Scenario-based Data Set Generation for Use in Digital Forensics: A Case Study

Thomas Göbel [1], Harald Baier [1], and Dennis Wolf [2]

Abstract: Digital forensics is a rapidly growing and highly relevant field of cybersecurity. In case of an incident, the subsequent digital forensic investigation and analysis shall reveal the respective digital evidence. However, although electronic devices and their data play a central role in each crime investigation, data sets to train experts or to validate tools are sparse. While manual data set generation is a time-consuming, elaborate, and error-prone task, tool-based data synthesis is an excellent candidate for simplifying data generation and solving the data set gap problem. Synthetic data sets can be used, for example, to test and refine forensic tools and methods under controlled conditions. In addition, entirely new approaches can be explored. Several promising data synthesis frameworks for digital forensic data set creation have been published lately, the most recent of which is `ForTrace`, a freely available, community-driven data synthesis framework written in Python for generating digital forensic data sets. This paper shows how to apply `ForTrace` in a large-scale manner without human interaction. Our main goal is to show the usability of `ForTrace` and demonstrate its practicality and benefits for the digital forensic domain. We therefore provide a sample usage of `ForTrace` in two scenarios, namely a VeraCrypt and a malware use case, and present the definition of the corresponding configurations.

Keywords: Digital forensic data set, Digital corpora, Synthetic data, Ground truth data, Labeled data set, Data set generation, Data set creation, Data synthesis framework, `ForTrace`

1 Introduction

In the digital forensics domain, the number and heterogeneity of devices per examination case is steadily increasing. The demand for up-to-date data sets is high in order to train digital forensics practitioners and make faster progress in the development and validation of forensic tools [GBB17; Go22]. However, the manual creation of data sets is a complex, tedious, error-prone, and time-consuming task [Ga07; Ga09; Ga12; GBB23], which increases the need for solutions such as the automated creation of data sets [Bi24; Ce21; Du21; Gö20; Gö22; MPZ22; SDL17].

`ForTrace` [Gö22] is a recent data synthesis framework that can automatically generate forensic images with well-known forensic evidence. It is a community-driven Python 3 application capable of generating high-quality holistic data sets, i.e., the generated data set comprises associated dumps of a persistent storage device (e.g., a hard disc), the volatile

[1] University of the Bundeswehr Munich, RI CODE, Werner-Heisenberg-Weg 39, 85579 Neubiberg, Germany, thomas.goebel@unibw.de, https://orcid.org/0009-0001-5670-8150; harald.baier@unibw.de, https://orcid.org/0000-0002-9254-6398
[2] Central Office for Information Technology in the Security Sector (ZITiS), Zamdorfer Straße 88, 81677 Munich, Germany, Dennis.Wolf@ZITiS.bund.de, https://orcid.org/0009-0008-5929-6615

doi:10.18420/inf2024_25

memory (i.e., RAM), and a network capture. On the one hand, it is possible to access and control `ForTrace` via its Python interface in a programmatic manner. On the other hand, for the sake of usability, `ForTrace` may also be operated scenario-based in a non-programmatic way through YAML configuration files. This eases access to the framework for non-Python affine users since a human-readable text file defines the scenario and is then supplied to `ForTrace` to generate the respective data set.

In this paper, we focus on demonstrating the usability of `ForTrace` through a case study consisting of two distinct use cases or scenarios: (1) an encryption scenario based on the widespread VeraCrypt software and (2) a prominent ransomware scenario. We show how the actual user interactions of a scenario are configured and how the actual data set creation is put into practice with `ForTrace`, highlighting its practicality and benefits. For specific details on the implementation of `ForTrace`, we refer to the original works [GBT24; Gö22; LGB22; WGB24]. Besides the actual data set creation functionality, `ForTrace` provides an XML report that contains the ground truth of the executed scenario. This report provides key information for evaluating the generated data set. While a `ForTrace` scenario is running, next to changes in the file system, many typical Windows artefacts are created, such as in the Windows EventLog, Registry, Jump Lists, Thumbcaches, etc. [SS16]. To store the actual ground truth data of each synthesis run, these artefacts are included in the aforementioned XML report. In the following forensic evaluation, we will demonstrate the key findings in a comprehensible way.

The rest of the paper is organised as follows: we present related work in Section 2 and briefly introduce the `ForTrace` architecture in Section 3. After the description of our two sample scenarios in Section 4, we provide insights into the actual `ForTrace` generation process in Section 5. We then evaluate the generated traces in Section 6 and conclude our paper in Section 7.

2 Related Work

Besides the `ForTrace` framework [Gö22], several other data synthesis approaches have been published in the past. The best known of these are discussed in the following.

`EviPlant` [SDL17] uses a base image as a starting point. The challenges or traces can then be downloaded in the form of *evidence packages*. This has the advantage that large files do not have to be sent multiple times, which is particularly interesting for teaching purposes. The evidence package only needs to be injected into the base image and the investigation can be started. Unfortunately, the original GitHub repository has gone offline in the meantime.

`hystck` [Gö20] is a Python-based framework that can create network and hard disc traces. The creation can be automated using Python scripts or YAML configuration files. Automated synthesis makes it possible to create a wide variety of traces within a VM with little effort, which can be distributed efficiently by defining the contents as changes from a template image. However, the project only supports Windows-based systems and is no longer being developed further, as it is now being continued as `ForTrace`.

`TraceGen` [Du21] is another Python-based framework to automatically generate forensic images. It is based on an emulator that translates high-level actions and simulates user behaviour by performing (sub)-operations inside a VM, e.g., using an Internet browser or modifying files on a hard disc. All changes are stored on a disc image and simultaneously logged in a separate file that serves as the ground truth. The framework currently only supports the emulation of Windows-based systems and its source code has unfortunately not been published.

In addition, the community provides further data set generators, e.g., for the synthesis of mobile device images or network traffic: `FADE` [Ce21] is a proof of concept (PoC) to inject static traces in an Android-based emulator by using the Android Debug Bridge (ADB). The authors directly modify files and database entries in an Android VM to mimic user-created content. `AutoPoD-Mobile` [MPZ22] is another, superior injection tool in the context of mobile devices, i.e., it is not able to generate images from a normal desktop system. It is a PoC framework for generating Android datasets using ADB, APIs from selected apps, and a Google account. Unfortunately, as mentioned in the paper, the tool only works on a few physical and meanwhile outdated Android devices (e.g., Samsung A50, Huawei Mate 20 Lite). `ForTT-Gen` [Bi24] is a tool that can generate and replicate network traffic using a hybrid model that combines the replication of real data and the generation of synthetic data through statistical techniques.

To our knowledge, the latest and most powerful framework capable of generating images from Desktop PCs is `ForTrace` [Gö22], which builds upon the `hystck` framework [Gö20]. The framework supports automated image generation by simulating human-computer interactions. According to the authors, this approach is intended to create more realistic scenarios, as it not only inserts traces into existing images (as `FADE` and `AutoPoD-Mobile` do), but fully interacts with a running operating system in real-time. Along with the generated associated disc image, the memory dump and the network capture, `ForTrace` also provides a log of the synthesised traces, which serves as ground truth.

3 Fundamentals of the ForTrace Data Synthesis Framework

In this section, we briefly introduce the digital forensic data set generation framework `ForTrace`. It is the successor of the `hystck` [Gö20] data synthesis framework, which is only available in Python 2 and no longer maintained.

`ForTrace` follows a client-server architecture depicted in Fig. 1 [Gö22]. The *Framework Master* is located on the host machine and communicates with the client-side *Guest Component* via an unmonitored private network. Inside the Guest Component resides the *Interaction Manager* that exchanges information about the current state of the scenario

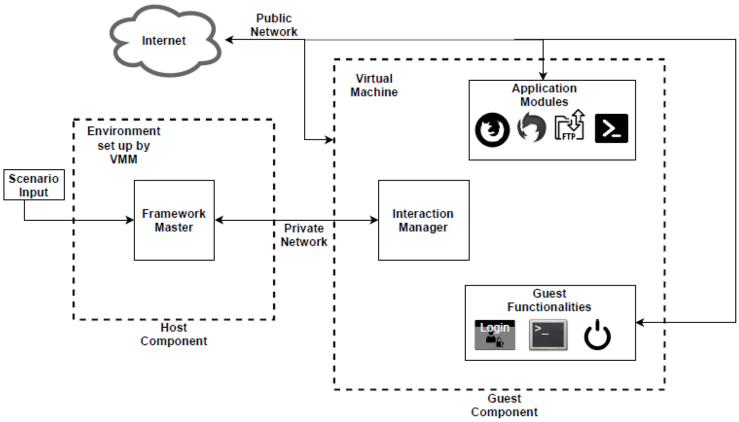

Fig. 1: Client-server architecture of the `ForTrace` data synthesis framework [Gö22]

execution with the Framework Master. It interacts with the client-side *Application Modules* – used to control individual applications via their exposed APIs – and *Guest Functionalities*, which group general functionalities like the control of the power-state together. Each guest virtual machine (VM) has access to a second network interface (Public Network) to connect to the local network or the Internet. This interface is monitored by `tcpdump` to create the pcap file of a scenario.

A key feature of `ForTrace` is its ability to create random user interactions, for example, to simulate browsing behaviour during the data synthesis process, providing realistic wear-and-depth in the resulting data sets. We refer to [GBT24; Gö22; LGB22; WGB24] for more details about `ForTrace`.

4 Example Scenarios for Data Synthesis with ForTrace

This section introduces the two scenarios used in this paper to show the usability and utility of `ForTrace`. Our scenarios are available within a prepared master VM, which we make available to the community via the following link: `https://cloud.digfor.code.unibw-muenchen.de/s/IWDF24`. The master VM features the latest version of `ForTrace` and two nested VMs that provide the corresponding Windows templates to drive both scenarios. Both VMs contain a fully functional guest installation of `ForTrace`. Tab. 1 shows the specifications of the downloadable VM. We stress the resource availability on your system and the nested virtualisation to execute a scenario without performance issues.

	Host VM	**Windows Guest VM1**	**Windows Guest VM2**	**Service VM**
OS	Ubuntu 22.04	Windows 10 22H2	Windows 10 22H2	Debian 12
Storage	160 GiB	40 GiB	40 GiB	20 GiB
RAM	24 GiB	8 GiB	8 GiB	4 GiB
CPU	10 vCPU cores	2 vCPU cores	2 vCPU cores	2 vCPU cores
Hypervisor	QEMU 7.2	–	–	–

Tab. 1: `ForTrace` setup specifications, including a master VM with two nested VMs and a service VM

4.1 Scenario 1 – The Supposedly Secure VeraCrypt Container

In the first scenario, the Windows Guest VM is used. A user first downloads a sensitive file from the Internet and then moves it into a previously created VeraCrypt container. To enhance the scenario, background noise is generated by the random user interaction feature of `ForTrace` as described in Section 3. The subsequent scenario step involves the Windows Notepad application, where the user types in the password for the VeraCrypt container, prints it out on paper, and then discards the file. From a security perspective, this leaves the possibility that the discarded password file could be recovered from the user's file system before it is overwritten, compromising the security of the VeraCrypt container. In the final flow of scenario 1, the simulated user makes use of the tool `sdelete`[3] to securely wipe the file containing the password for the VeraCrypt container after it was printed. Since `ForTrace` can acquire RAM dumps at any time in a scenario, this opens up the possibility of analysing the volatile memory dump to extract the password from the running Notepad instance.

4.2 Scenario 2 – The Supposedly Friendly E-Mail From a Colleague

The second scenario involves a Windows VM and a so-called service VM. It involves malware that is received via e-mail on the Windows VM. In addition, the framework's service VM is used to provide a company SMB, web, and an FTP server. For the e-mail server, both external e-mail accounts or the framework's service VM can be used. The attacker, Mallory, uses the webserver to upload his malware and shares the link by sending a phishing e-mail to his victim, Alice. Since both know each other (at least from the victim's perspective), Alice downloads the supposedly benign file via her webbrowser. Unknowingly, she then executes a ransomware that encrypts her computer and demands a ransom to recover her data. This scenario demonstrates the ability of `ForTrace` to simulate more complex scenarios with multiple participants, which other data synthesis frameworks in the digital forensics domain do not cover to this extent. The user of the `ForTrace` framework is able to configure several important aspects of this scenario through the YAML configuration files. For example, the user can customise the phishing e-mail, the webserver from which the file is downloaded, or the behaviour of the ransomware itself once it is executed on the victim's computer (e.g., whether and how data is exfiltrated before it is encrypted).

3 https://learn.microsoft.com/en-us/sysinternals/downloads/sdelete (last accessed on 2024-06-16)

5 The ForTrace Data Synthesis Process

This section shows how to produce a YAML configuration file and the subsequent data set based on the defined scenarios. More precisely, for both of the two scenarios from Section 4 we provide the respective YAML configuration file and then run through the `ForTrace` generation process. We also show how to generate the corresponding report, which serves as ground truth.

Fig. 2: Flowchart of the `ForTrace` Generator component

Fig. 2 shows the steps performed by the `ForTrace` Generator after initiating a scenario with a YAML configuration file. First, the Generator starts the guest VM and reads the configuration file. Then the Generator loads all default collections (which are shipped with `ForTrace`) providing URLs, files, and other information. All applications used in the scenario are opened to be ready to receive input. The set of actions is generated and then randomised using the seed supplied from the configuration. The seed allows for quick changes in the position of actions in a scenario without introducing manual changes to the code or the configuration. The Generator executes all actions, and after the last one concludes, the guest VM is shut down, and the pcap file is saved. An additional memory dump can be requested anytime during the scenario's execution. Once the synthesis process is complete, the image with all generated artefacts is stored on the host machine.

List. 1 shows a sample YAML configuration file of the first scenario with the addition of downloading a specific file from the Internet and performing random browsing operations. The first three entries `name`, `description`, and `author` serve as meta-information and are therefore ignored by the `ForTrace` Generator. The purpose of the `seed` field is to introduce randomisation which was discussed before. The `collections` section aims at defining any type of collection that should be used in the scenario, e.g., a single text file containing links to websites that should be visited (as in our case through the `friendly_urls.txt` file), or a directory with files that should be used throughout the scenario. The kind of data in a collection is dependent on its type. The entries within the `applications` section provide any settings for applications executed during the scenario. In the context of our sample configuration file, some basic VeraCrypt parameters for the container's creation and subsequent mounting operation are defined. Additional applications and services can also be configured here to create a larger scenario and thus more artefacts in the resulting image. Finally, the `needles` section contains further scenario-relevant data, in our case the 'sensitive' file that is downloaded from the given URL and then hidden in the VeraCrypt container.

```yaml
name: ForTrace VeraCrypt Sample Scenario
description: An example of creation and utilisation of a VeraCrypt container
author: Mr. X
seed: 1234
collections:
  c-http-0:
    type: http
    urls: ../../generator/friendly_urls.txt
applications:
  veracrypt:
    cont_path: 'C:\Users\fortrace\Desktop\container.vc' # location of the container to be
↪   created
    cont_size: 100 # size of the container in MiB
    cont_pass: "password" # password of the container
    mount_point: "Z" # mount point of the container
needles:
  n-http-0:
    application: http
    url: 'https://forensik.hs-mittweida.de/assets/images/brand/Fosil.jpg' #enter payload to be
↪   downloaded here
    amount: 1
```


Listing 1: Sample YAML configuration file for the first scenario

The YAML configuration file in List. 2 illustrates the configuration of the ransomware in scenario 2 in order to download the ransomware from the given IP address and port it to the local Downloads folder and store it as *scanme.exe*. For example, the AES encryption/decryption key is configured to be sent to the external party ('send_key_to_service_vm: True'). The option *exfiltrate_files:* defines whether the data is exfiltrated before it is encrypted. The option *remove:* defines whether the downloaded ransomware file is deleted or not. The option *wipe_file:* can be used to select whether the files should be securely deleted via `sdelete.exe` or via the normal deletion of the OS. In addition, in List. 2 one can see two options for how `ForTrace` can generate e-mails. Option one is to define an XML file (here: *email_hay.xml*), which is primarily used to fill the mbox files of Thunderbird (e.g., INBOX or Sent MBOX files). With option two, the e-mails are explicitly specified in the YAML file, which is primarily used to define malicious e-mails in the *needles* section.

Of course, further options can be defined in a scenario's YAML file, but using this sample, it should become apparent how the human-readable configuration file of `ForTrace` operates. Each application in `ForTrace` has its own selectable options which are documented in the `ForTrace` code API. Depending on the options selected, this function makes it possible to deploy more complex environments and drive more realistic simulations – especially since it does not require writing Python code.

The YAML configuration file snippet in List. 3 illustrates the configuration and use of additional required services provided by the `ForTrace` service VM, such as the e-mail and SMB services used in scenario 2 to access the e-mail and SMB services. The setup of the

```yaml
name: ForTrace Ransomware Sample Scenario
description: An example of sending, downloading and executing a ransomware, including data
↪ exfiltration
author: Mr. X
seed: 42
collections:
  c-ransomware-0:
    type: ransomware
    commands: ransomware-collection.txt
    email: email_hay.xml             # fill up MBOX file with arbitrary e-mails
applications:
  ransomware-0:  # Individual configuration of the ransomware behaviour
    type: ransomware
    service-vm: 192.168.103.8        # webserver IP to download ransomware
    service-port: 8080               # webserver port to dowload ransomware
    path: C:\Users\fortrace\Downloads # folder where ransomware is stored
    downloaded_file_name: scanme.exe  # name of the ransomware
    send_key_to_service_vm: True
    #http://evil-website.com/upload   # send_key_to_specific_url
    send_key_to_ftp_server: True      # decryption key is send to FTP server
    entry_point: C:\Users\fortrace\Documents  # folder that is recursively encrypted
    wipe_file: False                 # whether data is securely deleted
    force_powershell_log: True
    exfiltrate_files: True           # exfiltrate data before encryption
    remove: False                    # whether ransomware is deleted afterwards
    #key: test                       # Select your own encryption key
hay:
  h-mail-0:  # Benign e-mail
    application: mail-0
    sender: bob_fortrace@web.de
    recipient: alice_fortrace@web.de
    subject: Welcome to our company
    message: Dear Alice,\n\nwe welcome you to our company. We wish you a good start to your
↪ first week.\n\nBest regards, ForTrace.
    amount: 1
needles:
  n-mail-0:  # Specific malicious phishing e-mail
    application: mail-0
    sender: mallory_fortrace@web.de
    recipient: alice_fortrace@web.de
    subject: Instructions for new employees
    content: Dear Sir or Madam,\n\nplease download and run our in-house security scanner. You
↪ can find it at http:\\192.168.103.8\scanme.exe.\n\nBest regards\n\nMallory
    amount: 1
  n-ransomware-0:
    application: ransomware-0
    collection: c-ransomware-0
```


Listing 2: Sample YAML configuration file for the second scenario

FTP server that is used to exfiltrate data prior to the encryption is part of the ransomware application shown in List. 2.

```yaml
applications:
  mail-0:
    type: mail
    imap_hostname: imap.web.de
    smtp_hostname: smtp.web.de
    email: alice_fortrace@web.de
    password: Vo@iLmx48Qv8m%y
    username: fortrace
    full_name: Alice Fortrace
    socket_type: 3
    socket_type_smtp: 2
    auth_method_smtp: 3
  smb-0:
    type: smb
    username: service
    password: fortrace
    destination: \\192.168.103.8\sambashare
```


Listing 3: YAML configuration file to provide the required services on the service VM for the second scenario

6 Evaluation of the Data Synthesis Process and the Generated Images

After the actual data synthesis process, the most crucial part is evaluating the data sets using common digital forensics software such as Autopsy and Volatility. We compare the actual traces within the data set with the corresponding ground truth from the `ForTrace` report. Sample questions of doubt, which may be easily answered using the `ForTrace` report, are: When did what happen? Who triggered action X? Which files were modified?

6.1 Files from Scenario 1

Scenario 1, involving the VeraCrypt container, was configured to yield a RAM dump and a disc image since the network dump would not contain any interesting information. The RAM dump was created when the PowerShell session was still open. This enables to recover the used VeraCrypt key from the RAM dump[4].

6.2 Files from Scenario 2

To prepare the disc image for the forensic analysis with Autopsy, we first need to merge the differential *qcow2* image (file format that is used per default by KVM) of the clone of our initial Windows template VM image. Furthermore, we convert the image to a raw image file

[4] Please note: The comprehensive evaluation for scenario 1 is skipped here due to the page limit. However, the provided `ForTrace` VM contains the corresponding scenario configuration files and the generated images.

(`qemu-img convert -f qcow2 -O raw guest-xxx-0.qcow2 guest-xxx.raw`) to import it into a new Autopsy case.

Network Traffic: In Wireshark, we do not only see the DNS request for the MX server of our web.de mailbox, but also IMAPS/POP3S traffic. Since we configured the scenario using the *send_key* options to send the symmetric encryption key of the ransomware (Base64 encoded) via HTTP to the service VM (running on the local IP address 192.168.103.8), we see it in the network capture depicted in Fig. 3 as a GET request which the associated *DownloadString(Uri)* function of the ransomware is using[5]. In addition, it can be seen that the ransomware file is downloaded and stored as a file called *scanme.exe* from the same IP address (as configured in the YAML file).

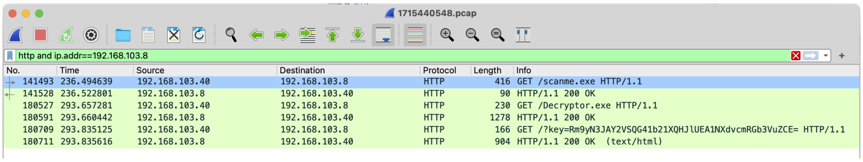

Fig. 3: Traces of the ransomware download in the network traffic capture

Since the *exfiltrate_files* option was specified in the YAML configuration, the files in the Documents folder get exfiltrated before encryption. The data exfiltration (including the encryption/decryption key *enc.key*) can be seen in the pcap file, since the remote peer uses FTP without encryption, as shown in Fig. 4.

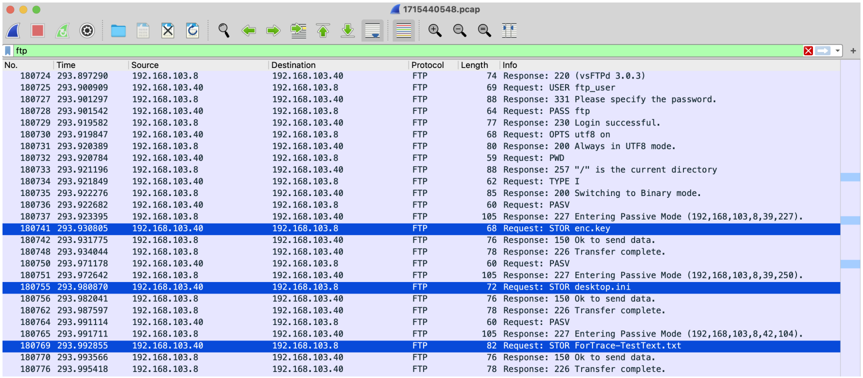

Fig. 4: Traces of data exfiltration to an external FTP server

The login credentials for Mallory's external FTP server can also be seen in the network capture. We can log in and see the intact files that were exfiltrated before the encryption started, as shown in Fig. 5.

[5] https://learn.microsoft.com/en-us/dotnet/api/system.net.webclient.downloadstring?view=net-8.0 (last accessed on 2024-05-12).

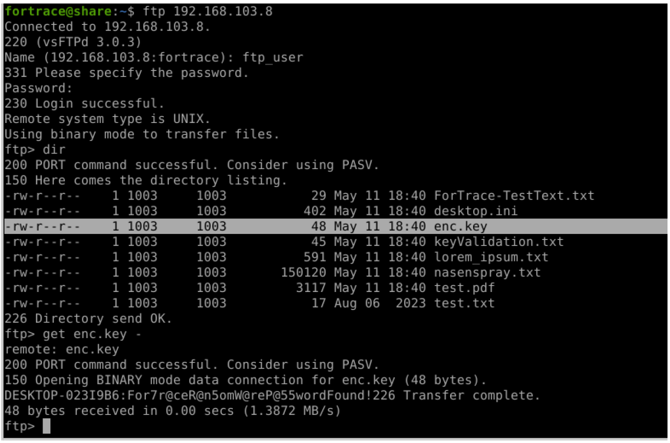

Fig. 5: Files that were exfiltrated to the external FTP server, including the ransomware key

Hard Disc Image: Using Autopsy, we now analyse the hard disc image. As relevant artefacts, we find the ransomware sample including its Zone.Identifier embedded as Alternate Data Stream (due to option *remove: False*) in the *Downloads* folder, as depicted in Fig. 6. On the Desktop, we find the *Decryptor.exe* that is also downloaded form the external webserver. This binary is used to decrypt/recover original files that are encrypted due to the ransomware.

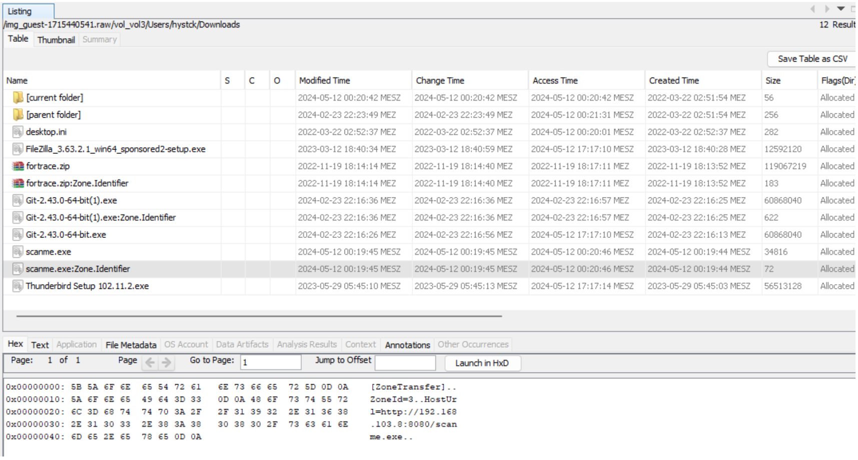

Fig. 6: Downloaded ransomware binary

Due to the defined option *entry_point*, Autopsy indicates that all files within the *Documents* folder are encrypted, while the original files were deleted (see allocation status), as can be seen in Fig. 7. Since we have only selected to remove (not wipe) the files, in this case, as long as no other data is stored in the unallocated byte offsets, it would be possible to restore the original files with Autopsy without having to carve the ransomware decryption key out of memory or analyse the network capture, etc. So we see that several different scenarios can easily be created with `ForTrace` depending on the specified configuration.

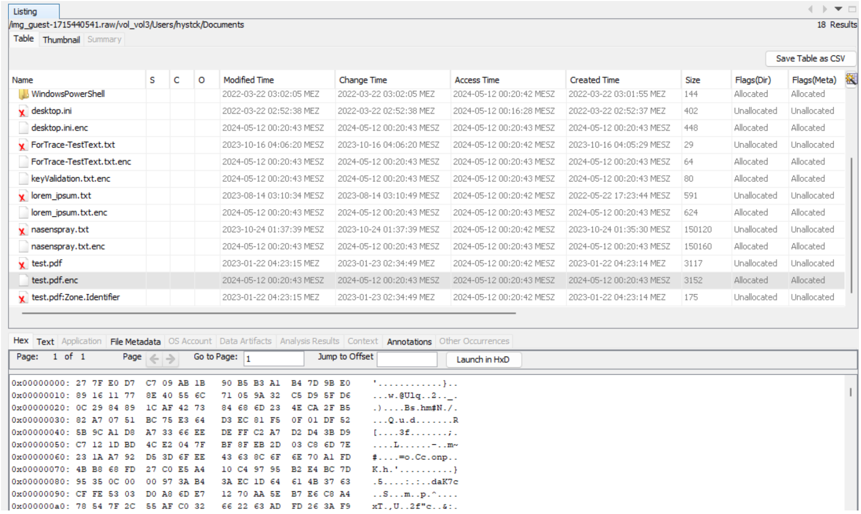

Fig. 7: Documents folder that is encrypted due to the ransomware

Many other relevant traces can be recovered with Autopsy, such as the PowerShell's logfile (*../Roaming/Microsoft/Windows/PowerShell/PSReadLine/ConsoleHost_History.txt*), the generated e-mails (an excerpt of the recovered INBOX file can be seen in Fig. 8), the configured SMB share (e.g., as Shell Bags), the web download of the ransomware *scanme.exe*, the Windows registry autostart entry created for *Decryptor.exe* (as shown in List. 4) that demands the ransom and gives the opportunity to decrypt files after rebooting the system.

```
user_run v.20140115
(NTUSER.DAT) [Autostart] Get autostart key contents from NTUSER.DAT hive

Software\Microsoft\Windows\CurrentVersion\Run
LastWrite Time Sat May 11 22:20:41 2024 (UTC)
  Decryptor.exe: C:\Users\hystck\Desktop\Decryptor.exe --C:\users\hystck\Documents
  [...]
```

Listing 4: *Run* registry key contents of the user's NTUSER.DAT hive

Memory Dump: As shown in Fig. 9, we search for running processes using Volatility 3 and its *windows.pslist* plugin and actually find the running ransomware *scanme.exe* with the PID 4976, as `ForTrace` dumps the memory during the execution of the ransomware. It can also be seen that the process's parent process is the *powershell.exe* process with PID 4636. The memory dump scan reveals the entire sequence of processes that led to

INBOX

```
From MAILER-DAEMON Sun May 12 00:02:14 2024
Content-Type: multipart/mixed; boundary="===============0296097543230650768=="
MIME-Version: 1.0
From: <bob_fortrace@web.de>
To: <alice_fortrace@web.de.de>
Subject: Welcome to our company

--===============0296097543230650768==
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit

Dear Alice,

we welcome you to our company.We wish you a good start to your fist week.
Best regards, ForTrace.
--===============0296097543230650768==--

From MAILER-DAEMON Sun May 12 00:12:14 2024
Content-Type: multipart/mixed; boundary="===============0936777291311604120=="
MIME-Version: 1.0
From: <mallory_fortrace@web.de>
To: <alice_fortrace@web.de.de>
Subject: Instructions for new employees

--===============0936777291311604120==
Content-Type: text/plain; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit

Dear Sir or Madam,

please download and run our in-house security scanner. You can find it at http:\\192.168.103.8\scanme.exe.

Best regards

Mallory
--===============0936777291311604120==--
```

Fig. 8: Excerpt from the Thunderbird INBOX file containing the specified e-mails

the execution of the ransomware. This encompasses not only operating system processes but also processes linked to the framework. The complete process list from the process scan is: `winlogon.exe > userinit.exe > explorer.exe > cmd.exe > python.exe > powershell.exe > scanme.exe` by comparing the process ID (PID) and its parent process ID (PPID).

6184	7564	firefox.exe	0×9789a6a90080	1	-	1	False	2024-05-11 22:20:34.000000	N/A	Disabled
4636	7308	powershell.exe	0×9789a80d6080	17	-	1	False	2024-05-11 22:20:36.000000	N/A	Disabled
4976	4636	scanme.exe	0×9789a6a8b0c0	14	-	1	False	2024-05-11 22:20:37.000000	N/A	Disabled
8108	676	WmiApSrv.exe	0×9789a71e3080	7	-	0	False	2024-05-11 22:20:38.000000	N/A	Disabled
2036	804	dllhost.exe	0×9789a898e080	8	-	1	False	2024-05-11 22:21:03.000000	N/A	Disabled

Fig. 9: Volatility's *windows.pslist* plugin to scan process list

We can then create a memory dump using Volatility's *windows.memmap* plugin (cf. with List. 5 to not only dump the executable (such as the *windows.dumpfiles* plugin would do) but also all the process's contents of virtual address space present in memory. With `strings` applied to the dumped memory process, the actual ransomware decryption key can be found multiple times.

Of course, there are also other methods for analysing the individual scenarios. This is just to give an impression of how `ForTrace` reflects relevant artefacts depending on how the data synthesis scenario is configured. In the YAML configuration file, for example, we can also specify that we use the ransomware with a hardcoded key or, if desired, define an arbitrary string that serves as the ransomware key. Depending on this, it would either be possible to

```
$ $./vol.py -f  Ransomware-dump.DMP -o .  windows.memmap --dump --pid 4976
$ strings -n 32 pid.4976.dmp
[...]
For7r@ceR@n5omW@reP@55wordFound!
[...]
```

Listing 5: Volatility's *windows.memmap* plugin to dump the memory of the malicious process

extract the key using a reverse engineering tool such as Ghidra (if it is hardcoded in the binary; cf. Fig. 10) or not (if the key was explicitly specified by the user in the YAML file).

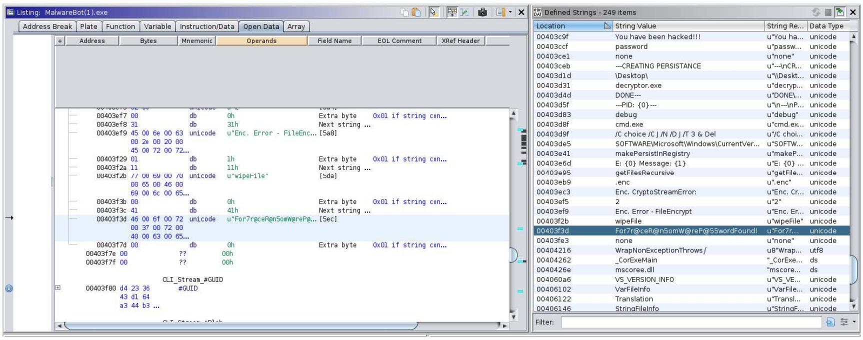

Fig. 10: Recovery of hardcoded ransomware key in binary file using Ghidra

7 Conclusion and Future Work

In this paper, we showed the capabilities of the `ForTrace` data synthesis framework in two scenarios. Scenario 1 introduced the basic functionality of `ForTrace`, a simple YAML configuration file, and the framework's Python interface. In the second scenario, we demonstrated that `ForTrace` is also capable of simulating more complex scenarios involving multiple participants and the provision of accompanying infrastructure, such as a ransomware setup and e-mail or file sharing servers. In the event that the YAML-based configuration is not desired (e.g., if the YAML configuration is not yet supported), all functions of `ForTrace` can also be used with Python scripts, as `ForTrace` provides its open source code API. In Sect. 6, the generated data sets were examined and the findings were evaluated using common forensic software such as Autopsy and Volatility. The suspected traces, depending on the selected scenario, were found.

As future work, further `ForTrace` functions must be validated, as this paper could not cover all functions provided due to a limited number of pages. This includes in particular the functions offered to generate more random data and a more detailed evaluation of the `ForTrace` reporting component. Further evaluation tasks include a more detailed look at the `ForTrace` GUI and the use of the `ForTrace` code API to script arbitrary forensic cases independent of the Generator's functionality.

References

[Bi24] Bistene, J. V. et al.: ForTT-Gen: Network Traffic Generator for Malware Forensics Analysis Training. In: 2024 12th International Symposium on Digital Forensics and Security (ISDFS). Pp. 1–6, 2024, DOI: https://doi.org/10.1109/ISDFS60797.2024.10527345, visited on: 07/30/2024.

[Ce21] Ceballos Delgado, A. A. et al.: FADE: A forensic image generator for android device education. WIREs Forensic Science 4 (2), e1432, 2021, DOI: https://doi.org/10.1002/wfs2.1432, URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wfs2.1432, visited on: 07/30/2024.

[Du21] Du, X. et al.: TraceGen: User activity emulation for digital forensic test image generation. Forensic Science International: Digital Investigation 38, p. 301133, 2021, ISSN: 2666-2817, DOI: https://doi.org/10.1016/j.fsidi.2021.301133, URL: https://www.sciencedirect.com/science/article/pii/S2666281721000317, visited on: 07/30/2024.

[Ga07] Garfinkel, S.: Forensic corpora: a challenge for forensic research. Electronic Evidence Information Center, pp. 1–10, 2007, URL: http://simson.net/ref/2007/Forensic_Corpora.pdf, visited on: 07/30/2024.

[Ga09] Garfinkel, S. et al.: Bringing science to digital forensics with standardized forensic corpora. Digital Investigation 6, The Proceedings of the Ninth Annual DFRWS Conference, pp. 2–11, 2009, ISSN: 1742-2876, DOI: https://doi.org/10.1016/j.diin.2009.06.016, visited on: 07/30/2024.

[Ga12] Garfinkel, S.: Lessons learned writing digital forensics tools and managing a 30TB digital evidence corpus. Digital Investigation 9, The Proceedings of the Twelfth Annual DFRWS Conference, S80–S89, 2012, ISSN: 1742-2876, DOI: https://doi.org/10.1016/j.diin.2012.05.002, URL: https://www.sciencedirect.com/science/article/pii/S1742287612000278, visited on: 07/30/2024.

[GBB17] Grajeda, C.; Breitinger, F.; Baggili, I.: Availability of datasets for digital forensics – And what is missing. Digital Investigation 22, S94–S105, 2017, ISSN: 1742-2876, DOI: https://doi.org/10.1016/j.diin.2017.06.004, URL: https://www.sciencedirect.com/science/article/pii/S1742287617301913, visited on: 07/30/2024.

[GBB23] Göbel, T.; Baier, H.; Breitinger, F.: Data for Digital Forensics: Why a Discussion on "How Realistic is Synthetic Data" is Dispensable. Digital Threats 4 (3), 2023, DOI: https://doi.org/10.1145/3609863, URL: https://dl.acm.org/doi/full/10.1145/3609863, visited on: 07/30/2024.

[GBT24] Göbel, T.; Baier, H.; Türr, J.: Generating Usable and Assessable Datasets Containing Anti-Forensic Traces at the Filesystem Level. In (Kurkowski, E.; Shenoi, S., eds.): Advances in Digital Forensics XX. Springer International Publishing, Cham, 2024, URL: https://link.springer.com/book/9783031710247, visited on: 07/30/2024.

[Gö20] Göbel, T. et al.: 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, 2020, ISBN: 978-3-030-56223-6, DOI: https://doi.org/10.1007/978-3-030-56223-6_5, visited on: 07/30/2024.

[Go22] Gonçalves, P. et al.: Revisiting the dataset gap problem – On availability, assessment and perspective of mobile forensic corpora. Forensic Science International: Digital Investigation 43, p. 301439, 2022, ISSN: 2666-2817, DOI: https://doi.org/10.1016/j.fsidi.2022.301439, URL: https://www.sciencedirect.com/science/article/pii/S2666281722001202, visited on: 07/30/2024.

[Gö22] Göbel, T. et al.: ForTrace - A holistic forensic data set synthesis framework. Forensic Science International: Digital Investigation 40, Selected Papers of the Ninth Annual DFRWS Europe Conference, p. 301344, 2022, ISSN: 2666-2817, DOI: https://doi.org/10.1016/j.fsidi.2022.301344, URL: https://www.sciencedirect.com/science/article/pii/S2666281722000130, visited on: 07/30/2024.

[LGB22] Lukner, M.; Göbel, T.; Baier, H.: Realistic and Configurable Synthesis of Malware Traces in Windows Systems. In (Peterson, G.; Shenoi, S., eds.): Advances in Digital Forensics XVIII. Springer International Publishing, Cham, pp. 21–44, 2022, ISBN: 978-3-031-10078-9, DOI: https://doi.org/10.1007/978-3-031-10078-9_2, visited on: 07/30/2024.

[MPZ22] Michel, M.; Pawlaszczyk, D.; Zimmermann, R.: AutoPoD-Mobile—Semi-Automated Data Population Using Case-like Scenarios for Training and Validation in Mobile Forensics. Forensic Sciences 2 (2), pp. 302–320, 2022, DOI: https://doi.org/10.3390/forensicsci2020023, visited on: 07/30/2024.

[SDL17] Scanlon, M.; Du, X.; Lillis, D.: EviPlant: An efficient digital forensic challenge creation, manipulation and distribution solution. Digital Investigation 20, DFRWS 2017 Europe, S29–S36, 2017, ISSN: 1742-2876, DOI: https://doi.org/10.1016/j.diin.2017.01.010, URL: https://www.sciencedirect.com/science/article/pii/S1742287617300397, visited on: 07/30/2024.

[SS16] Shaaban, A.; Sapronov, K.: Practical Windows Forensics: Leverage the power of digital forensics for Windows systems. https://www.packtpub.com/en-us/product/practical-windows-forensics-9781783554096, visited on: 07/30/2024, Packt Publishing, 2016.

[WGB24] Wolf, D.; Göbel, T.; Baier, H.: Hypervisor-based data synthesis: On its potential to tackle the curse of client-side agent remnants in forensic image generation. Forensic Science International: Digital Investigation 48, DFRWS EU 2024 - Selected Papers from the 11th Annual Digital Forensics Research Conference Europe, p. 301690, 2024, ISSN: 2666-2817, DOI: https://doi.org/10.1016/j.fsidi.2023.301690, URL: https://www.sciencedirect.com/science/article/pii/S2666281723002093, visited on: 07/30/2024.