Improving trace synthesis by utilizing computer vision for user action emulation

Schmidt, L.; Kortmann, S.; Hupperich, T.

DOI
10.1016/j.fsidi.2023.301557

Publication date
2023

Document Version
Final published version

Published in
Forensic Science International: Digital Investigation

Forensic Science International: Digital Investigation 45 (2023) 301557

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS 2023 USA - Proceedings of the Twenty Third Annual DFRWS Conference

Improving trace synthesis by utilizing computer vision for user action emulation

Lukas Schmidt [a, *], Sebastian Kortmann [b], Thomas Hupperich [a]

[a] University of Münster, Germany
[b] University of Amsterdam, the Netherlands

ARTICLE INFO

Article history:

Keywords:
Trace synthesis
Data synthesis
User emulation
Forensic dataset
Computer vision

ABSTRACT

Forensic analyses are performed by skilled forensic practitioners who require reliable, state-of-the-art tooling and ongoing training. To provide both, education and academia rely on realistic training datasets. Those datasets are crucial to teaching investigators, validating forensic tools, advancing algorithms, and pursuing research. At the same time, the forensic community faces a shortcoming of realistic datasets, mainly due to ethical and legal reasons. To overcome this challenge, prior work introduced several frameworks aiming to create unproblematic replications of real evidence. Those frameworks generate synthetic datasets by populating disk images with traces of emulated user behavior. However, it is general consent that existing frameworks have some drawbacks concerning the quality of generated datasets, particularly due to the incorporation of unrealistic traces in GUI-based environments. Reviewing the implementation details of common frameworks, we found that current solutions miss realistic trace synthesis, reducing the quality and usefulness of synthesized datasets.

By leveraging computer vision, this paper introduces a novel approach aiming to enhance the quality of synthetic datasets. We propose an architecture and provide an open-source implementation utilizing a hypervisor for creating Human Interface Device (HID) input, which is controlled by computer vision algorithms to imitate human-like user actions. In this way, we provide external GUI automation capabilities that enable more realistic trace synthesis than existing solutions and open up the applicability to a wide range of GUI-based operating systems. In contrast to previous research results, our approach is independent of software running in virtual machines, further optimizing the quality of generated datasets by omitting automation artifacts. Our experiments indicate that using external GUI automation for user action emulation results in a greater amount and a more widespread distribution of traces. Therefore our approach may refine the quality of datasets in this field.

1. Introduction

The quality of research outcomes, the successful education of forensic practitioners as well as the technological development of reliable, state-of-the-art tooling depends on how realistic underlying training datasets are. Focusing on applications in digital forensics, training datasets can be used to teach future investigators (Moch and Freiling, 2009), validate forensic tools, techniques and algorithms (Garfinkel et al., 2009) or advance machine learning models aiding in forensic investigations and forensic analysis (Du et al., 2020). When utilizing datasets in academia, the quality of datasets plays a crucial role, as outcomes are directly affected by the quality of underlying data (Abt and Baier, 2014; Grajeda et al., 2017; Garfinkel et al., 2009). At the same time, the forensic community suffers from limited availability of appropriate datasets. Also referred to as the dataset gap problem (Gonçalves et al., 2022), recent research highlights that available datasets are often of poor quality, containing outdated examples and unrealistic content.

A main cause of the dataset gap problem lies in ethical and legal concerns regarding the exchange of real evidence, also confined by insurmountable requirements for privacy protection due to the nature of forensic investigations. Notably, sharing of datasets may lead to copyright infringements, further narrowing down any exchange of datasets. With this in mind, the majority of datasets used in academia are held private (Grajeda et al., 2017; Abt and Baier,

* Corresponding author.
E-mail addresses: lukas.schmidt@wi.uni-muenster.de (L. Schmidt), s.kortmann@uva.nl (S. Kortmann), thomas.hupperich@wi.uni-muenster.de (T. Hupperich).

https://doi.org/10.1016/j.fsidi.2023.301557

Table 1
Trace generation techniques of existing frameworks.

Framework	Populated System	Trace Generation	Guest Agent	GUI Interaction
Forensig² (Moch and Freiling, 2009)	Linux	Shell	SSH-Server	–
ForGeOSI (Krüger, 2014)	Linux, Windows	Powershell, Shell	VirtualBox	–
ForGe (Visti et al., 2015)	NTFS Partition	Python	–	–
ForGen (Keighley, 2017)	Windows	Shell, Puppet	Puppet, VirtualBox	–
EviPlant (Scanlon et al., 2017)	Windows	–	–	–
VMPOP (Park, 2018)	Windows	Powershell	VirtualBox	–
hystck (Göbel et al., 2020)	Linux, Windows	Python	Hystck Agent	–
TraceGen (Du et al., 2021)	Windows	Python	VirtualBox	pywinauto (Win32-API)
ForTrace (Göbel et al., 2022)	Windows	Python, Powershell	ForTrace Agent	–

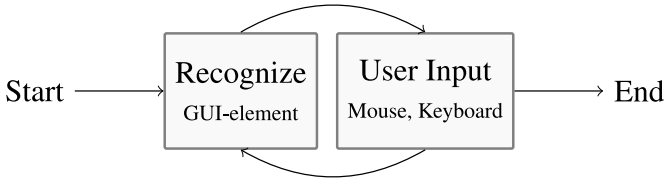


Fig. 1. User action loop.

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}} \quad (1)$$

Fig. 3. TM_CCOEFF_NORMED - OpenCV.

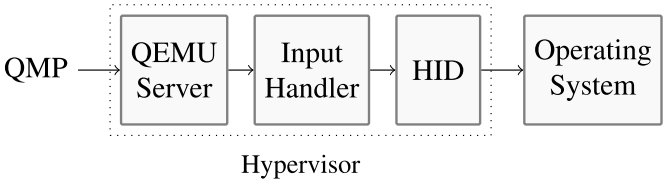
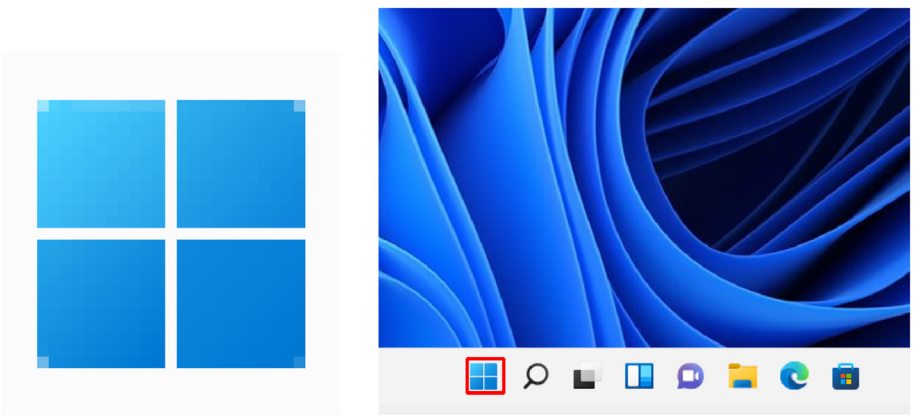


Fig. 4. Externally controlled user input.

2014), and even though some datasets containing real-world data exist (Garfinkel et al., 2009), access to these data sources is restricted. The shortcoming of realistic datasets negatively affects the whole field, hindering rapid research outcomes and only allowing inadequate scientific comparison of research results (Garfinkel, 2007; Abt and Baier, 2014). Prior work introduced unproblematic replication of real-world evidence as solution for the dataset gap problem (Garfinkel et al., 2009), synthesizing traces in disk images by manually conducting user actions. Considering that creation of synthetic datasets by hand is a resource- and time-intensive task, manual techniques evolved into various powerful frameworks aiming at automating and scaling the creation of realistic datasets (Moch and Freiling, 2009; Du et al., 2021; Göbel et al., 2022). Those frameworks usually emulate user actions in virtual machines to populate disk images with synthetic traces.

Although streamlining the creation process and lowering the costs for synthesizing datasets, it is general consent that the usage of those frameworks has some drawbacks concerning the quality of generated data, particularly due to incorporation of unrealistic traces (Moch and Freiling, 2009; Du et al., 2021). Unrealistic traces mainly originate from insufficient Graphical User Interface (GUI) automation capabilities, so that the emulation of user actions becomes inaccurate, creating fewer and less diverse traces compared to real-world data. Furthermore, all proposed frameworks rely on additional software to implement user action emulation. This software, also known as guest agent, offers required control mechanisms while running in the virtual machine. However, the usage of guest agents leads to installation artifacts and unwanted traces polluting the generated disk images, further lowering the quality of datasets.

By introducing a novel method for trace synthesis, we provide a way to incorporate truly realistic traces in artificially-created datasets. Optimizing the quality of synthetic datasets, we aim to positively affect future efforts in research, education and practice. More detailed, this work makes the following contributions:



(a) Home Button template. (b) Matched template in screenshot.

Fig. 2. Template matching with OpenCV.

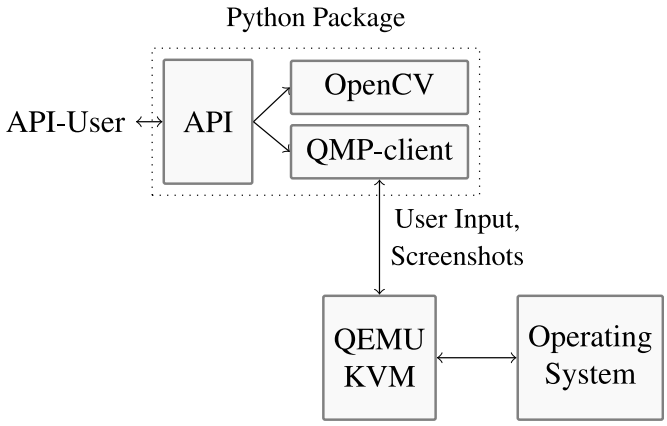


Fig. 5. Pyautoqemu.

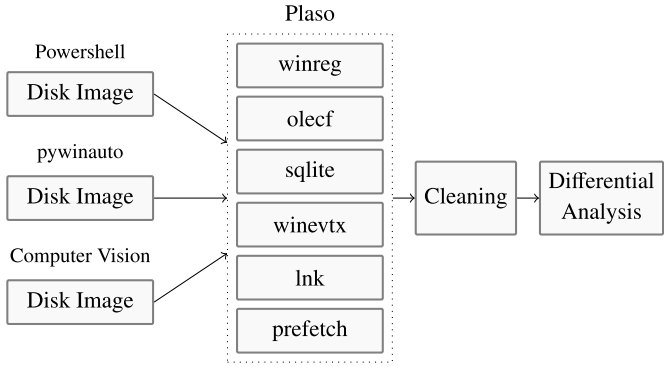


Fig. 6. Methodology - trace analysis.

- We develop an architecture and open-source framework for user action emulation using computer vision (Sec. 3), improving the synthesis of realistic traces. In contrast to existing approaches, our proposed solution provides hypervisor-assisted, externally controlled emulation of user actions and is therefore Operating System (OS) agnostic. Being independent of any additional software running in guest machines, our approach omits automation artifacts and allows for the emulation of human-like user activities.
- We show that our solution creates more realistic traces, measurable in a greater amount and a more widespread distribution of traces compared to common approaches (Sec. 5). The results strongly indicate that our approach improves the quality of synthetic datasets.
- To the best of our knowledge, our study is the first to compare the traces generated by commonly used emulation techniques, allowing to reason about their quality and closeness to reality. In so doing, we propose a novel method for evaluating trace-generation (Sec. 4), based on differential analysis of disk images using open-source forensic tooling (Garfinkel et al., 2012).
- Furthermore, we identified traces that may provide additional value in forensic investigations. We found traces pointing out to human-like GUI-interactions as well as program execution, whose content is currently missed by the parsers used for trace extraction (Sec. 5).

In the remainder of this article, we review related work, providing an overview about the state of the art in synthetic trace-generation and the demand for realistic datasets (Sec. 2). We evaluate our approach (Sec. 6) and finally draw a conclusion including an outline of future work (Sec. 7).

2. Related work

2.1. Missing realistic datasets

The field of digital forensics is missing a vital piece of scientific groundwork (Garfinkel et al., 2009): publicly available, standardized and realistic forensic datasets. A primary reason for the shortage of datasets is that sharing of real-world data is hampered due to ethical and legal reasons (Garfinkel, 2007; Garfinkel et al., 2009; Grajeda et al., 2017), further amplified by demands on privacy protection and the threat of possible copyright infringements when publishing datasets. Inhibiting the reproducibility of results and wasting valuable resources in obtaining custom datasets (Garfinkel, 2007; Woods et al., 2011), the absence of appropriate datasets adversely affects research and education efforts. Being a well known obstacle in forensic science, the shortcoming of available datasets is commonly referred to as the dataset gap problem (Garfinkel, 2007; Gonçalves et al., 2022).

In case real-world data is not available, the usage of realistic synthetic data is argued to be a valid alternative (Abt and Baier, 2014). When being representative of real-world evidence, synthetic datasets can be used as a substitute in forensic research, education and development. While leading the way to synthetic dataset creation, the aspect of realism is of fundamental importance, directly affecting the intended purpose. Thereby, realistic datasets can be seen as an indicator of "transferability of research results" (Abt and Baier, 2014). In a similar vein, Grajeda et al. (2017) emphasize the quality of datasets depend on their conformity with real-world data. Summarized, the use of realistic training data correlates with quality of research outcomes. Being of central importance for optimal results in teaching forensic practitioners, the need for realistic datasets "is difficult to overstate" also from training and educational perspective (Garfinkel et al., 2009). To achieve optimal outcomes, forensic practitioner education needs realistic data. The advent of machine learning and artificial intelligence in digital forensics further amplified the demand for realistic datasets. The reliability of pre-trained models depends on the quality of the training dataset, deciding over the usefulness in real world scenarios, which further establishes the relevance of realistic datasets (Du et al., 2020).

Realistic datasets should be representative of data found in real-world cases, ensuring tools and algorithms work in the field (Garfinkel, 2007). In the past, Woods et al. (2011) found existing data corpora to be "insufficiently realistic" for education and research purposes, defining the characteristics of realistic data corpora. Data should be of "realistic wear and depth", containing traces created by realistic user actions and usage patterns, introducing background noise with traces of typical user behaviour like application usage or web browsing. Furthermore, datasets can be considered to be realistic when being consistent, not only from a technical perspective, but also with regard to setting, fictitious interactions and linked background data, introducing another layer of realism. Distinguishing between the two layers of realism Woods et al. (2011) described, the *logical realism* defines the overall sense of a forensic scenario and may not always be technical, considering sociological and criminalistic facets. Furthermore, the *technical realism* results from the appearance of traces in computer systems. This work is about improving the technical realism of synthetic datasets, focusing on how realistic traces can be artificially created to support dataset generation.

2.2. Challenges of manual trace generation

Populating disk images with synthetic traces not containing any private or otherwise protected data is a widely accepted way to address the dataset gap problem in forensic science (Garfinkel, 2007). Trace synthesis can be carried out manually, but creating a plausible and consistent disk image this way is challenging. Due to replaying user actions by hand, the manual generation also requires carefully planning. Being complex and resource-intensive tasks, the manual generation binds valuable human time and effort (Woods et al., 2011).

Due to the effort required to replay actions by hand, synthetic datasets may suffer from suboptimal quality, e.g. missing realistic wear and tear like background noise (Göbel et al., 2022). Arguably, the amount of work needed to create a dataset may be why researchers stop updating and further diversifying datasets after their first release (Grajeda et al., 2017), resulting in a shortage of current datasets. This is why several frameworks for trace generation and disk image population evolved, aiming to ease and scale the dataset creation process through automation. Another advantage of synthetic trace generation is the ability to log actions for establishing ground-truth and labeling data.

2.3. State of the art in synthetic trace generation

Göbel et al. (2022) presented an overview of currently available dataset-synthesis frameworks. Evaluating the published approaches, the overall idea behind dataset-synthesis frameworks consists of populating virtual machines with synthetic traces. Synthetic traces are created by emulation of user actions, with emulation techniques differing in underlying technology, capabilities and supported operating systems.

Moch and Freiling (2009) introduced a framework named Forensig^2, producing traces inside Linux-based virtual machines executing shell commands over the SSH-protocol, whereas the order of execution is controlled by Python scripts.

The framework VMPOP (Park, 2018) uses the VirtualBox hypervisor, virtualizing Windows machines with preinstalled VirtualBox Guest Additions. Consisting of a software agent running in the guest, the Guest Additions offer advanced control mechanisms, which VMPOP utilizes to execute Powershell scripts emulating user actions.

In a similar manner, the TraceGen framework (Du et al., 2021) also sets on VirtualBox to run and control guests. Optimizing trace synthesis capabilities, a Python interpreter is installed inside guest machines, which can be used for executing custom Python scripts. Utilizing the Win32-API with the pywinauto Python package, TraceGen is able to create traces by emulating user actions using the GUI.

The Hystck framework (Göbel et al., 2020) builds on the KVM-hypervisor to run virtual machines, implementing a custom agent written in Python to control the guests. This agent called "Interaction Manager" allows for executing arbitrary Python scripts responsible for trace generation, which opens the possibility to write plugins and use the rich Python ecosystem in the generation process. To date agents for Windows and Linux are available.

ForTrace (Göbel et al., 2022) is a Python3 rewrite of the Hystck framework, mainly focusing on Windows 10 guests. At the core, ForTrace is producing traces the same way as Hystck; guest Machines run a custom Python agent, able to generate traces through the execution of Python scripts. Explicitly mentioned is the possibility of executing Powershell scripts for more versatility in trace creation.

Further listed by Göbel et al. (2022) are two data synthesis tools not backed by peer-reviewed papers. Similar to VMPOP and TraceGen, the tool ForGeOSI (Krüger, 2014) runs virtual machines with VirtualBox, employing the VirtualBox guest additions to control machines and processes inside the guest. Traces are created by Powershell or Shell command execution in Windows and Linux environments. The second tool mentioned is named ForGen (Keighley, 2017), using Puppet to provision virtual machines executed with VirtualBox. The open-sourced code of ForGen offers two providers for trace generation, the Puppet Agent and Shell commands. To date, only software package installation modules built on Puppet are publicly available.

While also mentioned by Göbel et al. (2022), the ForGe framework and the EviPlant approach do not implement trace synthesis comparable to the other frameworks. ForGe (Visti et al., 2015) is a framework for creating NTFS disk images, intended for researching data-hiding techniques. Disk images are constructed by mounting NTFS partitions in a Linux host system, then using Python for distributing files and applying data hiding techniques in the mounted filesystem. Rather than generating traces, the framework EviPlant (Scanlon et al., 2017) provides mechanisms for extracting already created traces and injecting these trace packs into running virtual machines. This allows for an easier exchange of forensic datasets, but does not aid in trace synthesis.

3. Emulation of user actions using computer vision

Reviewing the implementation details of already published trace-generation frameworks (Table 1), we come to the conclusion that none of the solutions is able to reproduce perfectly realistic traces in GUI-based operating systems. Only one of the published frameworks can control the operating system GUI (Du et al., 2021), and even though GUI control capabilities can be considered as a big step forward, the chosen proof-of-concept implementation utilizing the Win32-API has its limits. As an example, Windows provides multiple ways to create GUI-elements, but the Win32-backend can only control a subset of them. Hence we infer that existing emulation techniques can only partly reproduce real-world user actions, which we expect to result in unrealistic traces.

Furthermore, existing frameworks rely on guest agents to emulate user actions (Table 1). With great abstraction, user agents consist of software running inside virtual guest machines. These agents execute small programs mimicking user actions, thus generating the desired traces inside the guest. After finishing the trace population of the guest, the disk images of the guest machines can be used as training data. In context of forensic dataset synthesis, the usage of guest agents has some disadvantages. The installation of guest agents pollutes the resulting disk images with additional software artifacts, while interactions with guest agents lead to even more unwanted traces, like network traces or script execution artifacts. Another point to consider is, that the usage of guest agents raises the bar for supporting multiple operating systems. Supporting an operating system implies a customized guest agent with appropriate trace generation features, which may result in a lot of implementation work. Therefore trace generation frameworks mainly focus on Microsoft Windows, some on Linux. To date, other sytems like MacOS or mobile operating systems are rather ignored, also mirroring in availability of corresponding datasets (Gonçalves et al., 2022).

In the following, we propose a solution to overcome the mentioned disadvantages through hypervisor-assisted, externally controlled user input generation, combined with computer vision algorithms for GUI-automation.

3.1. Design choices

Daily users usually interact with computer systems through the

GUI. So traces created through GUI actions occur commonly and can be considered to be of great significance for creating realistic datasets (Moch and Freiling, 2009; Du et al., 2021). At the same time, the artificial creation of consistent GUI traces is hard to achieve (Du et al., 2021), caused by the fact that GUI interactions and side effects create consistent traces in various parts of the operating system. Replicating consistent traces spread to multiple locations such as registry hives, plain files and event logs is a challenge. In particular, because manifestation of traces is seldom known in advance.

So instead of developing a replication mechanism for side effects and corresponding traces, this study presents a GUI-automation solution that can mimic real-world user behavior, generating perfectly realistic traces.

Our solution focuses on offering improved trace synthesis, ready for integration in existing dataset-synthesis frameworks. As those frameworks are commonly implemented in Python, we provide a Python package to further facilitate integration.

In general, our ambition is to provide an OS-independent solution and omit the usage of guest agents. We therefore rely on the QEMU/KVM-hypervisor to run virtual machines, which is customizable and allows fine-grained control using the Qemu Machine Protocol (QMP).

```
# connect to vm
vm = VM.connect(qemu_ip, qemu_port)
# start browser
vm.cv_find("browser_icon.png")
vm.leftclick(x,y)
```

Snippet 1: Start a browser using GUI-automation.

3.2. Concepts of user action emulation

Human behavior is a loose concatenation of several human actions. An action can be self-contained, e.g. a click on an icon, or a compound action consisting of several sub-actions (Du et al., 2021). Examples include a user opening a website, entering some search string, and finally conducting a browser session.

GUI-based human–computer interaction can be seen as a loop iterating between two steps (Fig. 1). First, a user recognizes a GUI element, which signals some state or functionality. In response to GUI-element identification, the user performs certain input to drive the user action further, e.g. click a button or enter text. Finally, the user identifies the next GUI element, repeating the described behavior until a (compound) action is finished.

With our solution, we aim to replicate human behavior using software. To achieve this, we provide GUI-element identification using computer vision algorithms combined with external creation and control of user input utilizing the hypervisor. That allows for a virtual replication of user action loops and artificially mimicking real-world human behavior, without creating any additional traces inside guest machines.

3.3. GUI-element recognition

Existing GUI-automation solutions can be loosely divided into two categories. On the one hand, tools that allow for controlling user input using the OS Application Programming Interface (API). On the other more capable frameworks incorporating GUI-element recognition and scriptable execution of subsequent actions. A prominent example of the second category is SikuliX (Yeh et al., 2009), which uses OpenCV to identify GUI-elements in screenshots of the running system, followed by the execution of predefined actions. While having clear advantages in functionality, SikuliX runs locally, relying on OS-APIs to control user input. Concerning forensic dataset creation, using solutions such as SikuliX would lead to the creation of unwanted additional traces. Further constraints can be encountered when creating complex datasets with multiple systems running in parallel or when populating systems over a long period of time, which may prove useful for future use cases of GUI-automation in digital forensics and cyber security.

This leads us to develop a headless system, a system independent of graphical interfaces. Thus, our solution can run on servers without desktop environment. To the best of our knowledge, we did not find any evidence of an existing, comparable solution.

Employing the same computer vision algorithms used by SikuliX, we use the template matching functionality provided by the OpenCV framework to identify GUI-elements in screenshots. Template matching is used to determine the position of a smaller image, called *template*, inside a larger image (Fig. 2). The matching procedure subsequently moves the template, calculating the similarity between the template and the given image section for every move.

Given that a minimum similarity is reached by at least one image section, the area with the maximum similarity score can be considered equal to the template. We use the *TM_CCOEFF_NORMED* algorithm[1] for template detection (Fig. 3), as we found this algorithm to be sufficiently fast and reliable to emulate human behavior.

3.4. External emulation of user input

One of the key challenges of trace synthesis frameworks is to omit traces of itself (Göbel et al., 2022). To avoid any traces of automation, we chose to generate user input externally, controlled and emitted from outside of the guest virtual machine. Despite being a generally favored strategy, also referred to as the *gold standard* (Du et al., 2021) for avoiding automation artifacts, we are not aware of any implementation. So to the best of our knowledge, our solution is the first enabling external GUI-automation.

We incorporated a QMP client, which runs on the host machine and can communicate with the QEMU-/KVM-hypervisor to use the QEMU API (Fig. 4). In this way, we can instruct QEMU to generate input events of virtualized devices, which results in corresponding updates of the memory-queues of the emulated HID-USB devices. Thus controlling attached devices is completely transparent from within the virtual guest machine. As soon as the input events are successfully written into the HID-device memory regions, the OS driver recognizes input from the emulated HID-device similar to every other HID-device. As a result, no guest agent is needed to control user input events.

3.5. Combining pieces in the pyautoqemu-package

In order to facilitate the use of the introduced functionality, we provide a Python package named *pyautoqemu*. The *pyautoqemu*-package wraps our low-level implementation in easy-to-use high-level APIs, lowering entry barriers and allowing to reuse the package in dataset-generation frameworks (Fig. 5). To control the user interface, our API offers two functions using template matching to identify and determine the coordinates of a chosen GUI element in a screenshot of the corresponding virtual machine's desktop.

[1] https://docs.opencv.org/4.x/df/dfb/groupGUIautomationect.html.

cv_find(template_path) This function returns the coordinates of the template in a screenshot of the virtual machine's desktop, which is useful for determining the coordinates of a GUI-element.
cv_wait(template_path, seconds) For the provided timespan in seconds, this function searches for the template to appear on the virtual machine's desktop, which is useful for waiting for a state or event to appear.

Combined with high-level functions for generating user input, e.g. leftclick(x,y), doubleclick(x,y) and send_keys(keys), the overall workflow using these functions is rather simple. We can identify GUI-elements by providing a template, then calling one of the GUI-recognition functions. Conditional to the output, we can control the GUI in conducting appropriate user input emulation. At this point, it is possible to virtually replicate user action loops with *pyautoqemu*, finally allowing to emulate realistic human behavior.

```
# opens notepad
vm.start_with_gui("notepad")
# input file content
vm.send_text(content)
# file menu
x,y = vm.cv_find("/cv_gfx/notepad-file.png")
vm.leftclick(x,y)
# save as
x,y = vm.cv_find("/cv_gfx/notepad-saveas.png")
vm.leftclick(x,y)
# choose filename
x,y = vm.cv_find("/cv_gfx/notepad-filename.png")
vm.leftclick(x,y)
vm.doubleclick(x,y)
vm.send_text(file)
# save
x,y = vm.cv_find("/cv_gfx/notepad-save.png")
vm.leftclick(x,y)
# close notepad
x,y = vm.cv_find("/cv_gfx/notepad-close.png")
vm.leftclick(x,y)
```

Snippet 2: Emulating file creation with Notepad.

4. Synthesizing traces

To evaluate our Computer Vision approach, we want to compare the created traces with those generated by other emulation techniques. The performed literature and source code review (Sec. 2) shows that user action emulation is mostly performed using Powershell or Python. The TraceGen framework (Du et al., 2021) stands out, as it is the only framework capable of emulating GUI-actions. It does so using Python and the *pywinauto*-package with Win32-backend. So we decided to emulate user actions with Powershell and Python, and where applicable use the same GUI-automation like TraceGen. To maximize comparability, we execute the same imaginary user scenario for each emulation technique:

1. Implement the user actions necessary for scenario execution.
2. Populate disk images in executing the user scenario.
3. Extract traces.

4.1. Execution environment

Each emulation variant is used to execute the same imaginary scenario, running in a virtual machine based on the same initial disk image. This disk image contains a system running the Windows 11 OS, which we already configured and also comes with necessary software preinstalled. Regardless of whether the emulation technique needs all software, the disk image contains the same software for all approaches.

We configured useful settings to ease automation, like AutoLogon, disabling Windows Update or deactivating screensavers. This ensures virtual machines come up flawlessly each boot cycle.

Using Powershell or Python emulation, we need to initiate and control the emulation of user actions externally. Being a lightweight alternative to guest agents of VirtualBox, we choose an SSH server in the guest machine as the control instance, which can transfer data to the guest and execute programs within the guest. Worth to be mentioned SSH servers have no access to the GUI of the Windows OS, so without adjustments *pywinauto* can not be used to control the GUI. As a workaround, we call the tool *psexec* to run the Python interpreter over SSH, allowing more privileged execution of Python scripts and access to the graphical environment. We initially set the default shell to Powershell, so switching the shells for every SSH command can be avoided.

4.2. Emulated user actions & scenario

Due to the significant amount of possible user actions, we can only implement a representative subset of actions to reason about the applicability of our approach and allow a comparison of the three emulation variants. Actions should preferably be compound user actions (Du et al., 2021), which consist of multiple steps, different complexity and GUI interactions. We construct an exemplary imaginary scenario, emulating typical user actions whose traces are interesting to evaluate. The following enumeration describes the scenario and the high-level implementation.

1. Search session using the Edge browser, searching the term *largest financial institutions* on duckduckgo.com.
2. The url https://sqlmap.org is opened using the Edge browser.
3. Download of the file https://github.com/sqlmapproject/sqlmap/zipball/master.
4. A text file with suspicious content is created using Notepad.
5. The downloaded file sqlmap.zip gets unzipped.
6. The unzipped tool *sqlmap* gets executed via the CLI using the destination IP *127.0.0.1*.

Technical constraints sometimes led to small differences in implementation and the resulting emulated behaviour. As these deviations are relevant for later trace evaluation, we shortly describe these differences. In recently published research, user action emulation with Powershell is based on the execution of Powershell built-in commands and scripts; therefore no GUI-behaviour emulation takes place. While we tried to automate GUI-actions when using Python, this was not always possible. In contrast to our Computer Vision approach, we found the Win32-backend of *pywinauto* had no way to control the Edge browser. This is why we chose to download files via the Python standard library. Similarly, unpacking archives using the Windows Explorer was straightforward using Computer Vision, but rather complex using *pywinauto*. So we also used the standard library to extract archives, resulting in different traces.

We execute the scenario in a virtual machine. For each emulation technique, we initially reset the used virtual machine to the snapshot of the base image, then use the chosen emulation variant to execute the scenario and save the populated disk image for later evaluation.

5. Results: Trace Analysis

Evaluating which traces are the most relevant traces to analyze and compare, we came to the conclusion that it does not provide

much value to analyze evident additional traces like installation artifacts of a Python interpreter or other software packages. Instead, we focus on evaluation of traces which can considered to be realistic side effects of user actions, like entries in registry hives, event logs or prefetch files. To extract traces we use Plaso,[2] which is a Python-based engine used to create timelines based on traces found in a computer system, for example log files taken from a disk image. Plaso comes with parsers for many publicly known files and formats. Apart from usage in incident response and forensic investigations, Plaso has been used in previous scientific publications for trace evaluation (Du et al., 2021), which is why we also choose Plaso. Using Plaso, we explore the following file types in the created disk images, parse them and store the extracted traces in plaintext format for later analysis: Windows registry hives (**winreg**), Object Linking and Embedding Compound File format files (**olecf**), sqlite databases (**sqlite**), Windows Event Log files (**winevtx**), link files (**lnk**) and prefetch files (**prefetch**).

5.1. Methodology

To statistically explore the synthesized traces, we conduct a differential analysis of the previously extracted traces using Python and Pandas (Fig. 6).[3] A differential analysis is a well known research instrument in digital forensics (Garfinkel et al., 2012). which allows to compare feature sets in computing the feature delta, showing any changes between the features. We consider the traces created through emulated user actions to be our features and the sets of extracted traces to be our feature sets.

Preparing the data for analysis, we clean the trace sets using the following steps:

1. Removal of traces without timestamps in every trace set.
2. Removal of duplicate entries equal in timestamp and content for each trace set.
3. Generate the union of all trace sets, then remove all duplicate entries in the union.

Cleaning the data in this way ensures all equal traces spread over the trace sets are cleared, so we can focus on analyzing traces which are unique for each emulation approach. As a lot of the extracted traces contain dynamic content like hash values, time stamps or user IDs, direct comparison is not always feasible. So we normalize the data in extracting unique identifiers unambiguously identifying traces and corresponding events for each category, like tuples of event id and logsource in the Windows Event Log. To reason about the quality of traces our computer vision approach creates, we analyze the traces synthesized by each emulation approach with regard to amount and distribution of traces as well as automation artifacts. Furthermore, we point out which traces are only created using computer vision.

5.2. Overview

Initially, we compare the amount of extracted traces split by file format and trace set. The usage of GUI-automation leads to an increase of created traces (Table 2), where the computer vision approach creates even more traces compared to GUI-automation with Python.

This is true except for events in the Windows Event Log, which may be the case due to automation artifacts (Sec. 5.3).

Plaso provides various parsers for the different file formats, allowing to enrich traces with additional information. We use the output of these parsers to compare the different emulation approaches with regard to the distribution of generated traces. As we see in table 3, emulation of GUI-based user actions not only creates a greater amount of traces, but also results in a greater distribution of traces.

5.3. Automation artifacts

It is rather obvious, that the installation of automation software and guest agents produces datasets of lower quality, as disk images get polluted with unwanted artifacts like executables, libraries and more files. So we are more interested in additional automation artifacts the trace synthesis approaches create. In order to analyze these artifacts hinting to user action emulation, we evaluate which traces do not come up as part of the computer vision trace set. As a result, we found no artifacts of automation extracted by the sqlite-, lnk- and olecf-parsers, but traces in the event log, in the registry and in prefetch files strongly indicate usage of automation solutions in the Powershell- and Python-generated images (Tables 4, 5 and 6).

A lot of event log entries in the audit log correspond to remote logins, hinting to the execution of emulated user actions using the SSH protocol. As we configured Powershell to be the default login shell, remote logins lead to Powershell artifacts also when using the Python emulation approach. The traces found in the prefetch files also refer to emulation, as the execution of Python, Powershell, Psexec and SSH/SFTP show.

5.4. More realistic traces using computer vision

Both GUI-interaction approaches generate more widely distributed traces (Table 3) compared to the Powershell trace set. Browser operations result in a greater amount of cached pages and saved cookies, Windows Timeline artifacts in the registry contain traces of executed programs, and traces of file usage can be found in link files, OLECF files (AutomaticDestinations) and registry keys (MostRecentlyUsed). To evaluate in which way the usage of our computer vision approach result in even more realistic traces, we explore which traces are created when only using computer vision.

The contents extracted by the Plaso parsers show (Table 3), that the usage of our computer vision approach results in the creation of unique traces. The registry keys parsed with the *winreg/userassist* parser prove execution of programs via the GUI (Table 7), additionally documenting which programs were executed in using the Windows Task Bar. As those traces only show up in the computer vision trace set, we conclude that UserAssist entries can also be used as a proof of manual program execution. Moreover, we think those traces can be considered to be a good indicator for human-like interactions on a system, as automation solutions rarely run programs by clicking on items in the Task Bar, Quick Launch Menu or Start Menu.

In a similar manner, the results of the *winreg/windows_typed_urls* registry parser prove that typed input happened in the

Table 2
Overview - extracted traces.

Parser	Computer Vision (CV)	Powershell (Ps)	Python (Py)
winreg	711	630	675
sqlite	62	13	49
olecf	33	–	9
winevtx	954	844	1066
prefetch	69	53	68
lnk	6	–	2

[2] https://plaso.readthedocs.io/en/latest/.
[3] https://pandas.pydata.org/.

Table 3
Distribution of GUI-Traces.

Parser	CV	Ps	Py
olecf/olecf_automatic_destinations	8	–	3
olecf/olecf_automatic_destinations/lnk	14	–	2
olecf/olecf_automatic_destinations/lnk/shell_items	8	–	1
olecf/olecf_default	3	–	3
sqlite/chrome_27_history	18	1	15
sqlite/chrome_66_cookies	50	13	50
sqlite/windows_timeline	9	–	5
winreg/amcache	1	1	2
winreg/mrulistex_shell_item_list	2	–	–
winreg/mrulistex_string	4	–	4
winreg/mrulistex_string_and_shell_item	4	–	3
winreg/mrulistex_string_and_shell_item_list	1	–	–
winreg/userassist	8	–	–
winreg/windows_typed_urls	1	–	–

Table 4
Traces of automation - event log.

Event ID	Event Log Source	CV	Ps	Py
400/0x0190	PowerShell	–	7	8
403/0x0193	PowerShell	–	7	6
40961/0xa001	Microsoft-Windows-PowerShell	–	7	8
40962/0xa002	Microsoft-Windows-PowerShell	–	7	8
4097/0x1001	Microsoft-Windows-CAPI2	–	–	1
4103/0x1007	Microsoft-Windows-PowerShell	–	2	–
4634/0x121a	Microsoft-Windows-Security-Auditing	–	4	12
4717/0x126d	Microsoft-Windows-Security-Auditing	–	2	2
4718/0x126e	Microsoft-Windows-Security-Auditing	–	2	2
53504/0xd100	Microsoft-Windows-PowerShell	–	7	8
600/0x0258	PowerShell	–	42	48
7045/0x1b85	Service	–	–	6
800/0x0320	PowerShell	–	2	–

Table 5
Traces of automation - registry keys.

Parser	CV	Ps	Py
HKLM\Security\Policy\Accounts\default	–	1	1
HKLM\Security\RXACT\default	–	1	1
AMCACHE - PSEXESVC.exe	–	–	1

Table 6
Traces of automation - prefetch files.

Process	CV	Ps	Py
POWERSHELL.EXE	–	2	3
PSEXEC.EXE	–	–	1
PSEXESVC.EXE	–	–	3
SFTP-SERVER.EXE	–	–	2
SSHD.EXE	–	2	2
PYTHON.EXE	1	1	4

Table 8
Computer Vision - winreg/windows_typed_urls

Program	Value	CV
Explorer	C:\Users\win\sqlmap.zip	1

Windows Explorer (Table 8), in this case typing the path of the *sqlmap.zip* file.

We further explored which traces are not supported by Plaso parsers, but are nonetheless part of the computer vision trace set. We identified unique traces in the Windows Registry (Table 9), and manually examined the content to filter out those with forensic value.

As a result, we found the *AppCompatFlags* registry key to contain traces of program execution, in our case the execution of Microsoft's Edge browser. Moreover, the *CloudStore* registry key contains a lot of user configuration settings. The subkeys related to *windows.data.unifiedtile.localstartvolatiletilepropertiesmap* seem to hold current settings of the Windows Start Menu, including traces of file usage and run programs. In our computer vision trace set, the registry key contains traces which indicate the execution of Notepad and Windows Explorer as well as recently used files. Saved in a binary format not fully known to us, we suppose the key may hold additional forensic value.

Other registry keys of forensic value are those found as child keys of *HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer* (Table 10), which may contain further traces about the usage of files and programs. In our case, the *FeatureUsage\AppLaunch* subkey also hints at the execution of the Microsoft Edge browser.

In contrast to the analysis of the Windows Registry, analysing the Windows Event Log led to no unique traces of forensic value. Similarly, we found no unique traces of notable forensic content to be extracted from the other file formats.

6. Evaluation and discussion

As discussed previously, the quality of research outcomes, the successful education of forensic practitioners as well as the development of forensic tooling depends on how realistic underlying training datasets are. At the same time, the forensic community suffers limited availability of appropriate datasets, also known as the dataset gap problem. To overcome this challenge, previous work proposed to artificially create appropriate datasets by replicating real-world evidence. In general, those datasets are created by synthesizing traces in disk images utilizing user action emulation. Earlier studies presented different approaches for trace synthesis. However, we reasoned that common approaches provide inaccurate user action emulation, resulting in traces differing from those created by real-world user actions. It is important to reiterate even

Table 7
Computer Vision - winreg/userassist

Executed Program	CV
%APPDATA%\Microsoft\Internet Explorer\Quick Launch\User Pinned\TaskBar\File Explorer.lnk	1
%APPDATA%\Microsoft\Internet Explorer\Quick Launch\User Pinned\TaskBar\Microsoft Edge.lnk	1
%APPDATA%\Microsoft\Windows\Start Menu\Programs\System Tools\Command Prompt.lnk	1
MSEdge	1
Microsoft.Windows.Explorer	1
Microsoft.WindowsNotepad_8wekyb3d8bbwe!App	1
\Program Files\WindowsApps \Microsoft.WindowsNotepad _10.2103.6.0_x64__8wekyb3d8bbwe \Notepad\Notepad.exe	1
\Windows\System32\cmd.exe	1

Table 9
Computer vision - unique registry keys.

Registry Key	CV
HKCU\Software\Microsoft\Windows NT\CurrentVersion\ AppCompatFlags\Compatibility Assistant\Store	1
HKCU\Software\Microsoft\ Windows\CurrentVersion\CloudStore\Store\Cache\ DefaultAccount\$de${SID}$$windows.data. unifiedtile.localstartvolatiletile propertiesmap\Current\Data	1

Table 10
Computer vision - explorer registry keys.

Registry Key	CV
\ExtractionWizard\ShowFiles	1
\FeatureUsage\AppLaunch\MSEdge	1
\Wallpapers\BackgroundHistoryPath0	1

partly unrealistic traces negatively affect outcomes when used in research or for educational purpose. So with this work, we intended to improve trace synthesis by introducing a novel way of GUI-automation. Combining hypervisor features and computer vision algorithms to perform user action emulation, we developed a solution to replicate human-like user actions and synthesize truly realistic traces.

In this study, we showed that our solution was able to create more realistic traces than common approaches and may refine the quality of datasets in digital forensics. As groundwork for evaluation, we compared the resulting traces of our computer vision approach to those created by using Powershell or Python (Sec. 5). In line with the expectations of previous studies (Moch and Freiling, 2009; Du et al., 2021), we can observe that the usage of GUI automation creates a greater amount and more diversified traces. This holds to be true when using Python for GUI-automation, but we see these effects are even increased using our computer vision approach. Interpreting amount and distribution of traces as measures for realism of traces, we see this as a strong indicator for providing more realistic trace synthesis. Further evidence for this can be found in the creation of unique traces, proving human-like GUI interactions took place. These traces were only generated using our computer vision approach, providing realistic traces unachieved by other approaches. As we only implemented a small subset of user actions, we expect these effects to grow even further, the more actions we emulate for dataset synthesis. Complementary, the absence of traces in the Powershell- and Python-generated trace sets may be considered an artifact on its own, proving a rather unrealistic system usage without GUI interactions.

In contrast to other approaches, we can see that our computer vision solution beneficially omits the creation of traces hinting to automation. Similar to the incorporation of realistic traces, avoidance of automation artifacts results in higher-quality datasets, which is a further advantage. One may argue that other emulation variants may achieve the same, as occurring automation artifacts strongly depend on the emulation techniques brought to action. Nevertheless, we see the traces created in the context of emulation as exemplary, clarifying that avoidance of automation artifacts is hard to achieve when using in-system automation approaches. In conclusion, deploying other automation backends such as Selenium, Windows UI Automation or the usage of other guest agents would presumably lead to a similar amount of automation artifacts.

After providing the initial implementation, we found GUI-automation using computer vision surprisingly easy to handle. Emulating user actions mainly consists of taking screenshots of GUI-elements as templates, then emulating user input with a few lines of easy-to-understand code. We claim our computer vision approach is not only easier to use, but can emulate more actions than common API-based emulation. Our assumption is based on the fact that not all programs can be controlled by APIs, which can result in fundamental obstacles using current approaches. Using computer vision, we do not rely on any system APIs, and can even react to events that GUI-changes notify.

This way, our proposed solution offers a way to synthesize traces for all virtualizable GUI-based operating systems: e.g. MacOS, Linux, Windows and especially mobile operating systems, which are to date not supported by any of the existing trace synthesis frameworks (Gonçalves et al., 2022).

An apparent limitation of our method lies in the treatment of GUI-updates, as often introduced as part of software updates. Minor changes may not negatively influence the implemented actions, but as soon as GUI-changes become strongly visible, new templates for every action have to be provided. Even though the emulation of user actions becomes easier to handle using computer vision, there is still manual work required.

While we can emulate user actions and dynamically provide user input, there is currently no way to recognize successfully emulated actions. Of no consequence in our setting, this may affect long-term dataset creation, as undetected emulation failures could lead to useless datasets. It is not possible to read or interpret any textual output appearing as part of GUI-elements, limiting flexibility in the emulation of user actions. As both features would be helpful in long-running scenarios, they may be implemented in future releases.

It is important to highlight the fact that user action emulation may solve the problem of creating realistic datasets, but dataset exchange can still be narrowed down due to copyright restrictions of software contained in disk images. A solution to this would be to create and exchange evidence packs, as presented by Scanlon et al. (2017).

7. Conclusion

Digital forensics still faces a dataset gap problem (Gonçalves et al., 2022), negatively affecting research, education, and practitioner work due to insufficient training data. While prior work tried to overcome this challenge by synthesizing datasets, realistic traces of high quality are difficult to create and low quality affects further research. To tackle this issue, we introduced a novel way of user action emulation, optimizing the quality of synthetic datasets. By combining hypervisor features with computer vision algorithms, we have demonstrated that realistic user action emulation can be achieved. We have shown that our approach results in realistic trace synthesis, measurable in the amount and distribution of created traces and the absence of automation artifacts. Providing a way to populate datasets with realistic traces, we expect our proposed solution may be able to refine the quality of datasets used in forensics research and education.

To further validate this promising approach in larger settings and in long-running, automated data synthesis, our computer vision approach could be integrated into existing dataset-synthesis frameworks, combining the advantages of different solutions. In order to ease integration, we provide an open-source framework written in Python, available as *pyautoqemu*-package,[4] encouraging and empowering future work in the field.

In order to produce the most realistic results, the synthesis of datasets should take into account sociological and criminological

[4] https://wiwi-gitlab.uni-muenster.de/itsecurity/pyautoqemu.

facets as well as true-to-life settings. It would be useful to explore in which way these characteristics mirror in digital evidence, and then transfer the results to synthetic data creation. This may include future studies on typical user behavior and device usage, e.g. to answer how emulated user actions should get chained together to create realistic scenarios.

Acknowledgements

This research was funded by the research project "North-Rhine Westphalian Experts in Research on Digitalization (NERD II)", sponsored by the state of North Rhine-Westfalia – NERD II 005-2201-0014.

References

Abt, S., Baier, H., 2014. Are We Missing Labels? a Study of the Availability of Ground-Truth in Network Security Research, pp. 40–55. https://doi.org/10.1109/BADGERS.2014.11.

Du, X., Hargreaves, C., Sheppard, J., Anda, F., Sayakkara, A., Le-Khac, N.A., Scanlon, M., 2020. Sok: Exploring the State of the Art and the Future Potential of Artificial Intelligence in Digital Forensic Investigation, pp. 1–10. https://doi.org/10.1145/3407023.3407068.

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

Garfinkel, S., 2007. Forensic Corpora: A Challenge for Forensic Research 2007.

Garfinkel, S., Farrell, P., Roussev, V., Dinolt, G., 2009. Bringing science to digital forensics with standardized forensic corpora. Digit. Invest. 6. https://doi.org/10.1016/j.diin.2009.06.016.

Garfinkel, S., Nelson, A., Young, J., 2012. A general strategy for differential forensic analysis. Digit. Invest. 9, S50–S59. https://doi.org/10.1016/j.diin.2012.05.003.

Gonçalves, P., Dolos, K., Stebner, M., Attenberger, A., Baier, H., 2022. Revisiting the dataset gap problem – on availability, assessment and perspective of mobile forensic corpora. Forensic Sci. Int.: Digit. Invest. 43, 301439. https://doi.org/10.1016/j.fsidi.2022.301439.

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

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

Göbel, T., Schäfer, T., Hachenberger, J., Türr, J., Baier, H., 2020. A Novel Approach for Generating Synthetic Datasets for Digital Forensics, pp. 73–93. https://doi.org/10.1007/978-3-030-56223-6-5.

Keighley, J., 2017. Forgen. https://github.com/Jjk422/ForGen.

Krüger, M., 2014. Forgeosi. https://github.com/maxfragg/ForGeOSI/.

Moch, C., Freiling, F., 2009. The Forensic Image Generator Generator (Forensig2), pp. 78–93. https://doi.org/10.1109/IMF.2009.8.

Park, J., 2018. Trede and vmpop: cultivating multi-purpose datasets for digital forensics – a windows registry corpus as an example. Digit. Invest. 26. https://doi.org/10.1016/j.diin.2018.04.025.

Scanlon, M., Du, X., Lillis, D., 2017. Eviplant: an efficient digital forensic challenge creation, manipulation, and distribution solution. Digit. Invest. 21. https://doi.org/10.1016/j.diin.2017.01.010.

Visti, H., Tohill, S., Douglas, P., 2015. Automatic Creation of Computer Forensic Test Images, pp. 163–175. https://doi.org/10.1007/978-3-319-20125-2_14.

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

Yeh, T., Chang, T.H., Miller, R., 2009. Sikuli: Using Gui Screenshots for Search and Automation, pp. 183–192. https://doi.org/10.1145/1622176.1622213.

Glossary

API: Application Programming Interface
GUI: Graphical User Interface
HID: Human Interface Device.
OLECF: Object Linking and Embedding Compound File format
OS: Operating System
QMP: Qemu Machine Protocol