Chapter 2

REALISTIC AND CONFIGURABLE SYNTHESIS OF MALWARE TRACES IN WINDOWS SYSTEMS

Martin Lukner, Thomas Göbel and Harald Baier

Abstract Malware constitutes a long-term challenge to the operation of contemporary information technology systems. A tremendous amount of realistic and current training data is necessary in order to train digital forensic professionals on the use of forensic tools and to update their skills. Unfortunately, very limited training data images are available, especially images of recent malware, for reasons such as privacy, competitive advantage, intellectual property rights and secrecy. A promising solution is to provide recent, realistic corpora produced by dataset synthesis frameworks. However, none of the publicly-available frameworks currently enables the creation of realistic malware traces in a customizable manner, where the synthesis of relevant traces can be configured to meet individual needs.

This chapter presents a concept, implementation and validation of a synthesis framework that generates malware traces for Windows operating systems. The framework is able to generate coherent malware traces at three levels, random-access memory level, network level and hard drive level. A typical malware infection with data exfiltration is demonstrated as a proof of concept.

Keywords: Forensic datasets, data synthesis framework, malware traces, `hystck`

1. Introduction

Contemporary digital forensic investigations are encountering large amounts of increasingly complex traces that have to be analyzed [18]. Meanwhile, attackers are using sophisticated techniques to obfuscate their traces. As a result, it is becoming increasing necessary to employ tools that automate portions of digital forensic investigations. Simul-

© IFIP International Federation for Information Processing 2022
Published by Springer Nature Switzerland AG 2022
G. Peterson and S. Shenoi (Eds.): DigitalForensics 2022, IFIP AICT 653, pp. 21–44, 2022.
https://doi.org/10.1007/978-3-031-10078-9_2

taneously, digital forensic professionals need to be trained to deal with complex traces [19].

Developing a digital forensic tool requires considerable effort. In addition to providing the required capabilities, the tool and the results it yields must be accepted in judicial proceedings. This requires the tool to meet certain criteria [5]. An important criterion is testing, which ensures that the tool has been evaluated thoroughly. This requires a large amount of real-world data with adequate coverage.

Unfortunately, labeled data sets are rare in the cyber security domain as well as in specialized areas such as network security, biometrics and digital forensics [1, 11, 13]. A study by Abt and Baier [1] reveals that 70% of the published papers in network security rely on self-compiled datasets and only 10% of the datasets are released to the public.

A key problem with publicly-available datasets is that they are often constructed for special research studies and may not adequately represent real-world scenarios. Another problem is the paucity of datasets. Malware analysis is an important task in contemporary digital forensic investigations, but publicly-available corpora containing traces of executed malware are exceedingly rare [13]. This situation is primarily due to reasons such as privacy, competitive advantage, intellectual property rights and secrecy. What is needed is a configurable tool that can automatically generate malware-related forensic images for forensic tool testing as well as for training forensic professionals.

This chapter presents a concept, implementation and validation of a synthesis framework that generates malware traces for the Microsoft Windows operating system. The concept and implementation leverage the `hystck` framework [12], which is extended by a malware generation module. The extension is able to imitate characteristics of recent malware. Unlike current synthesis frameworks, it provides coherent digital forensic traces at three levels, random-access memory (RAM) images, network dumps and persistent hard drive images. The source code of the malware synthesis extension is available at GitHub [3].

The synthesis framework focuses on Windows-based malware because of the large global footprint of computing systems running Microsoft Windows operating systems. According to Statista [30], roughly 84% of the malware released during the first quarter of 2020 affected Windows operating systems. The framework relies on a client-server malware infrastructure model. It considers different types of communications between a remote access tool and command-and-control server. Configuration files are employed to create various, easily adaptable real-world scenarios with the respective ground truths. The validation reveals that the implementation is successfully integrated in the `hystck` framework

Table 1. Overview of related work on forensic data synthesis.

Tool or Authors	**Year**	**Type**	**Active**	**Tool Code Available**	**Image Types and Traces**
Carrier [6]	2010	Man	No	NA	R,P
Hadi [14]	2011	Man	No	NA	P,M
Garfinkel et al. [11]	2009-14	Man	No	NA	P
NIST [26]	≤2019	Man	Yes	NA	R,N,P,M
`Honeynet` [31]	2010-15	Man	No	NA	N,P,M
ID2T [8]	2015	Syn	No	Yes	N,M
FLAME [4]	2008	Syn	No	No	N,M
Forensic Image Generator Generator [22]	2011	Syn	No	No	P
Forensic Test Image Generator [33]	2015	Syn	No	Yes	P
EviPlant [29]	2017	Syn	No	Yes	P
TraceGen [10]	2021	Syn	Yes	No	P,N
`hystck` [12]	2020	Syn	Yes	Yes	R,N,P

and that the configured traces exist at the RAM, network and hard drive levels to provide coherent pictures of malware infections.

2. Related Work

This section discusses related work in the context of dataset generation and shows that no publicly-available, configurable data synthesis framework exists for generating forensically-relevant images for malware investigations.

Grajeda et al. [13] have discussed the availability of datasets for digital forensics and what is missing. A key gap exists with regard to framework-generated datasets where the framework code is publicly available and holistic views of the datasets are possible. Of particular interest are datasets that provide volatile, network and persistent images, and forensically-relevant traces of malware activity.

Table 1 provides an overview of related work on forensic data synthesis. The first column lists the tool or authors (researchers), the second column indicates when the tool or image was last updated and the third column specifies if the forensic image was created manually (Man) or if the work relates to a synthesis tool (Syn). The active and code available columns indicate current support of the tool or image and code availability, respectively. The sixth (last column) deals with the image type and malware traces. A generated image type is designated by R for RAM, N for network and P for persistent. An M designates if malware traces are present in the image or may be generated.

Manual Image Generation. Manually-generated forensic images are commonly used for digital forensic practice and training purposes. Such forensic images have been created by Carrier [6], Hadi [14], Garfinkel et al. [11] and NIST [26]. If a forensic image is created manually, it can be assumed that all the traces it contains are intended.

However, manually-generated forensic images are difficult to modify because the entire images have to be created anew. As a result, there are relatively few manually-generated images. Also, some forensic images, notably the images created by Garfinkel et al. [9, 11], are subject to access restrictions imposed by U.S. law [9, 35]. Additionally, manually-generated images quickly become outdated because they use old hardware, operating systems and/or versions of installed software.

The static nature of manually-generated forensic images poses another problem. When images are used for forensic training purposes or in forensic challenge competitions, the solutions are disseminated over the Internet, which negatively impacts learning and training. Techniques are available to address this problem [34]. However, it is much easier to automatically create individual images based on parameters that are set in a configuration file before the data synthesis process and determine the traces in the resulting forensic image.

Network Traffic Generation. Traffic generators produce network dumps for digital forensic investigations. The ID2T framework [8] enables the creation of labeled datasets for testing intrusion detection systems. Packet captures of arbitrary networks are collected and malicious traffic is injected into them to simulate network attacks such as distributed-denial-of-service attacks. ID2T also supports modern protocols such as IPv6.

FLAME [4] works in a manner similar to ID2T, but it generates network flows. The network flows contain basic information about grouped packets in flows specified in the NetFlow format and IPFIX standard.

Drive Image Generation. Drive image generators enable the creation of forensic images of persistent storage devices such as hard drives, solid-state drives and USB drives. The Forensic Image Generator Generator [22, 23] demonstrates the feasibility of developing generators of forensic images for students. The input to the generator comprises scripts written by an instructor and the output comprises a filesystem image and an automatically-generated report in human-readable form that defines the ground truth. The Forensic Image Generator Generator creates traces in a virtual machine (VM), enabling the system settings to be modified as required. However, the framework is out-of-date and is no longer maintained.

The Forensic Test Image Generator (ForGe) [33], unlike the Forensic Image Generator Generator, provides a user interface. Input instructions are provided in the form of database entries and the output contains drive images and information sheets. Although the tool is available at GitHub [32], it has not been updated since 2015 and does not provide network logs or RAM dumps.

The EviPlant framework [29] is a more recent image generator. The framework makes use of a base drive image, which can be downloaded. Additionally, challenges or traces are available in the form of `evidence packages`. In order to obtain a forensically-relevant drive image, a chosen evidence package has to be injected into the base image. However, injecting consistent traces is very difficult and the manual work involved in creating traces is barely reduced.

Multilevel Image Generation. Multilevel image generators provide at least two of the three levels of images, volatile, network and persistent images. TraceGen [10] is a recent framework that captures network traces as well as hard drive traces. However, the approach is currently published as a proof of concept, meaning that no code is available. To combine the advantages of manual and automatic trace generation, APIs such as `pywinauto` [20] are used to simulate user interactions with the graphical user interface. For example, setting a registry key via the interface generates different traces than when setting a registry key via a command line. This also solves the problem of evidence packages in the `EviPlant` framework. It appears that all available scenarios are in the form of Python scripts or all individual actions are in lists. This does not make it very user-friendly, especially for users with limited technical experience.

The `hystck` [12] framework can create traces in RAM, network logs and hard drive images. This is done automatically using Python scripts or via YAML configuration files. Automated synthesis makes it possible to create a variety of traces with little effort. The traces can be distributed efficiently in template and differential images.

Summary. Unfortunately, existing data synthesis frameworks support the creation of limited scenarios. Typical malware behavior may be replicated by certain commands, but important aspects such as infection vectors or RAM artifacts and important events, such as Syslog events, are missing. Frameworks that mimic malware techniques, such as MITRE's Caldera [21], have different goals than the synthesis frameworks discussed above. For example, Caldera can be used for red or blue team operations as well as for testing servers or security teams whereas frameworks such as `hystck` generate traces in template virtual machines in large quantities in an automated manner. Table 1 shows that no framework exists that combines the advantages of automated synthesis and malware trace generation. Moreover, the manually-created datasets containing RAM dumps, network captures and drive images are not coherent. In contrast, `hystck` is the only synthesis framework that provides coherent multilevel traces. This is where the work described in this chapter begins and it ultimately enables the creation of complex malware scenarios that are not generated by any other framework.

3. Forensic Dataset Synthesis Framework

Instead of creating a new framework for synthesizing malware traces from scratch, it was decided to extend `hystck`, the existing framework with the best fit. The extended `hystck` framework must create digital forensic traces by simulating natural human-computer interactions. The generated traces are intended to be as indistinguishable as possible from real-world RAM snapshots, network traffic and hard drive images.

In order to have good control over the generated traces, `hystck` employs a virtualization solution that leverages KVM and QEMU. The actual implementation makes use of Python because a platform-independent programming language enables the creation of traces for different guest operating systems.

The `hystck` framework has a client-server architecture (Figure 1). The server is responsible for controlling the client, which runs in the background and controls the guest's graphical user interface via an interaction manager. To prevent control traffic and forensically-relevant Internet traffic from mixing, the client component communicates using two network cards, an Internet card and a local network card. The local network is only used for control traffic between the client and server. Packets are captured at the Internet network interface using the `tcpdump` tool and stored in a PCAP file.

A template file is used as a base for each image that is created. The Linux, Windows 7 and Windows 10 operating systems are supported at this time. Important settings, such as the number of virtual machines to be created, template file names, IP addresses and actions to be taken, are stored in a text-based configuration file and are easily adapted. The `hystck` generator component (Figure 2) facilitates the creation of large

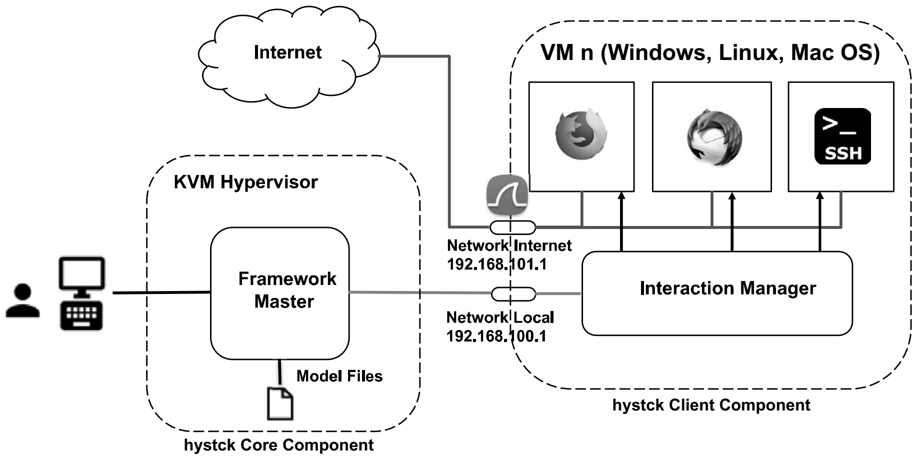

Figure 1. `hystck` architecture [12].

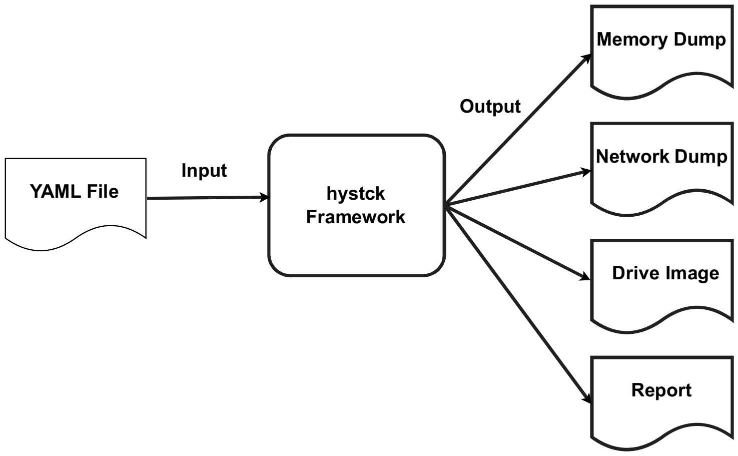

Figure 2. Generator component workflow [12].

amounts of traces using an easy-to-configure YAML file, eliminating the need to code complex Python scripts.

The actual creation of traces is accomplished by cloning the template virtual machine and establishing a connection with the guest virtual machine. Following this, sniffers are started on the corresponding interfaces and the desired actions are carried out by the agents on the virtual machines. Finally, the memory content is captured, the `tcpdump` tool is terminated and the captured contents are made available as RAM dumps and PCAP files. The PCAP files contain the forensically-relevant traces that should be explicitly generated as well as additional network traffic (e.g., realistic background noise corresponding to network communications of standard operating system services such as updates). In addition to creating PCAP files, `hystck` can create persistent drive images with the drive image generator. Several applications can be emulated and updated constantly due to the modular architecture. The applications are controlled via simple guest user interaction models. This makes it almost impossible to distinguish a generated image from an image created by real user interactions. Additional services such as email and file services required by the data synthesis process are provided by separate service virtual machines.

At the end of the data synthesis process, all the changes made, whether malicious or benign, are summarized by the reporting component. These are compiled into an XML-based report file that informs the user about all the generated traces.

4. Malware Dataset Synthesis

This section describes the approach for creating a forensic image generation framework that provides malware traces at the volatile (RAM), network and persistent (hard drive) levels.

4.1 Requirements

The malware dataset synthesis approach relies on the `hystck` framework because it is the only currently-maintained framework that publishes its code, is configurable and enables the generation of traces at the three desired levels. The requirements are defined to enable the development of realistic, complex scenarios. The `hystck` extension should make it possible to create malware traces in RAM, network traffic and on the hard drive. During real malware analysis, a digital forensic professional is usually confronted with traces such as command-and-control communications patterns, various persistence mechanisms and other artifacts created by malware. A client-server architecture is required to generate all these types of artifacts.

In particular, the proof-of-concept implementation uses the Windows registry, the creation of a service and dynamic link library search-order hijacking for persistence mechanisms. From an analyst's point of view, the Windows registry has the advantage of occurring twice, volatile in the RAM dump and persistently on the hard drive. To simulate as many different scenarios as possible, several protocols that are typically encountered during malware infections are employed. One is HTTP, which enables the transfer of large files. A second protocol is raw TCP

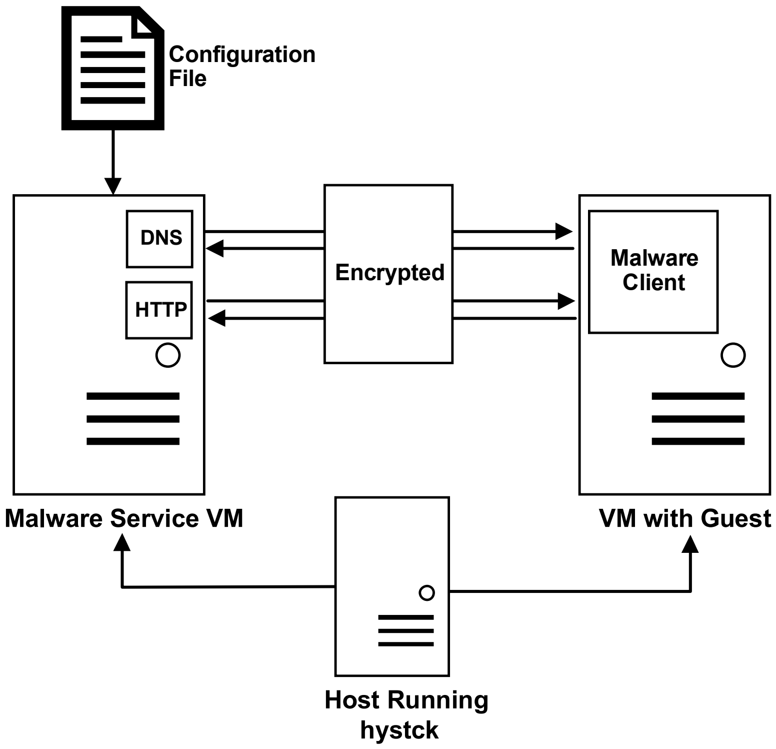

Figure 3. High-level architecture of the `hystck` malware synthesis extension.

transfer. A third is DNS, which is used less frequently by malware, but offers advantages in networks with firewalls because DNS traffic is often not filtered and is essential for a network to function. To create different levels of difficulty during forensic analyses, various encoding schemes and encryption methods are implemented. Indeed, the algorithms commonly used by malware authors are employed; these include Base64 as an encoding scheme and RC4 and AES for encryption. In order to focus forensic analysis on the various traces left by the malware as well as to realize different attack vectors, the client component must be delivered to the target system using different droppers in different ways.

4.2 Framework Architecture

Figure 3 shows the high-level architecture of the `hystck` malware synthesis extension. Several components are required to implement the desired functionality. The host needs a running instance of `hystck`. In order to use the malware synthesis component, the framework extension requires a server component and a client component. Since the malware service has to be implemented in C++ to use typical Windows libraries, it is necessary to integrate a second Windows service virtual machine (malware service VM), which serves as a command-and-control server, in addition to the existing Linux service virtual machine. The malware service virtual machine can be configured dynamically, which simplifies the integration of new scenarios. Additionally, the guest virtual machines of `hystck` are augmented with a corresponding malware client component. Communications between the client and server components employ the network protocols mentioned above.

Droppers and Delivery. Malware can be delivered by the client component in multiple ways. In the proof-of-concept implementation, a Microsoft Office macro is used as a dropper that downloads the entire malware. The dropper can be used as a normal variant and as a variant with special properties for bypassing antivirus software. To pass malware checks unhindered, VBA-stomping is employed, which exploits the undocumented property of VBA macros whereby only contained p-code is executed instead of the VBA macro code [7, 15, 16]. To create appropriate Microsoft Office documents, the EvilClippy tool available at GitHub [27] can be used. As an infection vector, the framework supports the delivery of malware in an automated manner via browser download, automated email delivery and automated download via the command line using `curl`.

Configurable Forensic Scenario Synthesis. In order to create as many different malware scenarios as possible and prevent digital forensic professionals from attaching importance to unimportant things, the server component for malware synthesis is generally first configured at startup using a JSON-like configuration file that is specific to each scenario. The advantage of a dynamically-configurable server component is also evident when, for example, multiple forensic images must be generated for machine learning algorithms. The configuration file enables a framework user to specify in advance exactly which commands should be executed during data synthesis. After the malware is launched on the target system, it gradually requests the commands to be executed.

Figure 4 shows the configurable malware scenario synthesis workflow. After all the commands in the configuration file have been executed, a special command is sent that finally terminates the malware on the victim machine. After the synthesis process, the intended traces should be present in the generated memory and hard drive images for subsequent forensic analysis. In addition, all the network traffic between the client

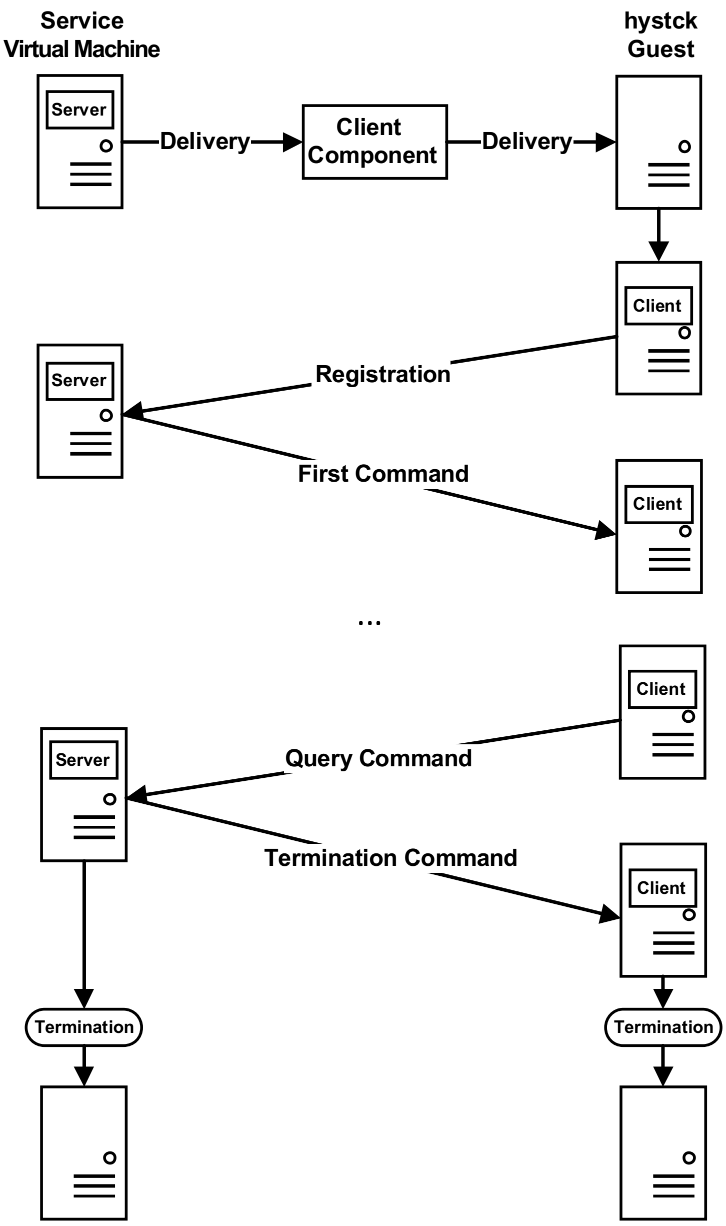

Figure 4. Configurable malware scenario synthesis workflow.

and server is stored in the PCAP files that are automatically generated by `hystck`.

Table 2. Malware synthesis commands.

Command	**Explanation**
Nothing	Does nothing and exists only for testing purposes
Execute	Executes arbitrary commands via the Microsoft Windows command line
Download	Downloads a file from the server to the client
Upload	Sends an arbitrary file from the client to the server
Hollow	Starts a new process via process hollowing
Inject	Loads a dynamic link library by injecting code into a running process
Persistence_Registry	Sets up a persistence mechanism by creating a registry key
Persistence_Service	Achieves persistence by creating a service
Change_Interval	Changes the query interval for new commands
Persistence_DLL	Creates a persistence mechanism by dynamic link library search-order hijacking

The proof-of-concept implementation has adequate commands for typical malware behavior as well as for leaving the corresponding traces in the memory and drive images. Table 2 describes the commands that are currently implemented for malware synthesis. In the future, new commands will be introduced to increase the functionality of the malware component.

Additional traces are created by droppers depending on the exact scenario configuration. For example, it is possible to download the main malware component at one time via a Microsoft Office macro or to download malware in multiple stages and then launch it by process hollowing. Depending on the configuration files, additional traces can be created in user accounts (e.g., user email accounts) as well as in the RAM and hard drive images during synthesis.

Communications and Encryption. Since a key requirement for the client and server components is to support multiple protocol types, the proof-of-concept implementation supports Base64 encoding and RC4 and AES encryption implemented via Microsoft's CryptoAPI in addition to the plaintext mode. The different protocol types enable a framework user to map different levels of difficulty to a scenario that would become apparent during subsequent forensic analysis.

The communications include client requests for commands, server commands in response and client notifications of command execution results. All encoding and encryption types can be used via HTTP as

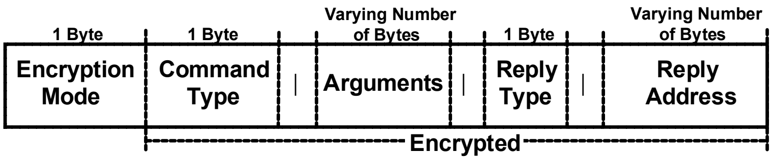

Figure 5. Malware control command structure.

well as DNS. In the case of HTTP, the initial request is made by a GET request; all subsequent requests are made by POST requests where the POST data contains the results. Communications via DNS use TXT records and domain names. The DNS server is implemented to respond to each request strictly according to the configuration file. When the client makes a request, a corresponding DNS response is created that contains a TXT record with the next command. The message about the execution status is specified by the domain name in the next request. Due to the protocol limitations, it may be necessary to append multiple TXT records [24]. In any case, regardless of the chosen protocol, all the commands have the form shown in Figure 5.

A separate protocol and encryption type can be chosen for each command. For example, it is possible to retrieve the first command via HTTP in plaintext and the second command Base64-encoded via DNS. In the case of HTTP, files for the upload and download commands are transferred via POST requests. In the case of DNS, a requested file is opened on the server side via the configuration file. It is then transferred to the client via TXT records. Again, the limitations of the DNS protocol must be considered, so it may be necessary to send more than one request per file. If a file is to be transferred from the client to the server, even more requests are necessary. The file is transferred in the requested domain name.

To avoid invalid characters, the HTTP and DNS protocols transfer files exclusively using Base64 encoding. This limits file uploads via DNS to about 40 bytes per request. However, normal Base64 is not used to avoid invalid characters. Since domain names do not support "=" and multiple dots, no trailing characters are introduced for padding purposes in an encoding; instead, it is adapted to the length during decoding. For the same reason, all the data encrypted using RC4 or AES is encoded with the same special Base64 variant.

Static keys exist for AES and DNS communications. Depending on the chosen configuration, the key can be extracted during subsequent forensic analysis from the executable located in the RAM dump, from the PCAP file or from the hard drive.

Notification and Validation. Notification of success or failure is done via POST data for HTTP and requested domain for DNS. In the case of HTTP, only the string `success` or `failed` is sent. In the case of DNS, a simple request is made to `SUCCESS.com` or `FAILED.com`. The success or failure message is then output via `stdout` so that the user of the synthesis framework is informed about the status of the generated image. Furthermore, the `hystck` reporting function is used to inform the user of the synthesis framework about all the automated user actions performed and, thus, about the underlying ground truth in the output images.

5. Malware Synthesis Module Integration

The `hystck` framework makes it possible to create scenarios via test scripts and via its generator component [12]. Test scripts have the advantage that they are precisely configurable as Python scripts and can, therefore, also generate arbitrary scenarios that are not covered by the generator component. The generator, on the other hand, enables scenarios to be configured using YAML files, which allows less technical users to create scenarios. The malware synthesis extension supports image generation using test scripts and the generator component.

5.1 Data Synthesis Using Test Scripts

In order not to block the main thread and to enable data synthesis, the login to the service virtual machine and the start of the server component should be executed in separate threads. The server component can then be started with the configuration file via SSH as shown in Figure 6. For simplicity, the configuration file is already present in the server in the example workflow. However, it is also possible to transfer the configuration file to the server via the SSH connection before malware synthesis starts. The thread in which the server component is started does not end until the server component is terminated.

To better understand the workflow in Figure 6, it is necessary to clarify selected persistence mechanism search-order hijacking. In the passed configuration file, the malware is requested to download a dynamic link library suitable for persistence and to set up a persistence mechanism in the client. The command-and-control server terminates automatically, which ends the thread. The main thread waits for this thread to terminate and then shuts down the virtual machine to test the persistence mechanism. Before restarting the virtual machine, the server component of the malware is restarted in a thread using the same function. This time, however, there is a different configuration file with different control

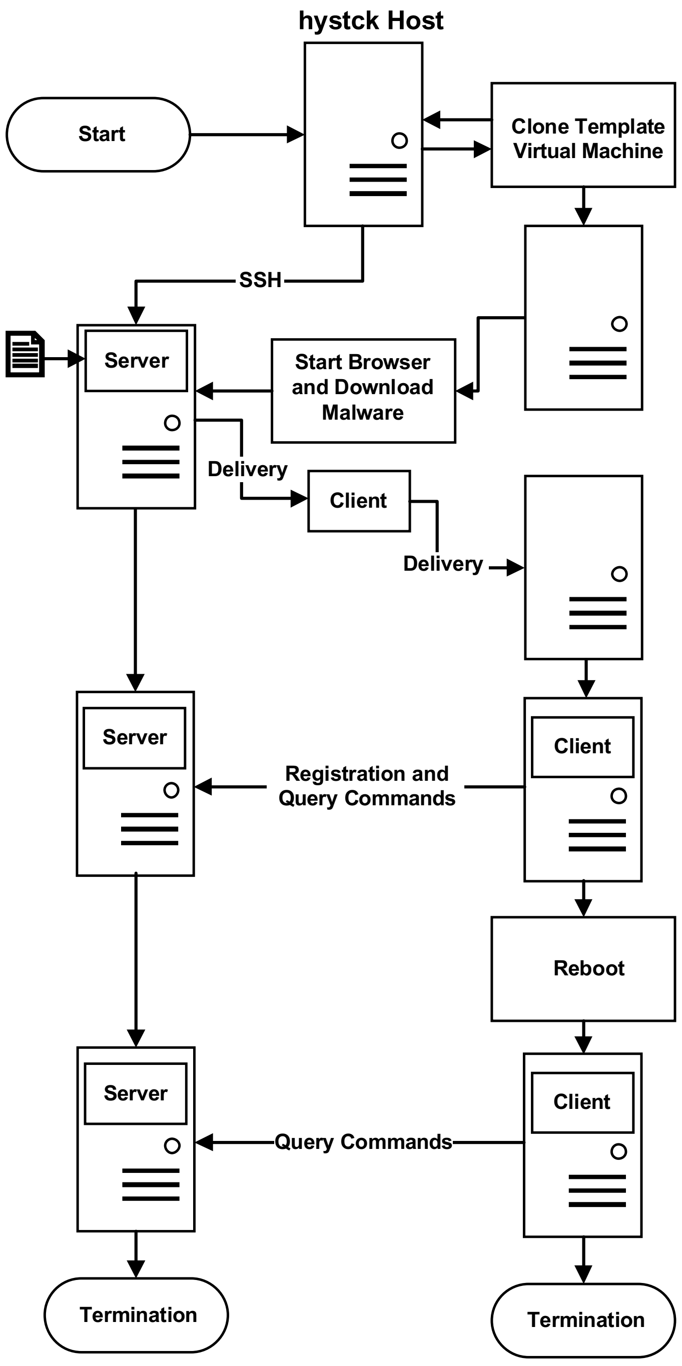

Figure 6. Data synthesis workflow using test scripts.

commands. Thus, if the malware is started again on the client due to the persistence mechanism, it connects to the server and executes the commands stored in the configuration file and terminates.

5.2 Data Synthesis Using the Generator

Aside from the test scripts that have to be programmed manually, the generator is the most important component of the `hystck` framework. The generator enables a user with limited technical skills to easily configure the data synthesis process using a YAML file, automatically generating a large number of traces rapidly and with relatively little effort. This is only partially the case with test scripts because they sometimes require several hundred lines of Python code to achieve the same functionality. In addition to the actual commands executed on the client, the YAML configuration file specifies information such as the service virtual machine (command-and-control server) address, email to be synthesized (e.g., for a phishing scenario), exact download paths and ports.

An important point has to be considered to integrate malware functionality in the generator. During the normal execution of the generator, the order of actions to be performed is shuffled randomly before execution, so it is not known which functions have completed and which have not. While this is not a problem when accessing a large number of websites, it can be problematic to obtain the correct malware component functionality. For example, before each scenario, it is necessary to ensure that malware has been downloaded or that a persistence mechanism has been created. Otherwise, an attempt could be made to execute the malware before it is delivered to the target system. The configuration settings passed to the YAML file are, therefore, only executed on restart if a persistence mechanism was previously set up or after the malware was downloaded. Figure 7 shows the data synthesis workflow using the generator.

The configuration files passed in a YAML file look very similar to the configuration files of the server component. Only special commands are added, for example, to trigger a restart. Within the YAML file, new collection types for the configuration files and application types are added for the malware component. This makes it possible to configure various parameters such as server IP addresses and download paths. Figure 8 shows an example YAML file.

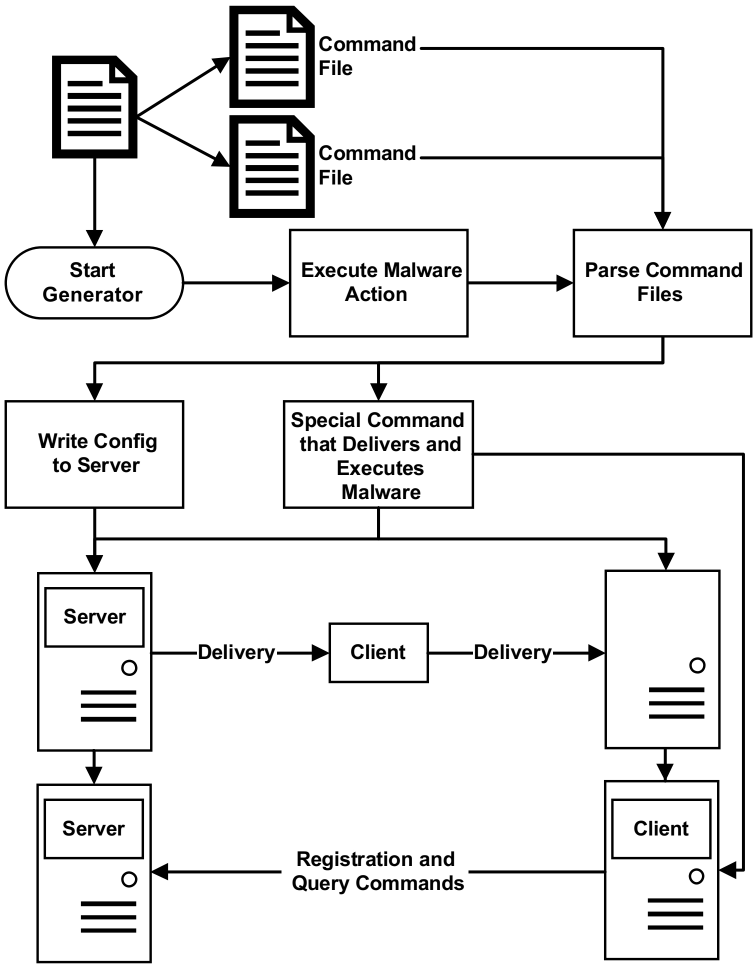

Figure 7. Data synthesis workflow using the generator.

6. Generated Dataset Validation

The dataset generated via data synthesis was validated by examining volatile RAM dumps, PCAP network traffic files and persistent drive images. Multiple evaluation phases were performed that focused on where and how traces could be found during a forensic investigation. This proved the existence of traces after data synthesis and, thus, validated the correct functioning of the proof-of-concept implementation.

RAM Dump. To find relevant traces in a RAM dump, it is necessary to first locate the malicious process. The traces differ depending on

```
name: hystck example
author: Lukner, Goebel, Baier
collections:
  c-malware-1:
    type:         malware
    commands:     /data/hystck/examples/generator/
                    collections/malware1.txt
    email:        /data/hystck/examples/data/
                    email_hay.xml
applications:
  malware-1:
    type:         malware
    service-vm:   192.168.122.219
    name:         MalwareBot.exe
    dnsServer:    192.168.122.219
    webServer:    192.168.122.219
    webPort:      7777
    service-port: 8080
    beacon:       3
    path:         C:\users\hystck\Desktop
hay:
needles:
  h-malware-2:
    application:  malware-1
    collection:   c-malware-1
```

Figure 8. Example YAML file.

when the RAM dump was created and which commands were executed. It is important to record handles and open network connections that are only valid during command execution. It can be difficult to acquire data from RAM at exactly the right time. Therefore, the query interval was extended and an automated RAM capture feature was incorporated in `hystck`.

Depending on the executed commands, a suspicious process can be detected by various characteristics in a RAM dump, such as open network connections and the process list. Depending on the chosen configuration of the malware synthesizer, certain anomalies may occur. First, an investigation could determine that an unknown process was started by dumping the process list regardless of the process name. Second, the process may appear in the open network connections linked to a UDP connection and to a TCP connection. A UDP connection could be suspicious because it would not belong to the DNS server in the system and a TCP connection could look suspicious because of the port used. Third, strings such as `http://192.168.122.219:8080/MalwareBot.exe` that provide information about the process origin would be discovered. After a malicious process is found, it can be extracted from memory using the Volatility tool and analyzed to discover its true functionality. By reverse engineering the executable, even the keys used in communications could be recovered. Fourth, traces of persistence mechanisms can be found in the process list based on the associated parent processes and the paths of the loaded dynamic link libraries of the processes.

Since malware can execute arbitrary code using `cmd` commands and download additional code and files, any number of additional traces can be created in RAM during data synthesis. If the malware performs operations such as uploads and downloads, then it is possible that handles to the corresponding files would still be open in memory. This information enables conclusions to be drawn about the exfiltrated data and downloaded malware code.

When the Inject command is executed, memory is allocated in the target process and the name of the dynamic link library to be loaded is written to the memory area. With the help of the `malfind` plugin, which recognizes such memory areas based on page permissions, the dynamic link library injection could be found. Another code injection technique that is not detected by `malfind` is process hollowing, when a process is started in the suspended mode and its executable is replaced by another executable. However, this can be detected using another Volatility plugin that is available at GitHub [25].

Network Traffic. Depending on the scenario and encoding or encryption method used, network traffic can be easy or rather difficult to analyze. The traces that can be extracted from network traffic include exfiltrated data in addition to the control commands sent. An infection can be detected in several ways, including a running malicious process in RAM, network anomalies or traces on a hard drive.

Suspicious DNS behavior in the proof-of-concept implementation may involve a large number of DNS requests with many TXT records occurring during a file transfer, especially when the requests are directed to a server that is not set as the DNS server for the system. The associated IP address would be of considerable interest during further analysis. Depending on the selected configuration, the entire malicious executable could be extracted and analyzed or at least the amount of exfiltrated data could be estimated from the capture. By choosing static keys, it would be possible to recover the RC4 and AES encryption keys by reverse engineering and thus decrypt the transmitted commands and identify the exfiltrated data.

Persistent Drive Image. Depending on the chosen malware settings, malware would leave various traces in the filesystem, browser download history and user email program. The ability to execute arbitrary code using `hystck` functions enables a large number of traces to be created. Examples include hiding traces by setting file attributes or generating arbitrary email for synthesizing a supply chain attack. Furthermore, persistence mechanisms could create additional traces. These traces would be found in the filesystem, for example, when using dynamic link library search-order hijacking or in the Windows Registry (e.g., a key under `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`) or due to execution via an autostart service.

Reporting. The malware components on the client and server output information via `stdout` for each executed command. In the case of the malware client, this is more for testing purposes because the output is not sent to the server; instead, the server is only informed whether or not the action was successful. This avoids additional network traffic, which should be kept to a minimum in the case of malware.

During automated execution, the server output is available to a framework user on the console during runtime and in a `hystck` report. This includes information about the success or failure of the last operation, beacon data and the next command sent to the malware. It would not be possible to validate the traces retroactively because implementing a command to execute arbitrary `cmd` commands would produce almost any number of traces. As a result, it would not be possible to validate the traces on the fly. However, in most cases, it can be assumed that the desired traces are present when the server receives feedback that execution was successful.

7. Conclusions

The extension of the `hystck` data synthesis framework developed in this research generates coherent malware traces for Windows operating systems at the RAM, network and hard drive levels. This enables the creation of realistic corpora that are needed to train digital forensic professionals on the use of forensic tools and to update their skills.

The new malware module integrated in `hystck` works with the generator component as well as test scripts. The new module makes it possible to create targeted malware scenarios as well as combine new malware features with existing `hystck` components. The server and client malware components are configured using JSON-like configuration files that contain large sets of implemented commands that generate malware traces.

The extended `hystck` data synthesis framework was validated using test scenarios for which coherent malware traces were created in RAM dumps, PCAP files and hard drive images. The validation reveals that the malware module is successfully integrated in the `hystck` framework and that configured traces exist at the RAM, network and hard drive levels to provide coherent pictures of malware infections. The framework incorporates a report module that records all the actions performed during data synthesis and, thus, all the generated artifacts. The resulting report provides the labeled ground truth and enables users to get a holistic picture of all the relevant traces existing in the generated RAM, network and hard drive images.

Future research will extend the data synthesis framework with additional malware-specific commands, network protocols, encryption types and persistence mechanisms to provide more configurable options for the data synthesis process, including obfuscating malicious control traffic and providing a variety of persistence techniques. In some places, constants are already reserved for extensions such as reserved flags for using raw TCP and providing full IPv6 support.

A limitation of the current framework is that the only dropper available is a Microsoft Office macro. Therefore, future work will implement additional droppers to represent alternative infection vectors, including a PDF document dropper.

Due to problems arising from the connectionless nature of the DNS protocol, it will be necessary to include sequence numbers in DNS communications to prevent file transfers from failing and transmitted commands from being lost. Additionally, the current implementation only uses the standard DNS port 53, which will be expanded.

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] I. Baggili and F. Breitinger, Data sources for advancing cyber forensics: What the social world has to offer, *Proceedings of the AAAI Spring Symposia – Sociotechnical Behavior Mining: From Data to Decisions?* pp. 6–9, 2015.

[3] Biometrics and Information Security Group (`dasec`), `hystck-malware-module`, GitHub (`github.com/dasec/hystck-malware-module`), 2022.

[4] 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.

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

[6] B. Carrier, Digital Forensics Tool Testing Images (`www.dftt.sourceforge.net`), 2010.

[7] R. Cole, A. Moore, G. Stark and B. Stancill, STOMP 2 DIS: Brilliance in the (visual) basics, Mandiant, Reston, Virginia (`www.mandiant.com/resources/stomp-2-dis-brilliance-in-the-visual-basics`), February 5, 2020.

[8] 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.

[9] Digital Corpora, Home (`www.digitalcorpora.org`), 2021.

[10] X. Du, C. Hargreaves, J. Sheppard and M. Scanlon, TraceGen: User activity emulation for digital forensic test image generation, *Digital Investigation*, vol. 38(S), article no. 301133, 2021.

[11] S. Garfinkel, P. Farrell, V. Roussev and G. Dinolt, Bringing science to digital forensics with standardized forensic corpora, *Digital Investigation*, vol. 6(S), pp. S2–S11, 2009.

[12] T. Göbel, T. Schäfer, J. Hachenberger, J. Türr and H. Baier, A novel approach for generating synthetic datasets for digital forensics, in *Advances in Digital Forensics XVI*, G. Peterson and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 73–93, 2020.

[13] 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.

[14] A. Hadi, Digital Forensic Challenge Images (Datasets), Champlain College, Burlington, Vermont (`www.ashemery.com/dfir.html`), 2011.

[15] N. Harbour, Flare-On 7 challenge solutions, Mandiant, Reston, Virginia (`www.mandiant.com/resources/flare-7-challenge-solutions`), October 23, 2020.

[16] S. Hegt, Evil Clippy: MS Office maldoc assistant, *Outflank Blog*, Amsterdam, The Netherlands (`www.outflank.nl/blog/2019/05/05/evil-clippy-ms-office-maldoc-assistant`), May 5, 2019.

[17] J. Huang, A. Yasinsac and P. Hayes, Knowledge sharing and reuse in digital forensics, *Proceedings of the Fifth IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering*, pp. 73–78, 2010.

[18] D. Lillis, B. Becker, T. O'Sullivan and M. Scanlon, Current challenges and future research areas for digital forensic investigations, *Proceedings of the Eleventh Annual Conference on Digital Forensics, Security and Law*, 2016.

[19] J. Liu, Ten-year synthesis review: A baccalaureate program in computer forensics, *Proceedings of the Seventeenth Annual Conference on Information Technology Education and the Fifth Annual Conference on Research in Information Technology*, pp. 121–126, 2016.

[20] M. McMahon and Contributors, What is `pywinauto`? (`pywinauto.readthedocs.io/en/latest`), 2018.

[21] MITRE Corporation, Caldera, GitHub (`github.com/mitre/caldera`), 2021.

[22] 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.

[23] 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.

[24] P. Mockapetris, Domain Names – Implementation and Specification, RFC 1035, 1987.

[25] `monnappa22`, HollowFind, GitHub (`github.com/monnappa22/HollowFind`), 2016.

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

[27] Outflank, Evil Clippy, GitHub (`github.com/outflanknl/EvilClippy`), 2021.

[28] Quarkslab, LIEF Project, GitHub (`github.com/lief-project/LIEF`), 2022.

[29] 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.

[30] Statista, Operating systems most affected by malware as of 1st quarter 2020, New York (`www.statista.com/statistics/680943/malware-os-distribution`), April 11, 2022.

[31] The Honeynet Project, Challenges (`www.honeynet.org/challenges`), 2022.

[32] H. Visti, ForGe, Forensic Test Image Generator, GitHub (`github.com/hannuvisti/forge`), 2015.

[33] 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.

[34] K. Woods, C. Lee, S. Garfinkel, D. Dittrich, A. Russell and K. Kearton, Creating realistic corpora for security and forensic education, *Proceedings of the Sixth Annual Conference on Digital Forensics, Security and Law*, 2011.

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