A Novel Approach for Generating Synthetic Datasets for Digital Forensics

Thomas Göbel, Thomas Schäfer, Julien Hachenberger, Jan Türr, Harald Baier

Chapter 5

A NOVEL APPROACH FOR GENERATING SYNTHETIC DATASETS FOR DIGITAL FORENSICS

Thomas Göbel, Thomas Schäfer, Julien Hachenberger, Jan Türr and Harald Baier

Abstract Increases in the quantity and complexity of digital evidence necessitate the development and application of advanced, accurate and efficient digital forensic tools. Digital forensic tool testing helps assure the veracity of digital evidence, but it requires appropriate validation datasets. The datasets are crucial to evaluating reproducibility and improving the state of the art. Datasets can be real-world or synthetic. While real-world datasets have the advantage of relevance, the interpretation of results can be difficult because reliable ground truth may not exist. In contrast, ground truth is easily established for synthetic datasets.

This chapter presents the `hystck` framework for generating synthetic datasets with ground truth. The framework supports the automated generation of synthetic network traffic and operating system and application artifacts by simulating human-computer interactions. The generated data can be indistinguishable from data generated by normal human-computer interactions. The modular structure of the framework enhances the ability to incorporate extensions that simulate new applications and generate new types of network traffic.

Keywords: Synthetic dataset generation, network traffic, operating system data

1. Introduction

Advanced, accurate and efficient digital forensic tools are vital to processing the large volumes of complex digital evidence encountered in digital forensic investigations. In order to be admissible in court, open source digital forensic tools must meet four criteria: (i) tools, techniques and procedures are thoroughly tested to assess the occurrences of false negatives and false positives; (ii) results are verifiable and falsifiable in

order to specify possible error rates; (iii) new procedures are discussed in the scientific community and subjected to objective peer reviews; and (iv) new procedures are accepted by the digital forensic community [4].

In order to evaluate a suitable digital forensic tool (e.g., for network traffic forensics), appropriate forensic testing and evaluation datasets comprising correctly-labeled data that are similar to real-world data are required. Tool testing, as suggested by Carrier [4], can only be performed if suitable datasets are available to assess the accuracy and generalizability of the results. NIST's Computer Forensic Tool Testing (CFTT) Program [13] is responsible for developing test methods for digital forensic tools and producing appropriate test data. The main criteria for digital evidence admissibility in court are that appropriate tests should be repeatable and reproducible. These criteria cannot be achieved without high-fidelity testing and evaluation datasets for forensic tools, techniques and procedures.

A number of datasets have been proposed for testing and evaluating digital forensic tools. Ring et al. [15] identify a broad spectrum of network-based datasets that have been released from 1998 through 2019. However, datasets quickly become outdated. They are often too academic, too specific, too synthetic (and thus too unrealistic) and/or too anonymized. Additionally, there is a lack of good real-world datasets.

Grajeda et al. [7] stress the relevance of up-to-date datasets and the importance of sharing them in the digital forensics community. Their research reveals that 198 of the 351 analyzed datasets (56.4%) were experimentally generated, where researchers considered *ad hoc* scenarios to create data for their experiments. Only 129 (36.7%) were real-world datasets and 16 (4.6%) were computer-generated datasets (e.g., using algorithms, bots or simulators). Grajeda and colleagues also noted that 45.6% (160 out of 351) of the datasets were newly created, but only 3.8% of them (6 of 160) were released to the public due to concerns about releasing digital forensic data, especially real-world data. According to some researchers [1, 7], barriers to publishing datasets include data protection laws, privacy and intellectual property concerns, lack of resources and/or capabilities, and lack of understanding of the importance of sharing.

Aside from the availability and standardization of datasets, the process of generating data is of crucial importance. The generated datasets must be reliable and realistic. Additionally, the more extensive the datasets, the better they are for education and training, and for the application of machine learning algorithms.

Meanwhile, the heterogeneity and complexity of modern infrastructures require the use of a variety of forensic acquisition and analysis

methods. Sophisticated analyses involve attack attribution based on recurring attack patterns and the correlation of diverse information from multiple data sources. Therefore, from a data synthesis point of view, it is not enough to merely generate network traffic in PCAP files, but also digital evidence from other sources (e.g., server-side information in Apache log files, client-side information in syslog/event logs and even memory dumps).

This chapter describes the `hystck` framework for generating synthetic evaluation corpora for digital forensics using a novel approach that allows the modeling of a complete infrastructure with a realistic network environment. Although the main goal is to generate benign and malicious network traffic, the proposed framework offers a more holistic data synthesis approach compared with existing traffic generators. Specifically, it simulates entire operating system sequences and applications via synthetic human-computer interactions, generating a ground truth that is realistic and comprehensive to the extent possible. The framework also enables researchers to generate network traffic as well as relevant digital evidence and artifacts in operating systems, such as data that is typically stored in application-dependent log files and in main memory.

The open source nature of the framework with complete source code and documentation can be leveraged by researchers to generate synthetic digital forensic corpora that are comparable to real-world corpora. The framework supports the generation of a reliable ground truth using a holistic approach with a real-world context that simulates a complete infrastructure with multiple running operating systems, application types and network protocols. An open API enables the synthetic generation of traffic to be programmed. Finally, the framework is modular and extensible, enabling researchers to simulate new operating systems, applications and network protocols.

2. Related Work

Demand for datasets has always been great in the intrusion detection community for evaluating new techniques and comparing their performance against existing ones. As a result, several approaches have been proposed for generating intrusion detection datasets. Molnar et al. [12] reveal that a large number of network traffic generators are available, but most of them are focused on specific application areas, which makes comparative evaluations extremely difficult. Other approaches generate synthetic traffic by mimicking human user activity, but the generated traffic typically is restricted to a single protocol or application type, or only contains data without context.

ID2T [6] is a Python-based network dataset synthesizer that was developed to overcome the shortcomings of the infamous KDD Cup 1999. ID2T essentially creates a new dataset by merging two PCAP-based network traffic dumps, one containing benign traffic and the other containing malicious traffic. During the merging process, ID2T accounts for network characteristics to avoid the artifacts seen in the KDD Cup 1999 dataset [9]. FLAME [2] is similar to ID2T, but it requires NetFlow-based traffic dumps. However, both ID2T and FLAME do not address the problem of generating initial network dumps.

Moirai [3] is a testbed creation framework that supports the emulation of Windows and Linux hosts. The framework uses an INI configuration file to define an "experiment" that has rudimentary support for installation routines, host process control and file exchange, but does not provide keyboard/window manager based controls. In addition, most commits were made during a short period of time, after which the testbed no longer appears to be maintained.

Emulab [8] is a network testbed that enables researchers to define and perform tests of virtual network environments. Emulab simultaneously refers to the open source platform for defining and controlling experimental environments, as well as the actual entity that runs the virtualized systems. A user interface and Python-based scripting language are provided for creating experimental hardware, software and network setups. Emulab supports GENI RSpec to enhance interoperability. DETER is an extension of Emulab that focuses on security-sensitive experiments. It incorporates several architectural changes to prevent malware from compromising other experiments and the underlying infrastructure. However, both Emulab and DETER do not provide opportunities for modeling and simulating user activities.

The LARIAT testbed [16] developed by MIT Lincoln Laboratory supports the modeling of benign and malicious activities in networked environments. Wright et al. [19] have further extended LARIAT to record and replay user activity at the user interface level using Markov chains. This provides a more realistic network footprint compared with the original LARIAT implementation, which was based on statistically-derived network events combined with an application protocol generator. While LARIAT receives good scores for its ability to simulate user activity, a major drawback is its limited accessibility – it is neither open source nor publicly available.

Related work on the automated generation of persistent disk images also deserves mention. NIST [14] has released reference datasets that provide forensic practitioners with simulated digital evidence for examination. Moch et al. [10, 11] have developed the Forensic Image Gener-

ator Generator (Forensig2) that generates filesystem images for digital forensics training courses. Another similar tool is ForGe – Forensic Test Image Generator [18]. Yet another similar system is EviPlant [17], which facilitates the efficient creation, manipulation, storage and distribution of digital forensic challenge problems for education and training purposes. However, all these tools are prototypes. Only the source code of ForGe is available, but the last commit was five years ago.

Traffic generators and image generators are disparate – traffic generators exclusively generate traffic and image generators do not produce traffic. Since the primary goal is to develop a holistic data synthesis framework that generates more than just network traffic and forensic disk images, existing generators are unsatisfactory. The aforementioned approaches do not provide mechanisms for dynamically modeling an entire infrastructure in a modular manner, nor are they open source and maintained or enable the automation of user activity. Indeed, the review of the literature reveals that no generator comparable to `hystck` combines all these features, and synthesizes network traffic and operating system and application artifacts based on human-computer interactions.

3. Framework Architecture and Functionality

This section describes the architecture and functionality of the `hystck` framework.

3.1 Overview

The `hystck` framework is designed to generate network traffic and other relevant digital evidence – that do not differ from real network traffic and disk images – by simulating human-computer interactions. Therefore, a special user interaction model was developed to capture and articulate human-computer interactions. The framework generates datasets by executing user interaction models. In order for user interaction models to generate network traffic, a solution with an operating system and graphical user interface is required. Additionally, it should be possible to install new software that can be executed by a user interaction model at a later time.

Virtualization was chosen when designing the framework because any operating system and applications can be installed on a virtual machine. The Kernel-based Virtual Machine (KVM) is employed for virtualization and `libvirt` is used for KVM administration. Other hypervisors supported by `libvirt` (e.g., VirtualBox) may be integrated into the framework. Since the framework must simulate the behavior of multi-

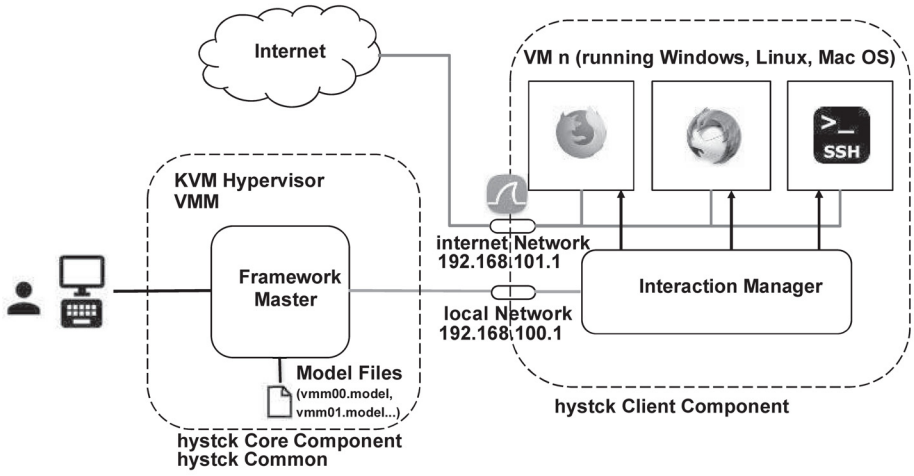

Figure 1. Framework architecture.

ple operating systems, a platform-independent programming language is required. This is why the framework was developed entirely in Python.

3.2 Framework Architecture

The framework engages a client-server architecture. The server-side has a framework master that manages the virtual machines. The client-side has an interaction manager, an agent that runs in the background and controls the graphical user interfaces of the virtual machines.

Figure 1 shows the interactions between the two main components. The framework master communicates with the interaction manager running on a virtual machine via a TCP socket on port 11000. The connection is used to send commands that control applications running on the virtual machine (e.g., start, close and window change) as well as keystrokes and mouse events.

The framework is divided into two parts to ensure that simulations work correctly. One is the server-side (physical machine) on which a specific scenario is implemented. The other part comprises the client-side virtual machines that execute commands in the scenario. Traffic produced by the virtual machines and sent to the Internet (`internet` network) is captured in the PCAP format using the `tcpdump` tool. To ensure that captures do not contain non-relevant control traffic, in addition to the `internet` network, a second network named `local` is created. The `local` network is used for communications between the framework master and interaction manager.

A separate virtual machine (guest) is created for each computer or user to be simulated. Therefore, virtual machine template files for Linux and Windows systems must be created in advance. All the images of the guest virtual machines are derived from the template files so that each simulated user works in an isolated execution environment and can use different software. The interaction manager currently supports the Linux, Windows 7 and Windows 10 operating systems.

The `constants.py` configuration file is used to adapt the framework settings. The configuration file contains information such as the number of virtual machines to be created, the names of the template files, the IP addresses of the `local` network (`192.168.100.1` in Figure 1) and `internet` network (`192.168.101.1` in Figure 1), and the MAC addresses of the virtual machine network interfaces for IP address assignment using DHCP.

The framework initiates the cloning of a virtual machine and establishes the connection to the guest (through which the guest sends and receives commands) based on its MAC address. This is accomplished using the `GuestListener` helper class. After the communications path has been established, the `Guest` class is invoked with the appropriate parameters to actually clone and start the virtual machine. After the virtual machine has started, the network interfaces are extracted and sniffers for the correct interfaces are started. The last key component is the `Agent` class, which handles the connection between the host and a virtual machine by having an agent running as an instance in the guest. At this point, commands may be invoked to start applications and perform tasks.

3.3 Data Synthesis Procedure

Figure 2 provides details about the operation of the framework.

1. The `VMM` class functions as a setup environment to create and control guests. It ensures that the default guest parameters (IP address, MAC address, template, `tcpdump`, etc.) are set to successfully clone templates. Also, it creates sockets on all the interfaces for the agents to listen on the guests.

2. The `Guest` class loads the parameters from the `constants.py` configuration file. The class creates and controls the guests using the template files.

3. The MAC addresses are linked to IP addresses and stored in the network configuration files for use by `libvirt`.

4. The `local` and `internet` networks are created by `libvirt`.

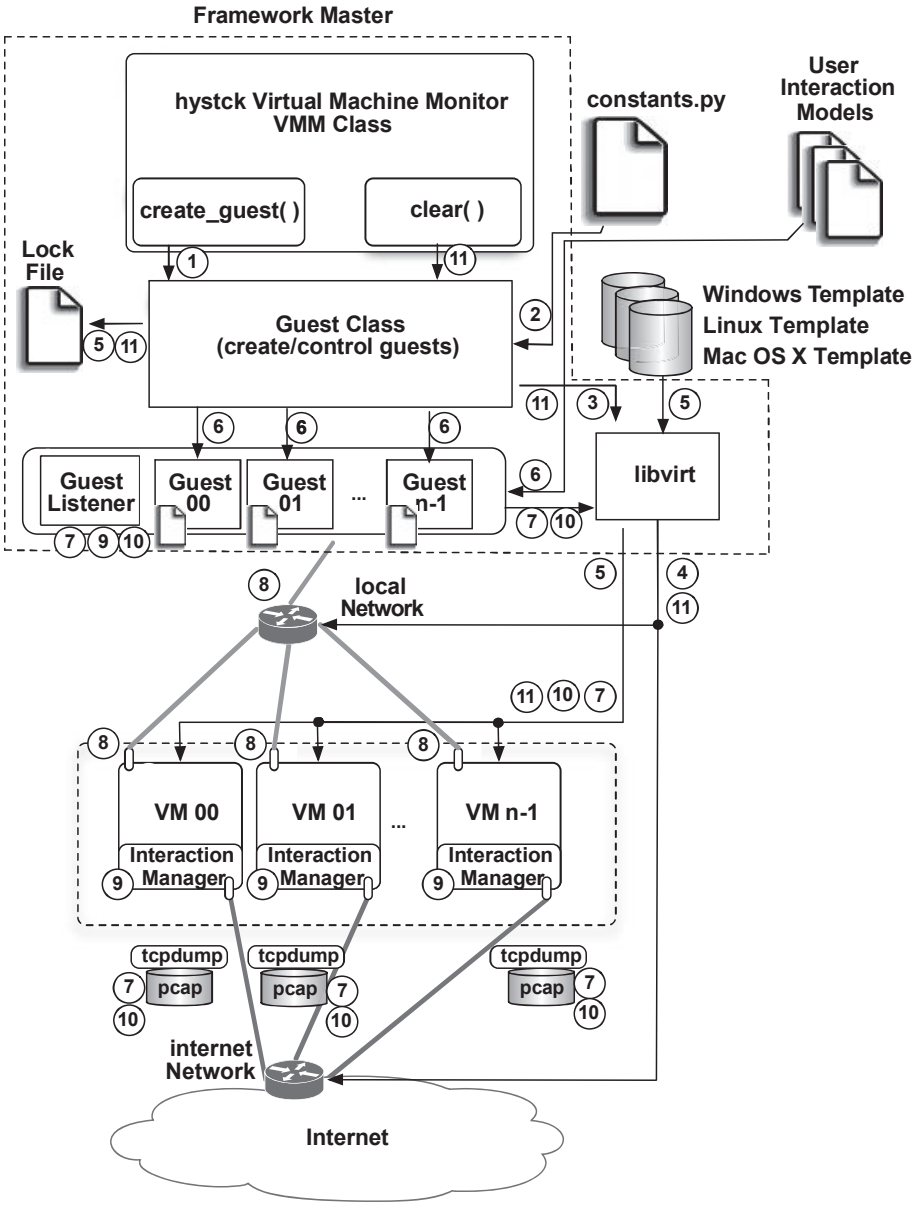

Figure 2. Data synthesis procedure.

5. The `Guest` class creates the virtual machines based on the templates using `libvirt`. In addition, a lock file is created.

6. The `Guest` class causes each guest to load its user interaction model.

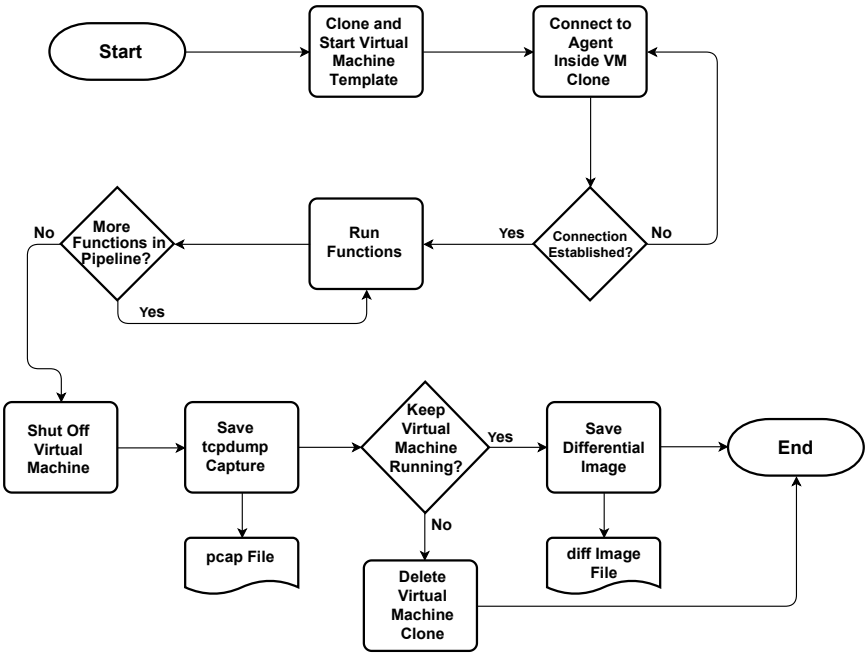

Figure 3. Framework flow diagram.

7. The user interaction models are executed in the guests. This causes the virtual machines to be started by `libvirt`. The `tcpdump` tool is started on the host for each virtual machine in order to record network traffic.

8. The guest instances connect to the virtual machine.

9. The process flow of each user interaction model proceeds, whereby the interaction manager controls the graphical user interface and simulates the desired behavior using the operating system and its applications (e.g., Firefox and Thunderbird).

10. The simulation is complete. The virtual machines are shut down by `libvirt` and `tcpdump` stops capturing traffic.

11. Finally, the virtual machines and the `local` and `internet` network interfaces are deleted by `libvirt`. The lock file is also deleted.

The flow diagram in Figure 3 shows the most important steps in the workflow. The workflow begins with cloning, starting and connecting to the virtual machine. Next, a test is made to check if the connection

Table 1. Supported framework features.

Function	Protocol	Windows 7/10	Ubuntu 19
Firefox browse URL	HTTP/HTTPS	Yes	Yes
Firefox click elements	HTTP/HTTPS	Yes	Yes
Firefox download	HTTP/HTTPS	Yes	Yes
Thunderbird send email	SMTP/SMTPS	Yes	Yes
Thunderbird receive email	POP3/IMAP/IMAPS	Yes	Yes
Thunderbird fill mailbox file	–	Yes	Yes
SSH connection/file transfer	SSH/SFTP	Yes	Yes
SMB file transfer	SMB	Yes	Yes
IPP print job	IPP	Yes	Yes
Pidgin IM and IRC	XMPP/IRC	Yes	No
VeraCrypt create container	–	Yes	NT
VeraCrypt (un)mount container	–	Yes	NT
Execute console commands	–	Yes	Yes
Change system clock	–	Yes	Yes
Multiuser capability	–	Yes	No

has been established; if this is true, the simulation functions are executed. The functions include browsing websites, sending email and exchanging instant messages. After all the simulation functions have been executed, the virtual machine is shut down. Following this, a parameter value is checked to determine if the virtual machine should remain in `virtmanager` or if it should be deleted. For example, the virtual machine image should be maintained if it is desired to correlate network traffic with operating-system-dependent traces. Finally, `tcpdump` is closed and the capture files are saved.

3.4 Supported Features

The framework currently simulates common applications and traffic types. Table 1 shows the list of supported features, including the simulation activities in the operating systems. Note that "NT" means that the feature was not tested. The following details are provided about the supported features:

- **Firefox:** Certain behavior by the Firefox web browser can be simulated. The most important feature is the ability to browse websites. With the support of the web browser environment, it is possible to simulate common traffic types, including typical traffic patterns such as those involving streaming platforms. It is also possible to interact with websites using an `xpath` component to trigger click events on website objects.

- **Thunderbird:** A Thunderbird module is available for creating email traffic. The module supports basic Thunderbird activities such as logging into an email account, and sending and receiving email.

- **SSH/SFTP:** The SSH protocol is widely used to securely log into and transfer files across enterprise server infrastructures. This is easily simulated because the framework provides full Linux Bash shell and Windows command line functionality.

- **Pidgin:** The Pidgin chat client supports a variety of chat services. It is possible to simulate instant messaging traffic (e.g., IRC) such as sending and receiving chat messages.

- **Botnets:** A module is available for simulating botnet attacks by Zeus, Asprox, Mariposa and Waledac. It is possible to generate network dumps of entire attacks from the victims' perspectives.

- **VeraCrypt:** A module is available for simulating the creation of encrypted containers. The module is only used in the image generation part.

3.5 Network Traffic Synthesizer

As mentioned above, the guest systems and virtual machines have two network interfaces. The `local` interface is for transmitting `hystck` commands while the `internet` interface is a bridge to the Internet. From the traffic generation point of view, the `internet` interface is interesting because all relevant traffic flows through it. The separation was implemented to prevent framework communications from showing up in network captures. Specifically, it ensures that all framework communications, such as telling a guest to browse a website, are handled by the `local` interface and everything else, such as actually browsing the website and sending and receiving email, are handled by the `internet` interface.

To obtain a traffic dump, `hystck` searches for a specific IP address and starts `tcpdump` with the corresponding network interface as a parameter. The `tcpdump` tool then captures every packet traveling through the network, outgoing as well as incoming. The tool continues to capture all the packets as long as the guest operating system is running and dumps them into a specified file.

In an experiment with Firefox, only two websites were visited, but the captured traffic exceeded 17,000 network packets. Many of the packets were related to the Firefox landing page (about 20 pages are loaded when

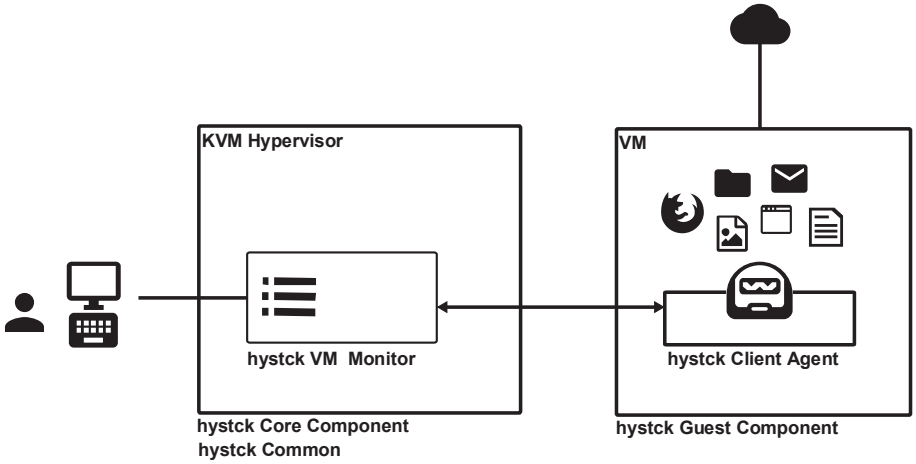

Figure 4. Generating persistent disk images.

the browser is opened). However, another large – and more important – group comprised packets associated with Windows services and updates that executed in the background from the time the operating system was started. Clearly, much more traffic would be captured when simulating complete operating system behavior. Thus, the framework can generate more realistic datasets than other traffic generators.

3.6 Disk Image Generator

In addition to network traffic, the framework can be used to generate persistent disk images as in [10, 11, 17, 18]. Depending on the user interaction model, the framework can partially automate forensic image generation, including background data and planted evidence.

Figure 4 shows how the framework can emulate a number of applications on the guest systems; this is highly versatile. Additionally, the modular architecture enables new applications to be added. It is possible to specify and emulate a variety of user behaviors on guests using simple user interaction models. Many plugins for web browsing, program execution and file injection are provided. System time can be altered automatically to simulate system usage over a large interval of time. The framework also creates a report of all the activities performed on the image with hash values. At first glance, an image would be indistinguishable from the image of an actual system operated by a user. This is advantageous for education and training purposes where a variety of datasets can be created in an automated manner. Only one large base image is necessary in the framework. The distribution of the actual im-

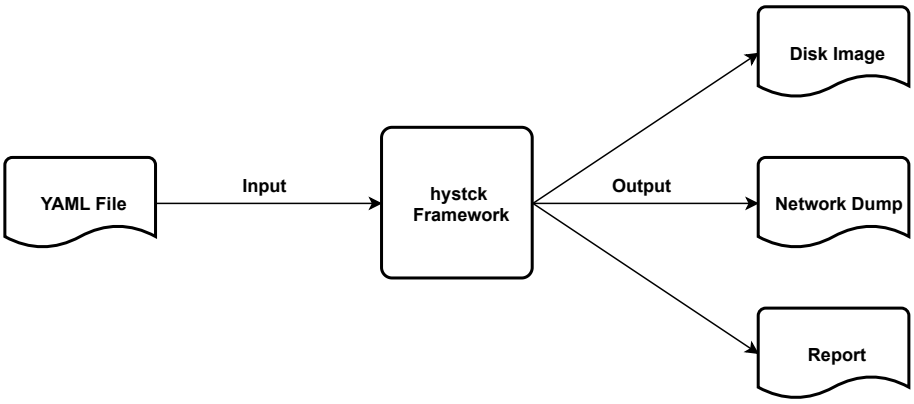

Figure 5. Generator workflow.

age is performed via snapshots in the QEMU-based file format, which are less than 1 GiB and only include the differences from the base image.

Locard's exchange principle states that every contact leaves a trace. This poses a problem during image generation. Every interaction with the operating system leaves a trace – usually multiple traces. These were seen in the framework components inside a virtual machine (e.g., Python installation, `hystck` agent source code, third party libraries, Bash scripts for the startup process and network connections to the virtual machine monitor). Future research will attempt to reduce the presence of these artifacts in images.

4. Generator and Reporter

This section describes the automated generation of images and network dumps. Also, it describes the reporting function, which summarizes everything that occurred in a virtual machine.

4.1 Generator

The `hystck` generator can almost automatically produce large amounts of traces and traffic. For example, the generator could instruct the framework to send hundreds of email messages and to browse numerous websites in order to generate benign and malicious data. Benign data corresponds to the generated haystack of good files. Malicious data corresponds to needles inserted in the haystack for subsequent detection.

The generator produces a variety of data types – normal, suspicious and malicious data (e.g., PDFs, images and text). As Figure 5 demonstrates, the generator takes in a YAML file and calls the functions associated with the YAML tags. The YAML configuration file helps setup

the traffic and image generation process. Next, several action suites are created that describe the order and protocols under which files are sent; the action suites execute on the virtual machines. If desired, the data to be sent over the network may be selected randomly.

The configuration file grants wide access to framework functionality because special programming expertise is not needed to generate different types of traffic. Moreover, every portion of the YAML file is described in detail in the documentation, which enables users to understand what can be done with the generator and how it can be done.

In addition to the generator, a separate virtual machine is used to provide necessary services (e.g., Samba server, print server and local mail server). The virtual machine is started headless without a graphical user interface. If the virtual machine is installed as specified in the documentation, then the framework knows exactly where to find all the relevant services.

Users who have good understanding of Python and the `hystck` codebase have the option of manually scripting scenarios without using the generator. An example of manually-generated user interaction model source code is shown later (Figure 7). Note that it is not necessary to develop models independently because tweaking the YAML file of the `hystck` generator could produce the same outcome or even richer outcomes as the operations are configurable.

4.2 Reporter

The framework and its generator make it easy to produce datasets with hundreds or thousands of executed operations. This would make it very difficult to manually keep track of every detail of every operation. The `hystck` reporter keeps track of what happened and when, maintaining information about every benign and malicious operation. The reporting function is very useful for education and training, enabling instructors to tailor images and/or network dumps to their teaching goals. Additionally, the reporter could produce an XML document that assists an instructor in grading student work (e.g., providing all the details about a scenario). This document could then be viewed via the web viewer integrated in the framework or using a tool that can parse XML files.

5. Framework Validation

As mentioned above, the `hystck` framework was specifically designed to simulate human-computer interactions in order to generate forensic images and network traffic that would at best be indistinguishable

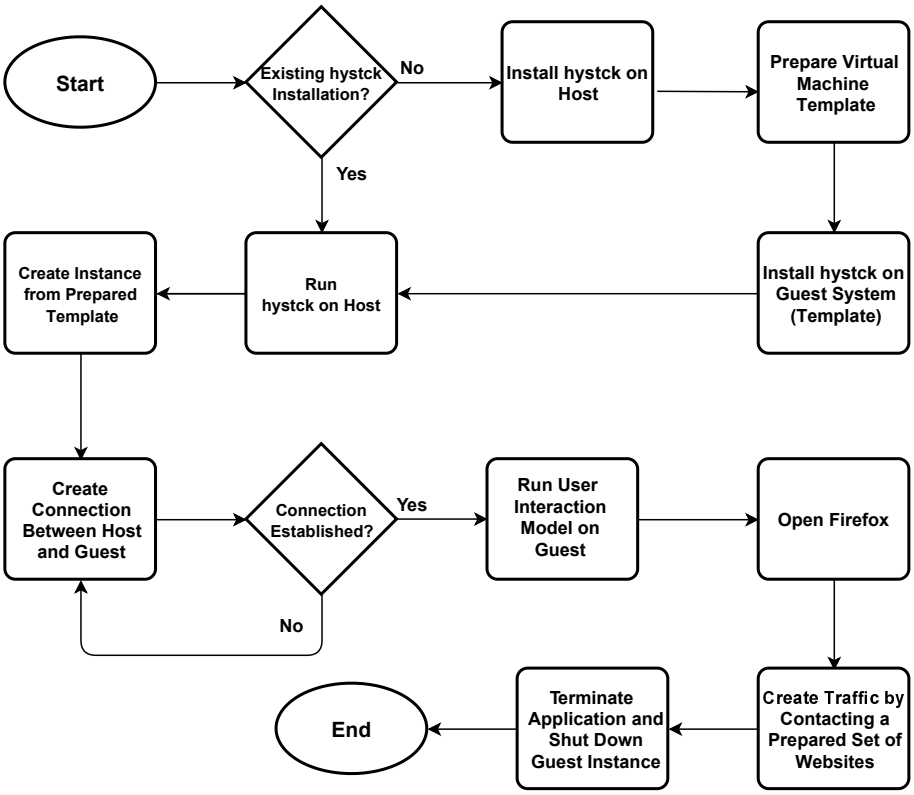

Figure 6. Traffic generation using Firefox.

from real-world human-computer interactions. This section describes two practical scenarios involving the generation of web and mail traffic using the framework. The section also shows how simulations of other applications can be integrated into the framework. Not every function can be discussed here, which is why interested readers are referred to the GitHub repository (`github.com/dasec/hystck`) for technical details.

5.1 Web Traffic and Mail Traffic Generation

While the modular structure of the framework supports the development and implementation of custom user interaction models to generate traffic and images for a variety of scenarios, several implemented models can inspire the creation of new scenarios with the same or different applications. This section describes two implemented models involving Firefox and Thunderbird.

Figure 6 shows traffic generation using the Firefox application. After installing the `hystck` framework and preparing the virtual machine tem-

plate as described above, the appropriate Python script is executed. The framework creates a copy of the prepared virtual machine and starts it. The virtual machine then establishes a connection to the host, following which the user interaction model is executed. In this example, the interaction manager, which runs on the client-side, opens an instance of Firefox, visits multiple websites in succession and downloads content before closing the application.

Figure 7 shows the code snippet corresponding to the user interaction model that simulates Firefox behavior. In the snippet, the framework waits for the connection to the interaction manager to be established, after which the web browser simulation begins. Depending on the chosen functions, it is possible, for example, to visit and interact with websites, download files and log into a Facebook account.

The procedure for generating traffic with Thunderbird is quite similar to the Firefox example shown at the bottom of Figure 7. One difference is that user authentication is required when using an email service. Therefore, before traffic is generated, the interaction manager must receive the user login credentials and the recipient's email address from the host-side script. Traffic is subsequently generated by sending an email from the user to the recipient.

Finally, as discussed above, the entire generation process is easily automated and repeated multiple times using the framework generator.

5.2 Framework Extensions

The framework is designed with extensibility in mind. A variety of user activities and operating system behaviors based on new applications can be generated at any time. The applications may be added using the `skeleton.py` file located in the application folder in the GitHub repository. The file provides developers with a basic structure of four classes. However, only two classes, `SkeletonVmmSideCommands` and `SkeletonGuestSideCommands`, are relevant. The other two classes are only for command parsing and do not need to be altered.

The `SkeletonVmmSideCommands` class is responsible for the host-side assembly of commands; it implements the functions that send the appropriate commands. The `SkeletonGuestSideCommands` class is the actual implementation. Everything written in this class is executed by the virtual machine and causes actions by an application or the operating system.

Two functions are provided in the skeleton to facilitate the addition of applications. The `open` and `close` functions trigger the corresponding actions in the application. Additional assistance for integrating custom

```
1   from hystck.core.vmHelper import vmHelper
2   from hystck.core.vm import vm
3
4   vmhelper = vmHelper()
5   vmobj = vmhelper.createVM(vmname="windows-vm01", template=
        "windows")
6
7   #wait for vm
8   for vm in vmhelper.vms:
9       while vm.state != "connected"
10          time.sleep(0.1)
11
12      # vm.state == "connected":
13      # Email login data for Thunderbird
14      mail.add_imap_account("imap.web.de", "smtp.web.de",
15          "anonymous_sender@web.de", "password", "Sender Name",
16          "Example Subject", 2, 3, 1, 3)
17      ...
18      # begin of firefox session by opening the application
            and visiting a website
19      browserobj = vm.openBrowser("firefox", "www.h-da.de")
20      # downloading via firefox
21      browser_obj.download_from(
22          'https://www.nist.gov/itl/ssd/software-quality-group/' +
23          'computer-forensics-tool-testing-program-cftt/' +
24          'computer-forensic-0-0-2',
25          'Video dd files (zip format)')
26      # login for facebook via firefox
27      browserobj.facebook_login(username="user@domain.de",
            password="password")
28      browserobj.close()
29      ...
30      # enter recipient address and message and send email
31      mail.send_mail(message="testmail", subject="testmail",
32          receiver="anonymous_receiver@web.de")
```

Figure 7. User interaction model code simulating Firefox and Thunderbird behavior.

applications is available in the template class files and framework documentation.

6. Future Work

While the framework is versatile and supports diverse tasks, it will never cover all the use cases because new operating systems, applications and protocols will always have to be integrated. However, the

modular architecture facilitates extensibility by enabling new user interaction models and scenarios to be incorporated. The framework can be made more functional by increasing the number of supported applications, including applications that use protocols such as FTPS, SIP and VoIP, and messenger applications such as WhatsApp Desktop and Skype. Many applications are cloud-based or require network connections to function and, therefore, generate traffic in the background almost "automatically." In addition to generating benign traffic, it is also necessary to simulate network attacks for attack traffic generation. The first steps in this direction have already been implemented by inserting malicious data (needles) into the network stream using the generator and by incorporating the botnet component.

As far as the core components of the framework are concerned, an important future task is to add data synthesis support for macOS. Framework agents will also be developed for Android and iOS.

Since the framework is based on the client-server model, it is cumbersome to test code changes quickly. This will be addressed by developing an efficient test base and debugging mechanisms. Another important task is to reduce artifacts in the generated images (e.g., agent source code and virtual machine monitor connections). In addition, it is necessary to integrate automated memory dump processes for Linux and Windows virtual machines to correlate relevant information with data captured by `tcpdump` and data stored in virtual machine persistent storage.

Other research problems are modeling normal user behavior and simulating daily routines and environment interactions in network infrastructures. Chinchilla et al. [5] describe some characterizations of Internet traffic and perform a user classification. Studies will be performed on user behavior to produce high-fidelity user simulations. It will also be necessary to develop metrics for dataset quality, including comparing synthetically-generated and real-world datasets.

7. Conclusions

The `hystck` framework offers a novel approach for generating digital forensic datasets. The data synthesis approach is more holistic than existing traffic generators. The framework models and simulates real human-computer interactions to generate network traffic and forensic images.

A key advantage of the framework is its extensibility. The modular, plugin-based design involving user interaction models enables users to write and integrate plugins for custom system interactions and realistic

traffic generation. Another advantage is scalability – it is possible to simulate a network of computers with diverse operating systems, users, applications and protocols, each running in their own virtual machines. More systems can be added to the network environment at any time by simply introducing more virtual machines. The framework also offers an efficient image distribution feature because only small differential images need to be shared. Furthermore, the framework operates in a time-independent manner as the system time can be changed at will; this makes it possible to simulate multiple user interactions during a desired time interval. Additionally, the framework is case independent in that it is not necessary to model user behavior manually; an API is provided to automatically interact with the framework.

As the future research tasks are implemented, the framework will evolve to become a powerful dataset generator. These datasets can be used for new network traffic analysis tools and network forensic approaches, simulating realistic user interactions for classical drive forensics, analyzing memory dumps and investigating network usage and data storage by mobile device applications.

The latest version of the framework source code is downloadable from the GitHub repository (`github.com/dasec/hystck`). The framework documentation, an installation guide and details about framework features and adding custom code for simulating other application types are available at `hystck/docs`. The pre-compiled documentation is available at `hystck/docs/src/_build/html/index.html`.

Acknowledgements

This research was supported by the German Federal Ministry of Education and Research (BMBF) under Forschung an Fachhochschulen (Contract No. 13FH019IB6) and by the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) under CRISP. The authors also wish to thank Reinhard Stampp and Sascha Kopp, who played important roles in implementing the framework.

References

[1] S. Abt and H. Baier, Are we missing labels? A study of the availability of ground truth in network security research, *Proceedings of the Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pp. 40–55, 2014.

[2] D. Brauckhoff, A. Wagner and M. May, FLAME: A flow-level anomaly modeling engine, *Proceedings of the Conference on Cyber Security Experimentation and Test*, article no. 1, 2008.

[3] G. Brogi and V. Tong, Sharing and replaying attack scenarios with Moirai, presented at the *Rendezvous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information (Information Systems Security Research and Education Meeting)*, 2017.

[4] B. Carrier, Open Source Digital Forensic Tools: The Legal Argument, @stake, Cambridge, Massachusetts, 2002.

[5] R. Chinchilla, J. Hoag, D. Koonce, H. Kruse, S. Osterman and Y. Wang, Characterization of Internet traffic and user classification: Foundations for the next generation of network emulation, *Proceedings of the Tenth International Conference on Telecommunications Systems, Modeling and Analysis*, 2002.

[6] C. Cordero, E. Vasilomanolakis, N. Milanov, C. Koch, D. Hausheer and M. Muhlhauser, ID2T: A DIY dataset creation toolkit for intrusion detection systems, *Proceedings of the IEEE Conference on Communications and Network Security*, pp. 739–740, 2015.

[7] C. Grajeda, F. Breitinger and I. Baggili, Availability of datasets for digital forensics – And what is missing, *Digital Investigation*, vol. 22(S), pp. S94–S105, 2017.

[8] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb and J. Lepreau, Large-scale virtualization in the Emulab network testbed, *Proceedings of the USENIX Annual Technical Conference*, pp. 113–128, 2008.

[9] M. Mahoney and P. Chan, An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection, *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, pp. 220–237, 2003.

[10] C. Moch and F. Freiling, The Forensic Image Generator Generator (Forensig^2), *Proceedings of the Fifth International Conference on IT Security Incident Management and IT Forensics*, pp. 78–93, 2009.

[11] C. Moch and F. Freiling, Evaluating the Forensic Image Generator Generator, *Proceedings of the International Conference on Digital Forensics and Cyber Crime*, pp. 238–252, 2011.

[12] S. Molnar, P. Megyesi and G. Szabo, How to validate traffic generators? *Proceedings of the IEEE International Conference on Communications Workshops*, pp. 1340–1344, 2013.

[13] National Institute of Standards and Technology, Computer Forensic Tool Testing (CFTT) Program, Gaithersburg, Maryland (`www.nist.gov/itl/ssd/software-quality-group/computer-forensics-tool-testing-program-cftt`), 2019.

[14] National Institute of Standards and Technology, The CFReDS Project, Gaithersburg, Maryland (`www.cfreds.nist.gov`), 2019.

[15] M. Ring, S. Wunderlich, D. Scheuring, D. Landes and A. Hotho, A survey of network-based intrusion detection datasets, *Computers and Security*, vol. 86, pp. 147–167, 2019.

[16] L. Rossey, R. Cunningham, D. Fried, J. Rabek, R. Lippmann, J. Haines and M. Zissman, LARIAT: Lincoln adaptable real-time information assurance testbed, *Proceedings of the IEEE Aerospace Conference*, 2002.

[17] M. Scanlon, X. Du and D. Lillis, EviPlant: An efficient digital forensics challenge creation, manipulation and distribution solution, *Digital Investigation*, vol. 20(S), pp. S29–S36, 2017.

[18] H. Visti, S. Tohill and P. Douglas, Automatic creation of computer forensic test images, in *Computational Forensics*, U. Garain and F. Shafait (Eds.), Springer, Cham, Switzerland, pp. 163–175, 2015.

[19] C. Wright, C. Connelly, T. Braje, J. Rabek, L. Rossey and R. Cunningham, Generating client workloads and high-fidelity network traffic for controllable repeatable experiments in computer security, *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, pp. 218–237, 2010.