Forensic Science International: Digital Investigation 38 (2021) 301133

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS 2021 APAC - Proceedings of the First Annual DFRWS APAC

TraceGen: User activity emulation for digital forensic test image generation

Xiaoyu Du [a], Christopher Hargreaves [b], John Sheppard [c], Mark Scanlon [a, *]

[a] *Forensics and Security Research Group, University College Dublin, Ireland*
[b] *Department of Computer Science, University of Oxford, United Kingdom*
[c] *Department of Computing and Mathematics, Waterford Institute of Technology, Ireland*

ARTICLE INFO

Article history:

Keywords:
Forensic disk image creation
Evidence planting
User emulation
Tool testing and validation
Forensic education

ABSTRACT

Digital forensic test images are commonly used across a variety of digital forensic use cases including education and training, tool testing and validation, proficiency testing, malware analysis, and research and development. Using real digital evidence for these purposes is often not viable or permissible, especially when factoring in the ethical and in some cases legal considerations of working with individuals' personal data. Furthermore, when using real data it is not usually known what actions were performed when, i.e., what was the ‘ground truth’. The creation of synthetic digital forensic test images typically involves an arduous, time-consuming process of manually performing a list of actions, or following a ‘story’ to generate artefacts in a subsequently imaged disk. Besides the manual effort and time needed in executing the relevant actions in the scenario, there is often little room to build a realistic volume of non-pertinent wear-and-tear or ‘background noise’ on the suspect device, meaning the resulting disk images are inherently limited and to a certain extent simplistic.

This work presents the TraceGen framework, an automated system focused on the emulation of user actions to create realistic and comprehensive artefacts in an auditable and reproducible manner. The framework consists of a series of actions contained within scripts that are executed both externally and internally to a target virtual machine. These actions use existing automation APIs to emulate a real user's behaviour on a Windows system to generate realistic and comprehensive artefacts. These actions can be quickly scripted together to form complex stories or to emulate wear-and-tear on the test image. In addition to the development of the framework, evaluation is also performed in terms of the ability to produce background artefacts at scale, and also the realism of the artefacts compared with their human-generated counterparts.

1. Introduction

High-quality digital forensic test images are required across a range of digital forensic use cases including education and training, tool testing and validation, proficiency testing, malware analysis, and the research and development of novel evidence processing techniques (Garfinkel et al., 2009; Woods et al., 2011). The high demand for these test images coupled with the difficulty in creating them presents a problem in all of the above-mentioned areas of digital forensics (Garfinkel, 2007). Several collections of device images have been publicly released as standardised digital forensic corpora in the last few years, e.g., the Real Data Corpus (Garfinkel et al., 2009) and the National Institute of Standards and Technology's (NIST's) Computer Forensic Data Sets (CFReDS) datasets (National Institute of Sta, 2019). There are also other examples of digital forensic test data, e.g., SQLite databases (Nemetz et al., 2018). Nonetheless, the overall diversity of these corpora is insufficient for many of the aforementioned use cases (Grajeda et al., 2017). Grajeda et al. (2017) analysed 715 digital forensic articles published between 2010 and 2015 and discovered that many of the associated datasets used in these experiments are not publicly available. The authors found that, similar to network security dataset research by Abt and Baier (2014), that many digital forensic researchers manually produced their datasets, that the datasets are often not made public after the research is completed, and that there is a lack

* Corresponding author.
E-mail addresses: xiaoyu.du@ucdconnect.ie (X. Du), christopher.hargreaves@cs.ox.ac.uk (C. Hargreaves), jsheppard@wit.ie (J. Sheppard), mark.scanlon@ucd.ie (M. Scanlon).

https://doi.org/10.1016/j.fsidi.2021.301133

of standardised, labelled datasets available. The authors concluded that these compounding issues produces "one of the major disadvantages facing the cybersecurity/forensics community to this day, which is low reproducibility, comparability and peer validated research".

The current approach taken in the generation of new, scenario-based, previously unanalysed test images is for an expert (e.g., professors, researchers, developers, accreditation bodies, etc.) to manually create these images. The first step in this process is a significant planning phase involving the design of a scenario and planning of the types of artefact that are required, along with ensuring trained personnel are available to perform the task over potentially a prolonged period of time. The technical implementation phase typically commences with the manual creation of a fresh virtual machine instance, creation of user(s), installation and configuration of a set of desired applications (e.g., browsers, instant messaging applications, email clients, file-sharing tools, etc.). Subsequently, the precise execution of the scripted set of actions, or 'story', is performed (Moch and Freiling, 2009). This process is extremely arduous and can typically take days or weeks of experts' time (Yannikos et al., 2014). This often long drawn-out process will likely result in a single viable case study (Garfinkel et al., 2009). Maintaining, indexing, and updating a corpus of digital forensic images is also a labour-intensive, arduous task (Garfinkel, 2012). One concern with this manual approach is that there is typically little time allocated to creating a viable amount of background noise, wear-and-tear, and other non-pertinent actions on the device. This means that the need for students (in an education context) or newly developed techniques (in a research context) to cut through the large amount of irrelevant data on a real system is significantly reduced.

One attempted approach to supplying device images with a large volume of wear-and-tear is the purchasing of second hand hard drives, as discussed by Moch and Freiling (2011) in the context of digital forensic teaching. Bolstered with the global tightening of data protection guidelines, the legitimate ethical and now in some cases legal concerns of this approach, this remain a subject of debate. As a result, using real-world data from second hand hard drives or previous investigations is not a viable approach in many instances.

This work aims at addressing the issue of emulating user activities and behaviours ensuring forensically realistic traces are created in the resulting test images – indistinguishable from those created by regular computer users.

The work specifically makes the following contributions:

- It provides a proof-of-concept framework for generating realistic user data inside a disk image.
- It provides several plugins for the framework to emulate user actions at different levels of complexity, from simple file copying, to a "Google research session" on a particular topic.
- It collects the network traffic generated by actions on the machine.
- It documents a method for validating the artefacts generated by automated user simulation against data generated by a human.
- It discusses in detail the lessons learnt from attempting to implement this framework including the intrusiveness of tools on the generated data sets. It also sets out a clear research agenda for extending this framework to provide major benefits in digital forensic education, research, and investigations.

The remainder of this paper is structured as follows: Section 2 provides an overview of the related work in the current literature and also tools available for user activity automation. Section 3 describes the overall methodology used in the creation of this framework and the software design. Sections 4 and 5 provide the results and evaluation of the approach. For the purposes of this paper, the evaluation focused on the component actions (i.e., the building bricks for more complex stories). Specifically, Section 4 demonstrates the volume of artefacts that can be generated in a small time period, but that simulate a long period of system use and background wear-and-tear. Section 5 considers the forensic traces of the automated approach to artefact generation, versus traditional human-preformed actions. Section 6 provides a discussion and evaluation of the work carried out and also describes the many opportunities for future work in this area.

2. Related work

This section covers some of the related work in this area, including the needs for datasets, requirements for datasets, and the several approaches that have been previously explored to avoid the manual effort required for synthetic disk image generation.

2.1. The need for digital forensic datasets

The need for realistic datasets is best explained by Garfinkel et al. (2009): "From a training and educational perspective, it is difficult to overstate the need for realistic data sets. Anyone who has been on the instructors side of the process will testify to the huge investment of time that goes into creating realistic forensic scenarios. Much of this work is not shared broadly and that is clearly inefficient and wasteful in a relatively small field with limited budgets." With the increasing need for artificial intelligence aided digital forensic investigation (Sanchez et al., 2019; Anda et al., 2020), there is a corresponding growth in the need for large, labelled datasets to be used for training these models (Du et al., 2020). The accuracy and reliability of pre-trained machine learning/deep learning based models is only as good as the data on which they are trained (Du et al., 2020).

A number of existing datasets including disk images, mobile phone dumps, and network captures are available in a Digital Forensic Corpora,[1] but at time of writing there are only eight "full scenarios" that could be found.

2.2. Desirable characteristics for digital forensic test images

Woods et al. (2011) outline four desirable characteristics of "realistic" educational corpora and digital forensic test images (adapted by Scanlon et al., 2017):

1. Answer Keys – these are solutions to the problems posed to students incorporating guidance as to what evidence could be located in which digital artefacts.
2. Realistic Wear and Depth – the sample hard drive images should contain realistic wear patterns, i.e., the hard disk image being investigated should have regular usage surrounding email, web browsing, application installations, file creation and deletion, and downloaded content.
3. Realistic Background Data – a key skill for a digital investigator to gain is the ability to decipher between pertinent and non-pertinent data on a machine. The injection of "incriminating" data should not be obviously the only non-OS/non-application data stored on the disk.
4. Sharing and Redistribution – as a general guideline, hard disk images created for the purposes of education should be made freely available for others to download.

[1] DigitalCorpora.org.

2.3. Manual disk image generation

This refers to the process described earlier where an expert is required to carry out all of the actions according to a designed scenario. In the context of the test image requirements from Woods et al. (2011), manual generation does allow answers to be created, but they need to be manually constructed after the disk image is finalised – although, the exact actions performed are known and can be manually documented. Realistic wear-and-tear and background data is very difficult for this method. However, any actions that are performed will be realistic as they are carried out by a real person, albeit pretending to be the user in the scenario. In terms of distribution, theoretically this is unaffected by the manual process, but anecdotally, because of the large amount of effort and therefore cost to create manual disk images, they are considered a valuable asset to organisations, which may hamper sharing, as identified by Grajeda et al. (2017).

2.4. Existing approaches for image generation

There are several examples of previous work that have attempted to overcome this need for manual disk image generation.

Moch and Freiling (2009) discusses the development of *Forensig2*, which is subsequently further evaluated in (Moch and Freiling, 2011). *Forensig2* allows automated artefact generation of both the hardware and software level, i.e., it can programmatically configure the virtual machine in *QEMU*, including the CPU, network, disks, etc. It can also carry out actions within the VM, including configuring partitions, copying files locally and remotely, and also various other operating system level actions (on Linux), such as installing software (using the command line). This work introduces many of the concepts to be taken forward in any automated disk image generator, i.e., logging of actions performed to use as a ground truth, the need for extensibility, and the concept of reproducible randomness (discussed in Section 6). However, it is difficult to see how the approach could realistically synthesise all of the actions on a Windows system caused by for example a user opening a file, which includes numerous registry artefacts, link file creation, jumplist entries, potentially Edge browser artefacts, etc. This limited ability to generate artefacts for a GUI based OS is acknowledged by Moch and Freiling (2011).

Yannikos et al. (2014) presents ‘model based generation of disk images’, which focuses on creating a formal model of the scenario to be built. Practically, the actions that were achieved were: creating file systems, creating and deleting files, writing raw data to a disk, downloading a file from the Internet, and disk image import/export. This would be very effective for file system level interactions, but to generate a realistic disk image that could be used to teach forensic investigation techniques at all levels of abstraction, operating system-level artefacts would also be needed and far higher-level operations would need to be emulated.

More recently the tool EviPlant, described in (Scanlon et al., 2017), provides solutions to a number of problems, certainly around distribution of digital forensic images through the use of ‘evidence packages’ and discusses the concept of injecting data into baseline images. While injecting background noise to a disk image to increase the content volume of a disk image does at least frustrate simple file browsing as a means to find files of relevance to a scenario, to generate artefacts over multiple artefact types, e.g., injecting web history, file system artefacts, event logs, registry files, that are all evidentially consistent with each other is extremely difficult. The EviPlant approach, while potentially increasing the reusability of manual effort, does not address the issue of reducing the manual effort in creating the evidence packages to begin with.

2.5. Summary

There is a clear need for high quality and diverse datasets in digital forensics. Real data will have its place in testing, but is difficult to obtain and manage, and is increasingly surrounded by ethical and legal concerns. Synthetic data is either manually produced, which is time-consuming, but does provide realistic artefacts for the actions performed. However, it is difficult to produce the volume of actions needed to produce a realistic “full-system” with a realistic, long usage history. Automated approaches have focused on injecting artefacts into disk images, which allows for larger scale artefact generation, but once a detailed examination is conducted, the limited realism of the disk image artefacts is likely to be revealed. Otherwise automation based approaches have delivered only command-line level user emulation, which cannot produce a disk image containing all the relevant artefacts needed to properly represent modern Windows based systems. This research aims to address these problems and provide synthetic, automated, realistic, digital forensic artefacts at scale.

3. Methodology

3.1. Overall design

The overall aim of the research is to provide an automated approach to generating disk images for research, teaching, or the validation of digital forensic tools or techniques. As a result of the limitations of injecting files into a disk image discussed earlier, the high-level approach is to attempt to programmatically automate the actions of a user and allowing the system to generate realistic artefacts, rather than trying to artificially create them.

The types of actions that can be performed on a target VM are therefore split into two categories:

- Machine Control Actions performed outside of the VM, e.g., power on the VM, unsafe shutdown the VM (i.e., “pull the plug”) and adjust the BIOS time.
- External User Actions performed from outside the VM, e.g., copy or move files, perform a Google search, shutdown the VM in a controlled manner.
- Internal User Actions performed inside of the VM, e.g., copy or move files, perform a Google search, shutdown the VM in a controlled manner.

An overview of the system design is shown in Fig. 1. At present, if we execute code inside the VM to automate internal user actions, there will be inevitable traces left on the disk of the VM that would not be present with human-only generated data. However, building disk images inside virtualisation platforms, which is common practise, is at some level inconsistent from a real system (for example virtual hard disk identifiers, virtual USB controllers, etc.) Certainly in educational assignments/proficiency test, this can be covered using the phrase “During your analysis, please ignore any virtualisation artefacts that are part of the data generation process”. Students/Test takers could also be instructed to ignore results of the artefact automation process, if these artefacts can be identified, minimised, and segregated. This is discussed further in Section 5.

3.2. Choice of technologies

The virtualisation technology used for this research was VirtualBox, owing to its cross platform compatibility. This could also be said for VMware, but as VirtualBox is freely available, this approach could be used by any organisation regardless of any budget limitations. Qemu is another option, but the setup process is

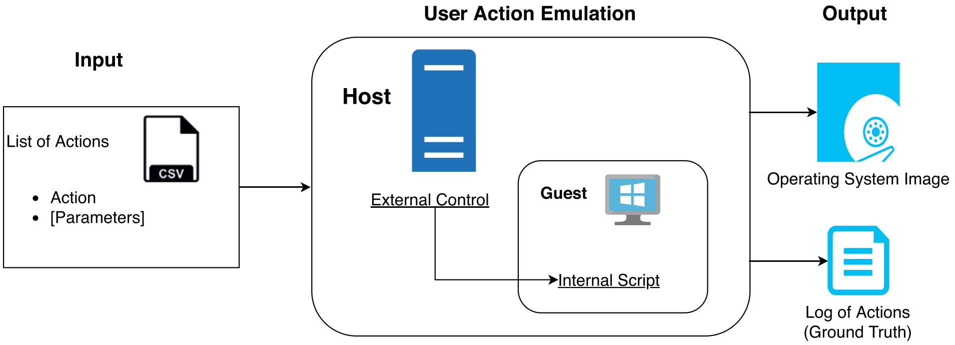


Fig. 1. Overview of the Image Generation Approach: the input is a list of user actions, the script running on the host controls the guest machine boot, performing internal and external actions and shut down. In the end, a forensic image with user traces in the guest OS is generated alongside the log recording the actions executed.

slightly more complex and the intention was to produce images automatically that could be easily later supplemented with later human-generated actions.

As per Moch and Freiling (2009), the automation is performed using a series of Python scripts with some additional libraries used to automate user actions. Python was chosen for its advantages for rapid prototype development and the large range of libraries available for emulating user activity.

The operating system chosen as the guest is Microsoft Windows as this still represents the majority of end user computers[2], and therefore the majority of systems seen in digital forensic laboratories.

3.3. Existing automation options

When implementing emulated user actions, there are a number of options available. At a high-level these include:

- Application Programming Interfaces (APIs) – For example, pywinauto[3] facilitates the automation of the Windows GUI using the Win32 API. It allows Pythonic interaction with GUI components, e.g., to save a file in Notepad:

```
app.UntitledNotepad.menu_select ("File->SaveAs")
```

- Simple Mouse and Keyboard Control – This allows injection of keyboard input, mouse movement and clicks at specific coordinates.
- Graphical User Interface (GUI) Interaction – this technique is more advanced than ‘blind clicks' above, and can also visually process any given application and programmatically execute specific mouse and keyboard actions. Examples of GUI based automation tools include Sikuli (Yeh et al., 2009) and PyAutoGUI. Sikuli enables screenshots of GUI control elements to be taken (such as a toolbar button or icon) which can be included in a sequence of actions in order to script complex interactions with any application. In a similar vein, PyAutoGUI[4] is a cross-platform GUI automation tool designed for programmatically controlling the mouse and keyboard. One benefit to this approach over Macros is that the script is resilient against any specific control element not appearing in precisely the expected coordinates on screen.
- Browser Automation – There are also application specific automation tools that could be used, for example Selenium[5] that automate most major web browsers.

3.4. Actions to automate

A small subset of user actions have been selected for automation in this proof-of-concept in order to test the validity and to show the potential of this approach. They can be broadly categorised into *component* and *compound* actions. The *component* actions are small individual actions that can be performed on a system, e.g., a file copied or moved, application launched, machine shutdown, etc. *Compound* actions are a set of *component* actions to be performed that may typically run over several minutes or hours, for example a “Google research session” would involve opening a browser, searching for a topic, viewing results by going back and forth to the search results. “Browsing the news” would involve opening a browser, visiting a news site and reviewing several articles of the day. Examples from both category have been implemented and also evaluated in terms of their “intrusiveness” to the disk image and realism compared to human-only actions.

3.5. Implementation of VM external actions

Two actions have been considered here: booting the VM and setting the BIOS time. Since we are using VirtualBox, the `VBoxManage` tool can be used to manipulate the VM.

Using `VBoxManage`, the VM can be booted using the command:

```
vboxmanage --nologo startvm [vm_name]
```

In addition, the BIOS time of the VM can be set with the command:

```
vboxmanage modifyvm [vm_name]
--biossystemtimeoffset [msec]
```

The parameter *msec* specifies a fixed time offset, in milliseconds, of the guest relative to the host time, which can be either a positive or negative offset. These external actions are necessary since actions such as booting the VM can obviously not be performed from inside the VM if the VM is not powered.

If network traffic associated with the activities is to be collected, the following command will create a.pcap dump of all packets:

```
vboxmanage modifyvm [vm_name] --nictrace[adapter-number] on --nictracefile[adapter-number]
file.pcap
```

Snippet 1 shows the developed method to set the system date and time for action emulation.

[2] 87.36% at time of writing (https://netmarketshare.com).

[3] https://pywinauto.readthedocs.io/en/latest/.

[4] https://github.com/asweigart/pyautogui.

[5] https://www.seleniumhq.org.

3.6. Implementation of VM internal actions

Causing actions to occur within the VM is slightly more complex, but again can be partly achieved using `VBoxmanage`. `VBoxmanage` allows you to execute a program inside the VM and also pass parameters to it. An example call to launch `notepad.exe` is shown below. Note the need to also pass parameters for the user account and user password to enable this.

```
vboxmanage −−nologo guestcontrol [vm_name] run
−−username [username] −−password [password] −−exe {}
−− “notepad.exe”
```

However, we require more complex interaction than simply launching a program so in the TraceGen framework, the Python interpreter is invoked inside the VM, referencing a specific internal script, along with some arguments. For example:

```
vboxmanage −−nologo guestcontrol [vm_name] run
−−username [username] −−password [password] −−exe {}
−− ”python.exe” [internal_script] [script_args]
```

This approach has some prerequisites. Firstly Python and any required packages must be preinstalled in the VM and the scripts to be run need to be made accessible to the VM. In the examples outlined in this paper, the Python scripts were added to a second drive (mapped to a specific drive letter in the VM), which can be detached from the VM on completion of the scenario generation. Anybody analysing the disk image can be told to ignore that specific drive letter.

In terms of specific internal actions discussed in this paper, the first *component* action is a file copy. The most basic way that this can be achieved is using the Python `shutil` module to copy a file from a source path to a destination. This is executed using the internal script `file_copy.py`. The resultant forensic artefacts of this action are discussed in Section 5. However, it is clear at this point that this does not reflect the manner by which a user in a graphical operating system would really copy a file from one destination to another. While there are many ways to achieve this objective, one option is to open a folder (via a double click of the folder icon), use the keyboard shortcut ‘Ctrl + C’, then open another folder, and use ‘Ctrl + V’. The consequences of each of these alternatives are discussed in Section 5.

A more complex internal action, a *compound* action, is a *Google research session*. This represents using Google to search for a particular keyword, and then viewing a subset of the results over a defined period of time. This could be for background noise generation or part of the relevant aspect of the scenario, e.g., researching wireless hacking tools. Several methods were considered to achieve this. The first method considered was to launch Chrome by injecting a press of the Windows key, keystrokes to type ‘Chrome’ followed by the enter key, as shown in Snippet 2. It then injects keystrokes into the address bar, which by default in Chrome performs a Google search. In parallel, the content of the search result page is retrieved, scraped with the *Beautiful Soup*[6] library and the resultant links also injected into the address bar, resulting in visits to those pages. However, this does not reflect the reality of the links being clicked by a user as each of the links are typed, rather than a link clicked. This therefore may not result in realistic associated artefacts. As an alternative, a browser extension called *SendKey* was installed, which can be configured to allow keyboard navigation of Google search results. With this in place, the keyword can be injected into the Chrome address bar (performing a Google search for that keyword), then the results selected and clicked using keyboard shortcuts, as demonstrated in Snippet 3.

As a second example, rather than using the file copying *component* action to transfer user data onto the virtual machine, where care is needed to ensure timestamps are consistent with the scenario, another method is to generate content directly on the VM. This is another example of an internal *compound* action, and uses the `pywinauto` module, which in turn uses the Win32 API to interact with applications on the VM and can be used to populate a disk image with content. This plugin can receive two arguments at present, the content to be ‘typed’ into the notepad file, and the path on the virtual machine that the file should be saved. The code used to achieve this is shown in Snippet 4.

3.7. Building stories

For the purpose of this work, the list of actions to be performed on a VM is termed a “story”. Stories are currently captured in individual Python scripts but the entire set of actions is contained within a single list, so using CSV, TSV or some other standard representation is trivial.

Each action to be performed consists of:

1. Date and time to perform action.
2. Action to perform.
3. Arguments related to the action to perform.

There are two modes by which a story can be executed. The first is *live simulation* mode. In this mode, the controlling script checks the time for the next action to be performed against the host (assumed to be correct) and the actions are performed when the scheduled time arrives. This means that the actions are carried out in real-time, but in an automated way. This mode provides the advantage that as all actions are performed in real time resulting in all timestamps, whether from the virtualised system or remotely fetched content, will be consistent. The disadvantage is that to generate several months activity, it would take several months. Therefore, the second mode is *compressed-time simulation*. In this mode, for each action, the clock of the virtual machine is adjusted to the specified time, then the action carried out. This has the disadvantage that there will be timestamp artefacts that are not consistent, but it does allow a large amount of activity to be synthesised in a very short time. A demonstration of this is shown in Section 4.

3.8. Virtual machine configuration

In addition to the internal and external scripts and the stories, at present, the VM must also be configured in a specific manner that is conducive to simulated user actions. The following changes were made to the VM to allow reliable user action emulation: 1), install *VirtualBox Extension Pack* on Host OS; 2), install *Guest Additions* on Guest OS; 3), install Python on Guest OS; 4), install related Python libraries and any other dependencies, e.g., SendKey; 5), install app needed in the action emulation on Guest OS, e.g., Google Chrome. There are also several other more subtle changes that are needed. For example, for now, it is necessary to auto-log in the user, rather than providing a password. Also, as the system may be running for several days or hours, the power settings of the guest and the host were modified to avoid sleeping or powering off displays as this could interfere with the user emulation process.

4. Results: continuous usage trace generation

This section demonstrates the volume and complexity of the artefacts that can be generated and considers the machine and human time costs of each approach. For this test the *compressed-time simulation* mode of the tool was used. Setting the system time

[6] https://www.crummy.com/software/BeautifulSoup/.

before booting the machine enables the emulation of usage over a much longer period, i.e., weeks, months or years of activity emulated in just hours or days. An input CSV file defines the actions to be executed on the machine. A random method can be used to make sure repeated operations time would not be exactly the same. The same *component* or *compound* actions can result in different artefacts being generated due to the specified arguments, e.g., which file to open.

Two experiments have been performed:

1. Setting the system time to a point in the past each time repeat simple actions: 1) set system time; 2) boot VM; 3) create notepad file; 4) shut down VM.
2. Setting the system time to a point in the future, repeat simple more actions each time: 1) set system time; 2) boot VM; 3) create notepad file; 4) copy `*.png` file from one folder to another; 5) use browser 6) run app WinSCP and login; 4) shut down VM.

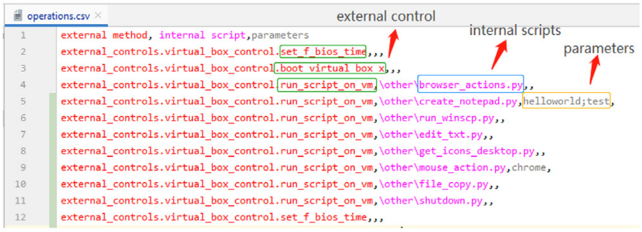
How many times to repeat machine boot actions and what actions to do each time the machine is booted are specified in the input file, as shown in Fig. 2. Continuous running and action execution is easy to specify, i.e., repeat the instruction in the CSV and change the parameters if desired. As observed from performing the experimentation, when an exception occurs, the process continues to perform to the next action. For background noise generation, this situation is acceptable; if the running is for emulating criminal actions, the audit log file should be checked after the generation process.

4.1. Comparison method

For analysis the traces left on the machine, *Plaso* is used for creating a “Super Timeline”. The generated timeline consists of events extracted from sources on different abstract level (e.g., file system, windows logs, registry, chrome history, etc.). Two disk images of the VM were made, one before the automated action emulation, and one after. And then two timelines were generated and compared.

4.2. Timeline comparison

Pandas is used to analyse the timeline generated from the test images. The generated timeline was a 2.5 GB CSV file. Due to the executed story relating to 2019, the first step operation on the timeline is to use a filter only focusing on timestamps from 2019. From the generated timeline, one important property is the source of event. In this experiment, there are eight categories, as listed in Table 1. The number of timestamps are also shown in this table, both before and after the automated actions were executed. Windows Event (EVT), Registy (REG), Web History (WEBHIST), Log (LOG), Portable Executable (PE), Link (LNK) can be seen to have increased. The Number of Object Linking and Embedding (OLE) had no changes. As one might expect, using the Windows OS changes the records in EVT, REG, LOG, PE, etc, running applications changes the Windows REG entries and Chrome usage changes the WEBHIST.



```
external method, internal script,parameters
external_controls.virtual_box_control.set_f_bios_time,,,
external_controls.virtual_box_control.boot_virtual_box_x,,,
external_controls.virtual_box_control.run_script_on_vm,\other\browser_actions.py,,
external_controls.virtual_box_control.run_script_on_vm,\other\create_notepad.py,helloworld;test,
external_controls.virtual_box_control.run_script_on_vm,\other\run_winscp.py,,
external_controls.virtual_box_control.run_script_on_vm,\other\edit_txt.py,,
external_controls.virtual_box_control.run_script_on_vm,\other\get_icons_desktop.py,,
external_controls.virtual_box_control.run_script_on_vm,\other\mouse_action.py,chrome,
external_controls.virtual_box_control.run_script_on_vm,\other\file_copy.py,,
external_controls.virtual_box_control.run_script_on_vm,\other\shutdown.py,,
external_controls.virtual_box_control.set_f_bios_time,,,
```

Fig. 2. Sample input CSV file.

Table 1
Event counts from different sources.

Event Source	Before Story	After Story	Increment
EVT	183,628	246,653	63,025
REG	81,181	125,598	44,417
WEBHIST	27,869	30,689	2,820
Log	2,753	3,160	407
PE	1,019	1,032	13
LNK	621	852	231
OLECF	348	348	0
Total	297,419	408,332	110,913

One interesting analysis is on the diverse value generated for each property. The diverse value for the date relates to how many days this machine was used, i.e., as long as the machine is being used, correlating timestamps are generated. The value counts for file name represents how many different file names are in the machine. Inode counts represents the file number on file system level. Table 2 shows the value counts for each of these different properties. Before the story was executed, in the generated timeline, the diversity of timestamp dates is 189. After the story's execution, it is 204. The increment on the count of file names is 11,099 representing the number of new were generated during the story's emulation.

Two continuous running test were conducted. The first is to set the system date and time in the past and the second is to set it to a point in the future. These are date in September 2019 and October 2019 respectively. Figs. 3 and 4 shows the number of event changes detected for each day. It can be seen that the day configured in the story results in corresponding changes on the timestamp. Also, as can be seen, the number of timestamps increased after the story's execution emulation. This is a result of later actions updating the same specific files modified by previously executed actions.

4.3. Summary

The experimentation and analysis presented in this section verified that the TraceGen framework can operate in a viable manner. This framework enables 1), a large amount usage traces generated in a compressed time period, 2), usage traces generated at a specific date in time, and 3), the generated actions are in a diverse set of categories.

5. Results: forensic traces of automation

This Section examines the forensic artefacts left as a result of the automated approach and compares them with ‘human-generated’ activity.

5.1. Comparison method

To determine if the actions generated using the automated approach are consistent with ‘human-generated’ actions, the following methods were used:

Table 2
Value counts of different properties.

Event Property	Before Story	After Story	Increment
date	189	204	15
file name	90,941	102,040	11,099
inode	31,350	36,139	4,789

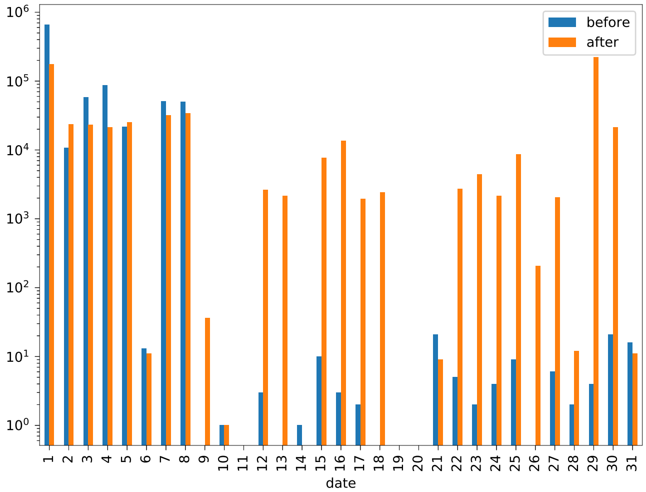


Fig. 3. Event counts of each day in September.

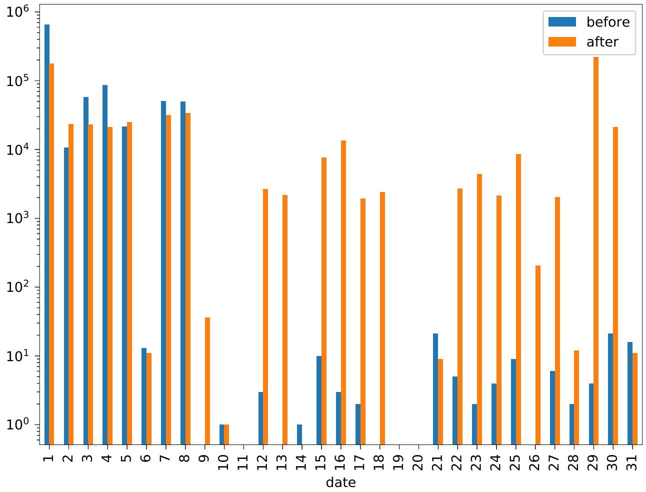


Fig. 4. Event counts increment of each day in October.

1. A disk image of the VM was taken.
2. A Procmon capture was started.
3. The action was performed (automated/human).
4. The Procmon capture was stopped and saved.
5. A disk image of the VM was taken again.

To analyse the resulting data, first the Procmon logs were inspected and the files and registry changes were recorded[7]. The final disk image was also processed using Plaso to generate a timeline of activity that could be inspected in detail around the timestamp of the action performed.

5.2. Results: running a simple script using `VBoxManage`

For this initial test, a 'hello world' Python script was written and added to the VM. This was scheduled to run using the TraceGen software and Procmon was used to record changes made to the system. While there were thousands of events recorded, most were read operations. Filtering the Procmon events by the filters described earlier indicated changes to the SAM registry hive. Furthermore, generating a timeline with Plaso also indicated similar file changes, and changes to the security.evtx event log.[8]

To investigate in more detail, an additional experiment was performed where the SAM registry hive was examined before and after the 'hello world' action was automatically executed. Looking at the 'F' value in the SAM file for the virtual machines's user, the last login time was indeed updated to the time of the scheduled action, and the login count was also updated by one. Examining the Security Event log, related events were also found showing login events (4624) and logoff events (4634) that originate from the `VBoxService.exe` process. This shows that running processes in the virtual machine using `VBoxManage` is not unintrusive, but using this experimental method, the effects can be measured, and if appropriate, ignored.

5.3. Results: automated file copying

This is the component-level action discussed above that takes an existing file and copies it to a new location. The automated approach for file copying achieves this using the Python `shutil` module. The changes identified using Procmon were all file system related, i.e., the creation of the new copied file, but also changes to `$LogFile` and the `$MFT`. This was also confirmed in the Plaso timeline output.

However, the typical set of actions for a human to copy a file from one location to another would be to open a folder location, perhaps use Ctrl-C to copy the file, browse to another folder, and use Ctrl-V to paste it in place. This manual approach was also tested and the artefacts recorded. Using this manual method, there were additional artefacts generated at the operating system level of abstraction that are not consistent with the simple file copy above using Python. For example, Procmon recorded registry changes to the ShellBag registry keys in `USRCLASS.DAT` resulting from accessing the two folders that contain the files copied. These artefacts can be of great benefit when identifying recent locations accessed by a user, so neglecting to generate them would remove a significant investigative lead, which will be even more important as we increase the amount of background noise on the generated disk images.

This difference shows the careful consideration that is necessary to exactly replicate the set of user actions that will perform the action to be simulated. In this case, the process still needs improvement as using a simple `shutil` operation, relevant operating system level artefacts are not generated that could have an effect on expert forensic analysis.

5.4. Results: automated 'Google research session'

This is a compound-level automation that opens Chrome, conducts a Google search for a term ('hello world' in this example) and views several of the search results. The analysis of this automated action was slightly different to the previous. The Plaso output for the period of automated Chrome browsing contained 1722 entries, including file creations, e.g., cached items from browsing, registry changes, e.g., user assist from launching Chrome, USN journal entries related to other file changes, and browser history, cache and cookie entries. The remainder of the analysis here focuses specifically on the browser history.

[7] The Procmon filters used were: operation is: WriteFile, RegDeleteKey, RegDeleteValue, RegRenameKey, RegSetInfoKey, RegSetValue, RegCreateKey. Also all default 'exclude' filters were removed to ensure nothing was missed.

[8] There were other files also listed: `Windows/System32/wdi/LogFiles/BootCKCL.etl` and `$LogFile`, but these were inspected and have not yet been found to be relevant.

The Chrome history file was exported and examined in a SQLite browser. As shown in Fig. 5 (the URLs table), there are entries for the Google search for ‘hello world’ (row 2), plus two other related pages (rows 3 and 4). If Fig. 6 (the visits table) is also considered, the search itself has visit id = 3 (referring to URL 2), followed by the visit to Wikipedia, which is visit 4 (URL 3), which even has a from_visit entry of 3. This shows that the full breadcrumb trail of link following is maintained even with the automated web browsing that makes up the user emulation. In a more general sense, even though there are additional artefacts related to running scripts and a browser extension installed (which could be filtered out), with care, it is possible to emulate user actions and create detailed artefacts that are the same as if a real user were carrying out the actions.

While this is a limited analysis, extending this analysis to the Current Tabs file, the cache, artefacts left from the SendKey extension, and all other 1,722 changes identified by Plaso, is outside the scope of this paper. Furthermore, it does not actually lend itself to this sort of manual analysis, which would be incredibly time-consuming and error prone. This, therefore, opens up a new research problem of trying to automatically compare the artefacts left from two sequences of actions, one manually performed by a user, and one automated. This would allow verification of individual automated actions to check that they are an accurate simulation and do indeed generate artefacts that are representative of real-world actions, and can then be packaged into stories.

6. Discussion, evaluation, and further work

As shown in Section 4, the TraceGen framework can be run in a mode such that with a small amount of input from a user to create the story to be generated, and a relatively short amount of machine time, it is possible to generate a large number of artefacts, including content, as well as file, operating system and application timestamps that span a much longer history of the virtual machine. Section 5 has also shown that while care does need to be taken that any automated actions do actually reflect what happens when a real user carries out the same action, it is possible to compare the two, and to some degree of abstraction the artefacts can be made consistent.

Despite the successful demonstrations and artefact comparisons discussed in Sections 4 and 5, there are several limitations to the current prototype and approach.

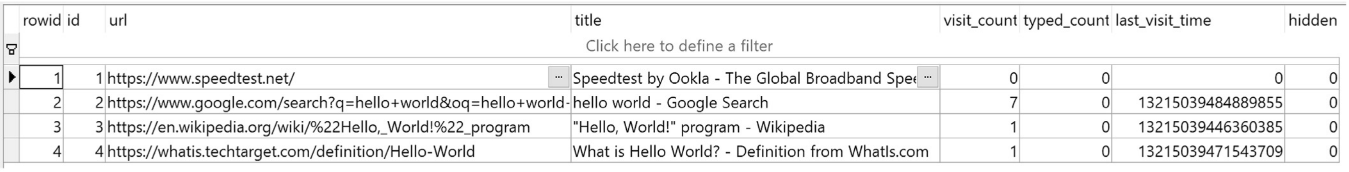
Even though it is possible to measure the artefacts that are inconsistent between the approaches, these differences do exist. Examples discussed can broadly be grouped into two categories. The first group are limitations of the individual internal scripts in terms of mirroring the artefacts left by real user actions, e.g., the file copy example that does not generate operating system artefacts, such as Shellbag entries. This is due to over-simplification in the user automation. This can be overcome by firstly identifying that there are differences, then finding alternative or additional automation mechanisms that do create the expected artefacts. Secondly, the overall approach of external control running internal scripts, which requires a login and therefore each process that is run updates the user login count.

Both of these are anticipated to be overcome in the next generation of the framework (which will be released as an open source tool). As a result of attempts to generate realistic file copying artefacts at the operating system level, the GUI automation tools *Sikuli* and *pywinauto* were tested. However, at present neither offer the flexibility needed to provide an arbitrary filename and allow GUI manipulation that would open a folder containing that file, and then copy it to another folder. Therefore, the next stage of the work involves development of bespoke computer vision approaches that can be used for the specific purposes of automating user actions that are of value to generating synthetic forensic disk images, and also validating that automated actions completed successfully. The gold standard for this work is full external control of the guest operating system from the host, resulting in no internal artefacts that relate to the automation. Several components are already in place, i.e., stories, sets of actions, external keyboard control, ability to screenshot the guest, etc, but additional capabilities for mouse control and the computer vision components need to be developed further.

There are also limitations to each of the modes in which the user automation can be run. As discussed in Section 3.7, the software can be run in either *live-simulation* mode or *compressed-time* mode. For the former, it does have the advantage that all timestamps that will be generated within the disk image will be consistent with the scenario since the VM is in-sync with the host and the real-world time. However, this does have the limitation that the software must be run for much longer, in fact, it must be run in real time for the period to be simulated. However, it should be reiterated that this is machine time, not human interactions, which is a significant cost and effort difference. However, in recognition that in some cases the correctness of some of the detailed timestamps may not be of interest and simpler disk images just with large volumes of artefacts need to be generated, the compressed-time mode allows this. However, there are likely to be detectable timestamp inconsistencies in the data generated. To identify and quantify both of the above differences, a comprehensive method also needs to be developed to easily validate the actions left by automated actions, against those left by human-generated actions.

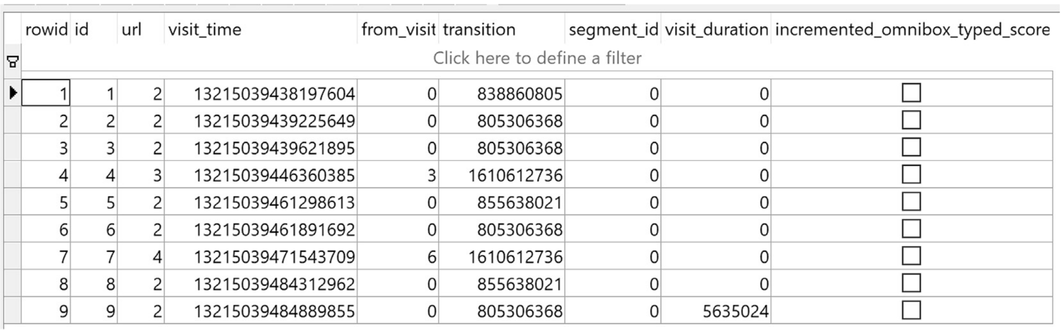
The specific implementation that has been developed also has several limitations. Of all the desirable user actions for automation within a scenario, only a subset have been implemented and evaluated for realism against human-generated actions. However, we have documented the approach used to conduct that evaluation, and demonstrated examples of how the automation can be changed to better represent a real-life action. More further work is also required in this area and an automated way of comparing the traces left by a real-life action against the machine generated set.

The implementation also has limitations in the expression of the stories. At present the main format for expressing the events to be carried out contains absolute times, i.e., run this action at this specific time. It is likely that a more flexible approach will need to be adopted for all of the potential use-cases of the TraceGen framework. For example, supporting relative times, i.e., run this command X minutes after the previous, or introducing randomness, e.g., run this command between X and Y seconds after the



rowid	id	url	title	visit_count	typed_count	last_visit_time	hidden
1	1	https://www.speedtest.net/	Speedtest by Ookla - The Global Broadband Spee	0	0	0	0
2	2	https://www.google.com/search?q=hello+world&oq=hello+world-	hello world - Google Search	7	0	13215039484889855	0
3	3	https://en.wikipedia.org/wiki/%22Hello,_World!%22_program	"Hello, World!" program - Wikipedia	1	0	13215039446360385	0
4	4	https://whatis.techtarget.com/definition/Hello-World	What is Hello World? - Definition from WhatIs.com	1	0	13215039471543709	0

Fig. 5. The urls table from the Chrome browser history showing a Google search for ‘hello world’, and two follow-up page visits.



rowid	id	url	visit_time	from_visit	transition	segment_id	visit_duration	incremented_omnibox_typed_score
			Click here to define a filter					
1	1	2	13215039438197604	0	838860805	0	0	□
2	2	2	13215039439225649	0	805306368	0	0	□
3	3	2	13215039439621895	0	805306368	0	0	□
4	4	3	13215039446360385	3	1610612736	0	0	□
5	5	2	13215039461298613	0	855638021	0	0	□
6	6	2	13215039461891692	0	805306368	0	0	□
7	7	4	13215039471543709	6	1610612736	0	0	□
8	8	2	13215039484312962	0	855638021	0	0	□
9	9	2	13215039484889855	0	805306368	0	5635024	□

Fig. 6. The visits table from the Chrome browser showing not only the detail of the individual visits, but that the Google search results page and the subsequent pages visited are recorded.

```
def set_f_bios_time(vm_name, datetime_to_set):
    from datetime import datetime
    datetime_now = datetime.now()
    datetime_to_set =
           datetime.fromisoformat(datetime_to_set)
    time_delta =  datetime_now - datetime_to_set
    command = r'vboxmanage modifyvm
    %s --biossystemtimeoffset -%d'
    % (vm_name, time_delta.total_seconds()*1000)
    f = Popen(command, stdout=PIPE, shell=True).stdout
    data = [eachLine.strip() for eachLine in f]
    return data
```

Snippet 1. This method is to set the system time to a the given datetime

```
def launch_chrome():
    pywinauto.keyboard.send_keys('{VK_LWIN}')
    time.sleep(0.5)
    pywinauto.keyboard.send_keys('chrome')
    time.sleep(0.5)
    pywinauto.keyboard.send_keys('{ENTER}')
    time.sleep(0.5)
```

Snippet 2. Example usage of the pywinauto module to launch Chrome by opening the Start Menu and typing ‘Chrome’ followed by the enter key.

```
def open_link_x_for_y_secs(link_no, time_to_view_page):
    # requires sendkey extension in chrome

    # activate sendkey google result browser
    pywinauto.keyboard.send_keys('{TAB}')
    time.sleep(2)

    for i in range(0, link_no):
        pywinauto.keyboard.send_keys('{DOWN}') # select link
        time.sleep(0.1)

    pywinauto.keyboard.send_keys('{ENTER}') # open link
    time.sleep(time_to_view_page)

    pywinauto.keyboard.send_keys('{LEFT}') # go back
```

Snippet 3. Example usage of the SendKey browser plugin to allow keyboard control of search results. This allows automated opening of several links that result from a Google search.

```
app = Application().start("notepad.exe")
app.UntitledNotepad.Edit.type_keys(sys.argv[1],
                                   with_spaces = True)
app.UntitledNotepad.menu_select("File->Save As")
app.SaveAs.Edit.type_keys(sys.argv[2],
                                   with_spaces = True)
app.SaveAs.Save.click()
app.UntitledNotepad.menu_select("File->Exit")
```

Snippet 4. Example usage of the pywinauto module to generate a new notepad file in a location on disk with customised content.

previous one. This is however contrary to the proposal in (Moch and Freiling, 2009) that discusses reproducible randomness, which suggests that a two stage approach may be better where fixed-time scripts are produced by another script that generates the random element. However, the exact reproducible nature of the artefacts generated cannot be fully supported in this approach, e.g., in situations where web browsing is performed, it cannot be guaranteed that the content viewed (and cached) will be consistent from one run of the story to the next.

However, despite the limitations of the current proof-of-concept, this research has demonstrated that this approach as numerous advantages. If we consider the benefits of synthesised data sets in general, they are obviously free of the ethical concerns around distributing real data, although they may still have copyright issues 1. In the case of this work, we sidestep the copyright issues as we distribute user simulation software, and a set of stories that cause certain actions to be performed at certain times, enabling the organisation to generate their own images. The copyright issue is then the responsibility of the organisation running the scripts to ensure that their operating system used within the VM is correctly licensed.

The primary advantage of machine generated disk images are that a large amount of human time investment is not required, and this has been argued in previous work that focuses on injecting content into baseline disk images. The specific advantage offered in this work is a significant improvement in terms of the realism of the data generated. This approach has much more focus on metadata and its importance in event reconstruction, over injecting and hiding specific content. If we consider the Google search browsing example, it might be possible to inject all the correct entries into the relevant databases, JSON files, and inject the corresponding cached files into the file system, but it is extremely difficult and is highly sensitive to version changes in the application. The approach of emulating the user, if done with care, is guaranteed to produce all of the correct artefacts, and is less sensitive to version changes, although depending on the emulation technologies in use may be sensitive to GUI changes of any application automated.

This research provides potential benefits in three areas: education, research, and practitioner work. In education, disk images are necessary for use in practical exercises and assessments.

Production of these disk images is a time consuming and therefore an expensive process. Use of this method would allow realistic disk images to be created with far less investment. In research, particularly in areas where attempts are being made to use machine learning for artefact extraction and event reconstruction, having disk images with suitable 'background noise' that contains realistic data with all of the detailed metadata, and having a machine readable log that represents the ground-truth of actions carried out, would be extremely valuable. Finally, in the practitioner community, with new requirements to validate the results from tools, being able to quickly and easily generate disk images with known content and known event histories would allow more thorough, extensive, and reliable tool testing. In all three areas this approach would generate realistic images, and, importantly, with a known set of performed actions and data, i.e..,ground-truth.

6.1. Conclusion

While the work presented in this paper is a proof of concept, the research has shown that a better approach to automated disk image generation is possible, and that it could have significant benefits in the areas of digital forensic teaching, research, and tool and process validation. While there is still a long way to go, this research has highlighted the problems with the existing approaches, has proposed and tested a more 'user emulation' focused approach and shown the advantages in terms of realism and flexibility of the data that can be generated.

References

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

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

Anda, F., Le-Khac, N.A., Scanlon, M., 2020. DeepUAge: Improving Underage Age Estimation Accuracy to Aid CSEM Investigation. Forensic Sci. Int.: Digit. Invest. 32, 300921. https://doi.org/10.1016/j.fsidi.2020.300921. http://www.sciencedirect.com/science/article/pii/S2666281720300160.

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. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. ARES '20. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3407023.3407068.

Du, X., Le, Q., Scanlon, M., 2020. Automated Artefact Relevancy Determination from Artefact Metadata and Associated Timeline Events. The 6th IEEE International Conference on Cyber Security and Protection of Digital Services (Cyber Security). IEEE. https://doi.org/10.1109/CyberSecurity49315.2020.9138874.

Garfinkel, S., 2012. Lessons learned writing digital forensics tools and managing a 30tb digital evidence corpus. Digit. Invest. 9, S80–S89. https://doi.org/10.1016/j.diin.2012.05.002. http://www.sciencedirect.com/science/article/pii/S1742287612000278 (the Proceedings of the Twelfth Annual DFRWS Conference).

Garfinkel, S., Farrell, P., Roussev, V., Dinolt, G., 2009. Bringing science to digital forensics with standardized forensic corpora. Digit. Invest. 6, S2–S11.

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

Moch, C., Freiling, F.C., 2009. The forensic image generator generator (Forensig2). In: IT Security Incident Management and IT Forensics, 2009. IMF'09. Fifth International Conference on. IEEE, pp. 78–93.

Moch, C., Freiling, F.C., 2011. Evaluating the Forensic Image Generator Generator. In: International Conference on Digital Forensics and Cyber Crime. Springer, pp. 238–252.

National Institute of Standards and Technology, 2019. Computer Forensic Reference Data Sets (CFReDS). https://www.cfreds.nist.gov/. (Accessed 14 October 2019).

Nemetz, S., Schmitt, S., Freiling, F., 2018. A standardized corpus for sqlite database forensics. Digit. Invest. 24, S121–S130. https://doi.org/10.1016/j.diin.2018.01.015. http://www.sciencedirect.com/science/article/pii/S1742287618300471.

Sanchez, L., Grajeda, C., Baggili, I., Hall, C., 2019. A practitioner survey exploring the value of forensic tools, AI, filtering, & safer presentation for investigating child sexual abuse material (CSAM). Digit. Invest. 29, S124–S142. https://doi.org/10.1016/j.diin.2019.04.005. http://www.sciencedirect.com/science/article/pii/S1742287619301549.

Scanlon, M., Du, X., Lillis, D., 2017. EviPlant: An Efficient Digital Forensic Challenge Creation, Manipulation, and Distribution Solution. Digit. Invest. 20, S29–S36. https://doi.org/10.1016/j.diin.2017.01.010. http://www.sciencedirect.com/science/article/pii/S1742287617300397. DFRWS 2017 Europe.

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

Yannikos, Y., Graner, L., Steinebach, M., Winter, C., 2014. Data corpora for digital forensics education and research. In: IFIP International Conference on Digital Forensics. Springer, pp. 309–325.

Yeh, T., Chang, T.H., Sikuli, Miller R.C., 2009. Using GUI screenshots for search and automation. In: Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology. UIST '09. ACM, New York, NY, USA, ISBN 978-1-60558-745-5, pp. 183–192. https://doi.org/10.1145/1622176.1622213.