to obtain realistic mobile forensic corpora. In: Peterson, G., Shenoi, S. (Eds.), Advances in Digital Forensics XVIII. Springer International Publishing.

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://www.sciencedirect.com/science/article/pii/S1742287617301913.

Hadi, A., 2021. Digital forensic challenge images (datasets). URL: https://www.ashemery.com/dfir.html. (Accessed 12 October 2021).

Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D., 2018. Maldozer: Automatic framework for android malware detection using deep learning. Digit. Invest. 24, S48–S59. https://doi.org/10.1016/j.diin.2018.01.007.

Keighley, J., 2017. Forgen. URL: https://github.com/Jjk422/ForGen. (Accessed 11 October 2021).

Keighley, J., 2021. Forgen - the future of forensic image generation? https://www.heacademy.ac.uk/system/files/downloads/jason_keighley_-_forgen.pdf. (Accessed 11 October 2021).

Le, Q., Boydell, O., Mac Namee, B., Scanlon, M., 2018. Deep learning at the shallow end: malware classification for non-domain experts. Digital Investigation 26, S118–S126. https://doi.org/10.1016/j.diin.2018.04.024.

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, Berlin, Heidelberg, pp. 238–252.

Ngejane, C., Eloff, J., Sefara, T., Marivate, V., 2021. Digital forensics supported by machine learning for the detection of online sexual predatory chats. Forensic Sci. Int.: Digit. Invest. 36, 301109. https://doi.org/10.1016/j.fsidi.2021.301109. URL https://www.sciencedirect.com/science/article/pii/S2666281721000032.

NIST - National Institute of Standards and Technology, Computer Forensic Reference Data Sets (CFReDS), 2021. URL: https://www.cfreds.nist.gov. (Accessed 10 October 2021).

Park, J., 2018a. 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.

Park, J., 2018b. pyvmpop. URL: https://github.com/jungheum/pyvmpop. (Accessed 11 October 2021).

Qadir, S., Noor, B., 2021. Applications of machine learning in digital forensics. In: 2021 International Conference on Digital Futures and Transformative Technologies. ICoDT2), pp. 1–8. https://doi.org/10.1109/ICoDT252288.2021.9441543.

Scanlon, M., Du, X., Lillis, D., 2017. Eviplant: An efficient digital forensic challenge creation, manipulation and distribution solution. Digit. Invest. 20, S29–S36. DFRWS 2017 Europe. https://doi.org/10.1016/j.diin.2017.01.010.

Visti, H., 2015. Forge - forensic test image generator v2.1. URL. https://github.com/hannuvisti/forge. (Accessed 13 October 2021).

Visti, H., Tohill, S., Douglas, P., 2015. Automatic creation of computer forensic test images. In: Garain, U., Shafait, F. (Eds.), Computational Forensics. Springer International Publishing, Cham, pp. 163–175.

Woods, K., Lee, C.A., Garfinkel, S., Dittrich, D., Russell, A., Kearton, K., 2011. Creating realistic corpora for security and forensic education. In: Proceedings of ADFSL Conference on Digital Forensics. Security and Law, pp. 123–134.

Yannikos, Y., Steinebach, M., Graner, L., Winter, C., 2014. Data corpora for digital forensics education and research. In: Peterson, G., Shenoi, S. (Eds.), Advances in Digital Forensics X. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 309–325.