Digital Investigation 26 (2018) 3–18

Contents lists available at ScienceDirect

Digital Investigation

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

# TREDE and VMPOP: Cultivating multi-purpose datasets for digital forensics – A Windows registry corpus as an example☆

Jungheum Park [1]

*Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, 20899, USA*

ARTICLE INFO

*Article history:*
Received 1 December 2017
Received in revised form
23 March 2018
Accepted 26 April 2018
Available online 28 April 2018

*Keywords:*
Forensic infrastructure
Dataset
Data corpora
Synthetic data
Reference data
Automated data generation

ABSTRACT

The demand is rising for publicly available datasets to support studying emerging technologies, performing tool testing, detecting incorrect implementations, and also ensuring the reliability of security and digital forensics related knowledge. While a variety of data is being created on a day-to-day basis in; security, forensics and incident response labs, the created data is often not practical to use or has other limitations. In this situation, a variety of researchers, practitioners and research projects have released valuable datasets acquired from computer systems or digital devices used by actual users or are generated during research activities. Nevertheless, there is still a significant lack of reference data for supporting a range of purposes, and there is also a need to increase the number of publicly available testbeds as well as to improve verifiability as 'reference' data. Although existing datasets are useful and valuable, some of them have critical limitations on the verifiability if they are acquired or created without ground truth data. This paper introduces a practical methodology to develop synthetic reference datasets in the field of security and digital forensics. This work's proposal divides the steps for generating a synthetic corpus into two different classes: user-generated and system-generated reference data. In addition, this paper presents a novel framework to assist the development of system-generated data along with a virtualization system and elaborate automated virtual machine control, and then proceeds to perform a proof-of-concept implementation. Finally, this work demonstrates that the proposed concepts are feasible and effective through practical deployment and then evaluate its potential values.


## Introduction

In recent years, the emergence and propagation of a broad range of information technologies are exploding, and they are being widely used for various purposes in our daily lives.

In a general accepted point of view, most software programs running on ICT (Information and Communications Technology) products are being imperfectly developed, and even when different developers implement program codes for identical operations, resultant codes and flows can be quite diverse. This is a result of a variety of factors such as the developers' level of skills and knowledge. These factors are more likely to cause serious issues in the field of information forensics and security, especially if software has potential vulnerabilities, errors or incorrect codes. For that reason, there have been demands for plausible mitigation strategies against the issues. As an example of such efforts, the digital forensics community has been trying to perform testing and validating different forensic tools, in order for examining if certain requirements for each tool are properly implemented, choosing a most suitable set among tools providing same functionalities, and encouraging continuous improvement.

In this circumstance, data corpora will play an important role in activities to meet the above-mentioned demands, if datasets are developed upon consideration of various use cases along with normal as well as abnormal user behaviors relating to each tool. That is, meaningful uses of these datasets may include; research, development, education, training, equipment check out, tool testing, and proficiency testing. Therefore, as a part of establishing a solid and fully functioning infrastructure, it is necessary to cultivate fine-grained datasets that can be used to support both academic and practical purposes. Although the necessity of systematically

☆ Certain trade names and company products are mentioned in the text or identified. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.

*E-mail addresses:* junghmi@gmail.com, jungheum.park@nist.gov.

[1] The author is a visiting scientist at the National Institute of Standards and Technology.

https://doi.org/10.1016/j.diin.2018.04.025
1742-2876/

developing reference data has been increasing, activities in the field of security and forensics are still hobbled by the lack of available datasets since Garfinkel stated in 2007 (Garfinkel, 2007; Grajeda et al., 2017). Moreover, existing datasets were usually created at that point in time for particular purposes, and then they have rarely been updated (Grajeda et al., 2017). Continuous endeavors are required to develop various data corpora that also embrace emerging technologies, not an easy mission but necessary.

The primary purpose of this work is to develop a systematic and practical methodology to improve the efficiency of dataset development, through providing fundamental techniques as an infrastructure. The key idea behind the proposed strategy is that forensic community demands a fundamental infrastructure for multi-purpose dataset generation, including practically usable procedures and publicly available assistance tools. In this context, this work devises a simple but efficient strategy that can be used as a guideline for corpus creators, by dividing the steps for generating a synthetic dataset into two classes: *user-generated* and *system-generated* reference data. The former consists of normal and experimental data supported by the specification of a target data, and it can contain unrealistic data beyond pre-defined limitations of the target. The latter is automatically generated by target systems such as operating systems or applications, so corpus creators are required to use relevant hardware or software to obtain system-generated data through mimicking user behaviors based on realistic scenarios.

With the above-mentioned procedural proposal, this paper presents a framework to assist the development of system-generated data by using a type-2 hypervisor and elaborate virtual machine automation. The framework is designed to maximize the scope of automatable parts, because the automatic approach has great advantages in regards of repeatability, reproducibility and traceability. As a result, the framework allows elaborate population of operating systems or user applications and the creation of logs that can be used to verify ground truth. This work proposes a detailed design concept of this framework, and then performs a proof-of-concept implementation.

Subsequently, this paper describes a Windows registry dataset as an example deployment of the proposed methodology. The dataset includes four primary categories of user-generated reference data as well as system-generated reference data extracted from six different versions of Windows from Vista to 10. Finally, this work demonstrates that the proposal is feasible through assessment of the dataset.

In summary, this paper makes the following contributions:

- This work introduces a systematic and practical methodology to improve the efficiency of dataset development in the field of security and digital forensics. The proposed methodology divides the steps for developing a synthetic corpus into two classes, and the two-class development strategy makes the data corpus applicable to multiple purposes.
- This work presents an automated framework to assist the efficient synthetic reference data development by using a type-2 hypervisor, and releases a Python package as a probing implementation to the public.
- This work demonstrates that the proposed concepts are feasible and effective, through a reference Windows registry dataset as a practical deployment, and then evaluates potential values of it.

The remainder of this paper is structured as follows: Section Related works summarizes existing studies relating to corpora that are used in security and forensics communities. Section Research objectives presents research objectives, and Section TREDE: Two-class Reference Data Development proposes a methodology for cultivating multi-purpose datasets. Section VMPOP: Virtual Machine Population Framework explains an automated system for supporting the creation of system-generated data. Section Deployment: cfreds-2017-winreg introduces a sample dataset developed based on the proposed strategy. Section Evaluation evaluates the efficiency and potential value of the proposal. Finally, Section Discussion and Conclusion and Future Direction present discussions and conclusions.

## Related works

In the field of security and digital forensics, several data corpora have been generated and made publicly available for use in research, development, education and tool testing. This section summarizes relevant data corpora for explaining this work's proposal. Prior to that, the following terms are defined based on the existing taxonomy proposed by Garfinkel (Garfinkel et al., 2009):

- **Real Data** – Data created by actual human beings. It is often subject to certain restrictions because of privacy or copyright concerns. Although real data is useful for evaluating effectiveness of forensic tools or techniques, it is not suitable for various purposes (e.g., tool testing) because of a lack of reliable ground truth.
- **Synthetic Data** – Artificially generated data for specific purposes, including but not limited to research, practice, education and tool validation.
  - **Synthetic Test Data** – If the purpose of generating a synthetic corpus is to test specific features in a group of tools, the generated corpus along with ground truth data will be synthetic test data for tool testing.
  - **Synthetic but Realistic Data** – If the purpose of data generation is to establish a situation where a forensic examiner might encounter in an investigation, corpus creators can generate synthetic but realistic data that is typically associated with widely used OSes and applications.
  - **Reference Data** – All types of synthetic data may also be called reference data when they provide detailed information including ground truth data and related annotations available for reference.

### *Existing data corpora*

Regarding real data corpora, Garfinkel et al. have created the Real Data Corpus (RDC) and Govdocs1 from real-world data (Garfinkel et al., 2009). The RDC is a collection of raw data extracted from data-carrying devices that were purchased on the secondary market around the world. Govdocs1 is a corpus of 1 million documents (txt, html, jpg, doc, pdf, etc.) downloaded from web servers in.gov domain using well-known search engines (Digital Corpora, 2018). Although Govdocs1 is freely available, accessing to the RDC requires Institutional Review Board (IRB) approval.

Since generating and sharing real data corpora are not simple tasks in terms of privacy and copyright issues, researchers and practitioners have been trying to create synthetic data for their needs. Of course, synthetic corpora may also be exposed to copyright or other issues, but these issues can be addressed by following proper redaction processes.

At present, examples of publicly available synthetic corpora including realistic and test datasets are; Carrier's Digital Forensic Tool Testing (DFTT) and DigitalCorpora.org. The DFTT project provides several image files for small and simple test cases (Carrier, 2018) and DigitalCorpora.org that has been releasing synthetic corpora such as `nps-2008-m57-jean`, `nps-2009-m57-patent` and `nps-2014-xbox1` (Garfinkel et al., 2009; Woods et al., 2011).

Among them, `m57-patent` is a relatively large synthetic dataset that consists of multiple storage images, memory dumps and network packets acquired during the generation process within a 17-day period. This corpus was created by researchers' largely manual efforts although a few actions were performed using automated scripts. When engaging in memory analysis, the Mem-Corp is a corpus that consists of memory image files acquired from physical and virtual machines (Vidas, 2011). The National Institute of Standards and Technology (NIST) has also been developing and sharing documented datasets of simulated digital evidence through the Computer Forensic Reference Data Sets (CFReDS) project (NIST, 2018). In addition, challenges and competitions from conferences (or organizations) associated with security and digital forensics have provided interesting data that have the potential to be utilized for research and education.

Outside of the most relevant corpora mentioned above, there are a lot of datasets created for specific purposes in various academic fields. For example, some of them are e-mails, malware, chat logs and multimedia (image, audio and video) (Grajeda et al., 2017).

An important note is that there is obviously a mismatch between supply and demand. Where demands on multi-purpose datasets are rising, there have been limited attempts to generate synthetic datasets systematically.

*Studies on automated synthetic corpora generation*

Moch et al. presented a system (Forensig$^2$) that produces file system images for training courses. With this system, users can program certain user actions (like creating, downloading, copying and deleting files) in a script, which is executed by the system using a combination of Python and QEMU virtual environment (Moch and Freiling, 2009). The authors established a requirement of the generator system that would enable users to program random behaviors in order to create different (but similar) disk images.

Another study also focused on developing an automated process to produce disk images. Russell et al. proposed the Summarized Forensic XML (SFX) that is an XML-based language for specifying scenarios including simple actions like file system operations, and then an SFX file can be processed by their image generation application to create image files (Russell et al., 2012).

To support efficient image creation, Flandrin designed Digital Forensics Image Creation (D-FIC) to create forensic test images by leveraging features and services provided by a cloud infrastructure to control virtual machines. Flandrin also implemented several tools to automate not only cloning, starting and shut downing virtual machines, but also copying files into the machines to create user data for validation. Although the approach did not consider various user activities, it was meaningful in terms of trying to automate image creation processes by using a virtualization system (Flandrin, 2012).

Next, Yannikos et al. proposed a stochastic process-based generation using finite discrete-time Markov chains in order to build models of real-world scenarios. To construct a model, it is necessary to first define actions (states) such as creating, writing, downloading and deleting files. Then, the probability of each action (state probability) is specified along with the probability of each transition between two actions (transition probability); the probabilities are used during the Markov chain simulation to generate synthetic data (Yannikos et al., 2012, 2014; Yannikos and Winter, 2013). Thus, this framework allows creating different disk images for identical scenarios without requiring additional programming.

In addition, there exist open source projects associated with the automated synthetic data generation. ForGe (forensic test image generator) is a Python-based tool to rapidly create simple test images. More specifically, the tool can create image files with NTFS and FAT file systems, and it supports several data hiding methods including; alternative data streams, extension change, concatenation of files, and slack space. It also provides means of creating variances between images through selecting files randomly from a central repository (ForGe, 2018). Another tool, ForGeOSI, was designed to automate disk image generation with the VirtualBox hypervisor. It simplifies the creation of virtual machines, and the automation of them that supports Windows and Linux operating systems. The tool actually uses APIs provided by VirtualBox in order to handle guest systems running inside the hypervisor. Therefore, users can automate various actions with a virtual machine such as starting the VM, executing processes, sending keyboard strokes and copying files (ForGeOSI, 2018). One of this work's results presented in Section VMPOP: Virtual Machine Population Framework was partly influenced by the ForGeOSI in terms of utilizing a virtual environment to automate the data generation.

*Other relevant approaches*

Scanlon et al. proposed EviPlant, a system designed for the creation, manipulation and distribution of forensic corpora. The fundamental premise is that a base image can be downloaded once, and challenges can then be distributed as evidence packages, which consist of the modified artifacts and associated metadata by performing a diffing of resultant image and the base image (Scanlon et al., 2017). Although their proposal is not closely related to automating the data generation, the authors provided an interesting concept to make the creation of digital forensic challenges easier, especially by enabling efficient storage and distribution of the differences identified from the diffing tool.

## Research objectives

Despite the above-mentioned efforts to enrich data corpora, there is still a lack of available datasets as well as demand for efficient synthetic data generation in the field of security and digital forensics. This work was started to reduce the gap between demand and supply.

*Common pitfalls in data corpora*

Before explaining how to approach the research subject, common inevitable pitfalls immanent in data corpora will be presented as follows:

- **Limited scope** – As mentioned above, datasets are useful when they are utilized for intended purposes. For example, a specific corpus created under consideration of various types of data fragmentation would be valuable for tool testing or education about data carving techniques. However, it may not be useful beyond its intended scope based on the infrastructure it used or the type of activity it recorded. When a dataset is created, available resources are typically used like operating systems, user applications and sample files. Many datasets do not include sophisticated user actions, needing the graphical user interface (GUI) since these are more complex and difficult to create (Russell et al., 2012; Yannikos et al., 2014; Moch and Freiling, 2011).
- **Outdated data** – Even though a corpus is generated using the latest technology, it will inevitably be outdated soon. An easy example is a corpus relating to operating systems and user applications. Such software installed in both desktop and embedded devices can be updated very frequently, so then the data contained in the corpus created earlier will lose its relevance to the present (Yannikos et al., 2014). Another example

can be found in network related data, because newly rising network protocols and attack techniques also need to be considered.

- **Insufficient verifiability** – Though most open data corpora provide the ground truth about the sequence of operations and timestamps, it is hard to include a complete set of information to support effective and sufficient verification. For example, a corpus may need to be organized with accurate logs and timelines of performed operations with as much detail as possible. Furthermore, if the overall generation process of a corpus can be replayed within standardized environments, e.g., tools (or scripts) developed for automated data generation, it may help enhance the verifiability of the corpus.
- **Legal issues** – A corpus is generally subject to intellectual property and privacy law (Garfinkel et al., 2009; Yannikos et al., 2014). Generally, the privacy issues are found in real data corpora, but the copyright issue may be a common subject of both real and synthetic data. Therefore, after creating a synthetic data corpus, it is necessary to perform proper redacting for items such as license keys.
- **Distribution issues** – If a corpus is to be open to the public, there are some considerations that need to be addressed before the distribution. The first issue is about how to store and manage data. Standardized formats for images and logs will be useful for efficiency of sharing data corpora and related resources. The second issue is the total size of a corpus. Very large datasets may need to be reduced in size if it needs internet-based distribution mechanisms.

### *Strengthening forensic corpora*

The primary aim of this work is to develop a systematic methodology for improving the utility and availability of forensic corpora through providing fundamental techniques to mitigate some of pitfalls mentioned above. To achieve this aim, the following four research objectives will be considered in the rest of this paper:

- Establishing an efficient synthetic corpus generation methodology for multiple purposes including research, development, education, practice and tool testing.
- Developing automated methods or assistance systems to help extend the scope of data corpora and reflect the latest trends whenever it is needed.
- Providing a design concept for producing the sequence of operations. To maximize verifiability, a corpus should consist of detailed specifications, descriptions and logs.
- To support digital forensic activities, applying the proposed methodology to a meaningful target as a practical example.

## TREDE: two-class reference data development

### *Overview*

To address objectives established in the previous section, this paper proposes TREDE, which is a hybrid strategy of two classes: *user-generated* and *system-generated reference data*. Here, TREDE stands for Two-class REference Data dEvelopment.

The first class is the user-generated reference data (UGRD), which is created experimentally based on the specification of file systems or data formats. The UGRD consists of specially crafted data that are prepared by corpus creators, so it usually contains plausible types of data officially supported by its specification. It might also be composed of unrealistic and implausible data beyond the limitations of a pre-defined format. Therefore, this UGRD is useful for education to understand detailed internals of specific file systems or storage formats. Moreover, it will be valuable for validation of associated algorithms, detection of potential vulnerabilities, and tool testing – especially verifying fault tolerance on exceptions and possible anti-forensic activities.

The second class is the system-generated reference data (SGRD), which is generated naturally by target systems like operating systems, user applications or online services. That is, corpus creators need to use relevant hardware or software (along with network connections if necessary), in order to generate this type of synthetic data. The SGRD can be constructed by mimicking actual user behaviors, i.e., realistic scenarios are required to perform it. For example, assuming that a target of generation is the SQLite database format, various software such as web-browsers and mobile apps can be utilized to obtain a set of meaningful SGRD. Therefore, SGRD greatly values for multiple purposes including research, development, education, practice, and tool testing as well.

The detailed steps will be described in the following subsections. The procedures proposed here could be applied to a variety of generation targets including but not limited to a file format, artifacts of an operating system, and traces relating to a specific application or service. For more information about this section's proposal, Section Deployment: cfreds-2017-winreg provides a practical example of synthetic corpus (a Windows registry dataset) development based on TREDE.

Note that the results of this work include procedural and technical approaches to overcome shortcomings mentioned in existing studies, and to support the needs for datasets available for multiple purposes. Even if this work's proposal is not the only solution, it would be a forward step toward balancing demand with supply of data corpora.

### *Manual vs. automated corpus generation*

When planning a corpus generation, corpus creators should consider two different ways in order to perform user actions to generate synthetic data. The first and simplest way is to create them by hand. This *manual* approach will be useful for the sophisticated control performed by experts who are familiar with the field of security and digital forensics. This approach, however, has limitations with respect to repeatability, reproducibility and traceability. When third parties need to repeat the data generation process and reproduce its results, it is often very difficult to achieve their goals if very detailed actions were performed manually at the original population process. It is also hard to manage events triggered by user actions including a sequence of detailed operations and their timestamps, so this approach has limited traceability if there is no additional effort to monitor and record all the events.

The second approach, *automated,* can be applied to create synthetic data when most parts of the generation process are automatable. This approach, of course, may require a lot of endeavors to develop assistant tools (or scripts) for the efficient automation. For example, if a scenario for the generation is too complicated, it may be not possible to automate depicting actual users' behaviors. If the automated approach is available, it has advantages that support repeatability and reproducibility by providing detailed automation methods, and also make the generation process traceable by developing dedicated logging modules. These advantages will enhance the completeness and usefulness of data corpora as reference data.

### *Towards fully and semi-automated generation*

Regarding the above-mentioned two generation approaches, this work's proposal is to promote the sustainable development and propagation of automated data generation methods and assistance tools. Even though a sequence of operations is difficult to

be fully automatable, it is still possible to develop semi-automated methods by minimization of manual operations and maximization of automated parts. That may be one way for strengthening development of data corpora.

*TREDE workflow*

As shown in Fig. 1, the initial step to obtain a data corpus is to determine clear purposes and specific targets for the generation process. Here, the target can be any interesting data in the field of security and digital forensics. It usually includes formatted files, network packet capture files, memory dump files, and disk images having operating systems. If a file format as the primary target is closely related to specific operating systems or user applications, these software programs should be considered as target systems of the following dataset generation steps.

An important aspect of this first step is to determine which classes of TREDE are available and required as a dataset, i.e., UGRD or SGRD or both of them. For example, assume that the primary target is a database storage format. In this case, it will be possible to consider both UGRD (created based on technical specifications for the format and related database engines) and SGRD (generated by a variety of user applications using the database).

*Steps for user-generated reference data (UGRD)*

1 **Defining requirements** – In this step, common requirements for the actual generation tasks are defined based on the determined purposes and target data (or systems). A set of requirements usually includes certain conditions of data to be generated, and limits the scope for practical generation processes. It may also consist of various considerations about fundamental limitations intrinsic in any specifications of the target. In practical applications, requirements can be defined broadly or narrowly according to the purpose of generating data. Thus, all the requirements established here will be effective to limit the scope of generation, and especially valuable for providing basic conditions to plan the following steps.
2 **Setting up execution environments** – The next step is to set up execution environments, which usually include operating systems, applications, external devices, hardware tools, and other associated resources. This step requires some selections to make the following steps performable: for instance, which environment will be used for establishing reference systems among the physical and virtual environment, which types of operating systems or user applications will be targeted, etc. It is important to note that this step is deeply associated with the actual generation steps, so the execution environments need to be built clearly and of course it should be continuously enhanced by feedbacks raised during the generation process.
3 **Performing categorization** – To maximize the usefulness and availability as UGRD, this step normalizes various types of data into several categories. Because UGRD consist of specially crafted data for including both normal and experimental values, it is necessary to consider forensically meaningful characteristics based on the target data's specification. For doing that, corpus creators first need to define certain categories. For example, assuming that the target data is a well-known file format, it may include normal file, normal file with deleted data, corrupted file, manipulated file, etc. Here, a category can comprise multiple classes, which are usually associated with internal components of the data format or user actions relating to the category. Note that a class can also contain multiple sub-classes for clearly representing its scope detail.
4 **Developing generation methods** – After the categorization, the next step is to design and implement clear data generation methods for each UGRD class. One or more generation methods for each class need to be listed along with detailed processes or algorithms. Here, fully automatable methods can be written in code and thus they can also be executed as assistance tools in the next step. A meaningful contribution of these assistance tools is to make data generation methods shareable for the demand of verification and customization. In addition, in the case of classes that require the semi-automated generation, it is necessary to provide specific manual operations (with figures or screenshots if possible) to make the last generation step more efficient.
5 **Executing tools and operations** – In this step, UGRD of a corpus will be created based on established generation methods. The operations of this step include executing assistance tools and manual operations, as well as managing outputs as a fine-grained dataset. Therefore, for the efficient execution, it will be better for corpus creators to prepare additional tools for convenient integrated data generation. For example, in a use case described in Section Deployment: cfreds-2017-winreg, several shell scripts were developed for integrated fully or semi-automated executions along with multiple assistance tools in Windows and Linux environments.

*Steps for system-generated reference data (SGRD)*

1 **Defining requirements** – This step is essentially the same with the first step in Section Steps for User-generated Reference Data (UGRD). Additionally, corpus creators need to have a deep understanding of the operations, features and forensic

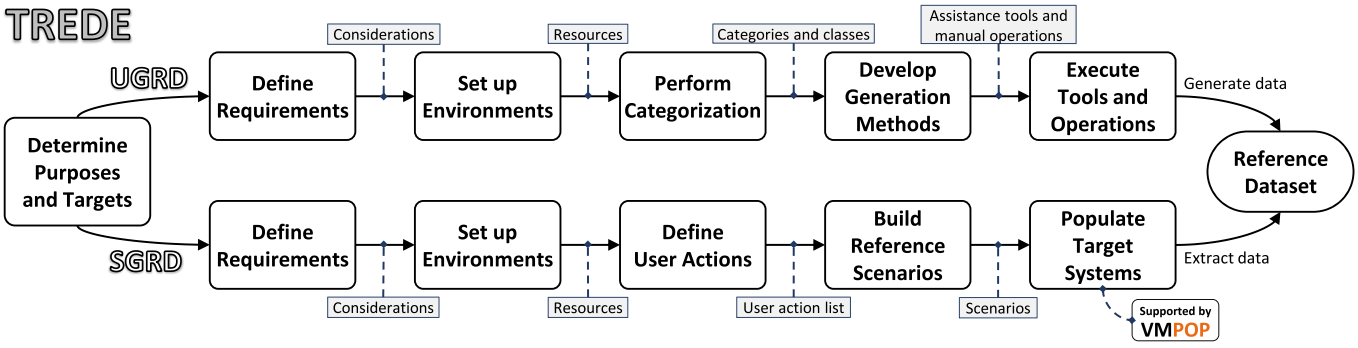


**Fig. 1.** TREDE workflow.

implications on target software (e.g., operating systems and user applications) or online services.

2 **Setting up execution environments** – This step is the same with the second step of UGRD in Section Steps for User-generated Reference Data (UGRD).

3 **Defining user actions** – In this step, corpus creators define any possible user actions that may produce significant effects on resultant data. Because it is closely related with the next step, each user action should be performed without any issues on at least one or more target systems. In this context, the user actions defined here highly depend upon several conditions, including generation purposes, characteristics of target systems, available resources, as well as experience and knowledge of corpus creators. For instance, assuming that the targets of a SGRD generation include various versions of Windows OSes, basic actions such as installing/uninstalling programs, creating/terminating processes and attaching/detaching external storage devices will be viable regardless of versions. On the other hand, actions relating to Windows Store applications will only be valid on specific versions of Windows.

4 **Building reference scenarios** – Here, one or more scenarios are built for systematically populating target systems. A reference scenario is composed of a series of user actions that are defined in the previous step. For the efficient functioning of the next population step, the scenario should contain detailed annotations to perform each user action realistically like by an actual user. For example, an annotation may include descriptions of various elements, such as directions for execution of each action (e.g., shell script or a sequence of keystrokes), account information, applications, sample files, and other parameters affecting the execution.

5 **Populating target systems** – After that, target systems (operating systems or user applications) are populated according to the reference scenarios, and then interesting data (as SGRD) are extracted from the populated systems. So, it is required to select appropriate population strategies with consideration for the execution environments. Like mentioned above, a fully automated approach will provide valuable advantages in terms of enhancement of the repeatability, reproducibility and traceability. In this context, Section VMPOP: Virtual Machine Population Framework will propose a practical population framework based on virtualization and automation techniques to generate SGRD. Note that the data extraction process will be executed according to corpus creators' needs for extracting some particular parts from populated systems, i.e., a specific range of sectors, files having specific file formats, files within specific paths, byte streams inside a file, etc.

## VMPOP: virtual machine population framework

### *Overview*

To assist the SGRD generation process, this work developed a new system named VMPOP: Virtual Machine POPulation system. The VMPOP is an automation framework designed for populating operating systems or applications, and extracting forensically interesting data from populated systems. The main purpose of this system is to facilitate the efficient execution of the actual data generation steps as highlighted in Fig. 1.

The following sub-sections propose a design concept of the VMPOP and introduce a proof-of-concept implementation of the proposed design. Note that the current release supports only Windows systems to show an example presented in Section Deployment: cfreds-2017-winreg, but it is customizable and expandable to automate other OSes.

### *Design*

The VMPOP is composed of five subsystems in order to support virtual machine automation. Fig. 2 shows components of the framework and their structural design concept.

*HIS: hypervisor interface subsystem*

The hypervisor interface subsystem (HIS) is a component of the VMPOP for communicating with hypervisors that controls virtual machines. This subsystem is composed of multiple virtual machine interface modules (VIM), and each VIM should be implemented for interfacing other subsystems with a specific hypervisor such as Oracle VM VirtualBox, VMware products, KVM and QEMU. That is, each VIM needs to be designed upon consideration of a variety of features including but not limited to; setting up registered virtual machines' preferences, starting up/shutting down the machines, and executing processes. These controllable features can be identified through understanding native application programming interfaces (API) supported by each hypervisor.

As shown in Fig. 3, a VIM of the HIS is in charge of a bridge for other subsystems to handle virtual machines supervised by a hypervisor. To build a solid bridge for efficiently transceiving requests and responses inside the framework, the hypervisor should support various APIs natively for controlling virtual machines, and each virtual machine (VM) should have a special program called typically guest agent. For user convenience, common hypervisors usually provide their own agent (e.g., VirtualBox Guest Additions).

The role of the HIS is to deliver requests derived from the OS automation subsystem (OAS) and event monitoring subsystem

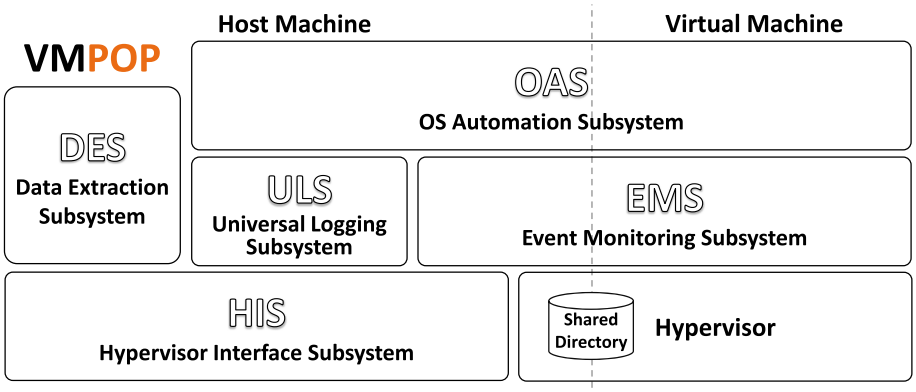


**Fig. 2.** Design concept of the VMPOP.

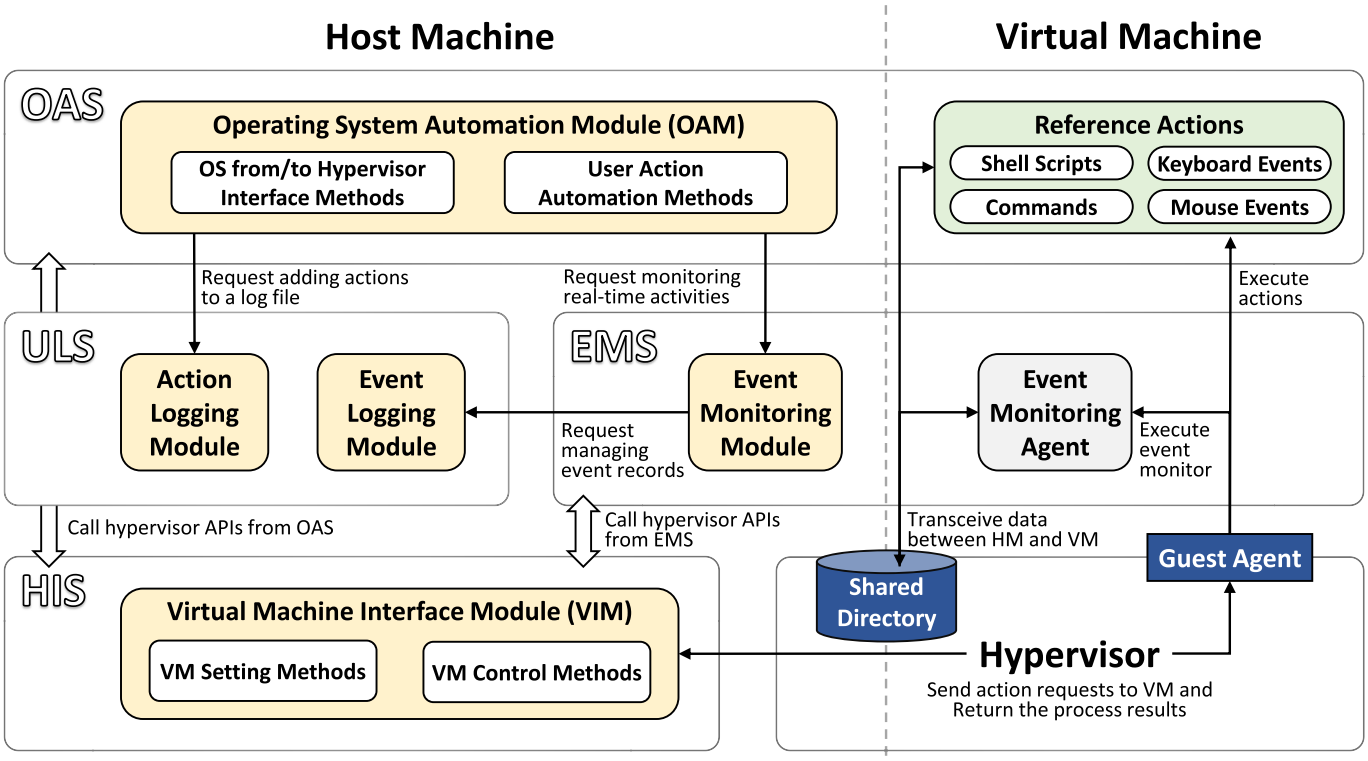


**Fig. 3.** Simplified processes inside the VMPOP.

(EMS). To achieve this role of a messenger, each VIM should include interface methods for setting and controlling virtual machines. This is done by using hypervisor APIs in order to send requests to VMs and receive the processing results from them. As an example of these interface methods, a shared directory can be configured to support convenient data sharing between the host and virtual environment as depicted in the Hypervisor part of Fig. 3.

*OAS: operating system automation subsystem*

The OS automation subsystem (OAS) is a component that provides user-friendly automation methods for automating user actions inside the target virtual machine. This subsystem consists of multiple operating system automation modules (OAM), and each OAM should be prepared to control an operating system like Windows and Linux. Therefore, the OAM needs to be implemented upon consideration of various characteristics of operating systems including multi-user systems, available commands, supported shell scripts, keyboard shortcuts and methods for handling processes.

In the VMPOP, actions implemented within the OAM are executed as reference actions inside the virtual environment as depicted in Fig. 3. For this process, automation methods within the OAM should be developed by calling hypervisor APIs and accessing a shared directory to move files between separated environments. There are various approaches to perform actions automatically in a virtual machine. Examples of them include, but not limited to executing default commands, running shell scripts, and sending keyboard/mouse events.

When implementing this subsystem, each automation method within the OAM may need to wait certain time for ensuring the completion of the current action before executing the next one. A simple idea to solve this issue may be adding a short or long time delay, but sometimes it will not be enough to cover cases where an action does not provide easily noticeable changes when it is completed. Those cases are usually related to specific automation methods depending on keyboard/mouse events. Therefore, it is necessary to establish proper waiting and checking policies according to the characteristics of each operating system and application. Section Automation Challenges will discuss about this issue as one of automation challenges during this work.

Meanwhile, before requests for user actions are sent by the OAS, the framework allows users to request monitoring on all events that occur during the execution of actual actions. The monitoring request is then processed by the event monitoring subsystem (EMS). In addition to this, there is a supplementary component, the universal logging subsystem (ULS), which provides logging modules for saving detailed action and event logs with timestamps. The role of the ULS is to process requests for logging from the OAS and EMS.

*EMS: event monitoring subsystem*

The event monitoring subsystem (EMS) is an optional but useful component for monitoring events that occur during the execution of actions as well as for saving the monitored event records. Although this subsystem is optional, it will be valuable for purposes on generating reference data associated with the field of security, forensics and incident response. It is because the VMPOP along with the EMS can provide ground truth data for enhancing applicability as reference data.

When the EMM receives a request for monitoring, it sends a request for executing an event monitoring agent to the target virtual machine through the HIS as shown in Fig. 3. The event monitoring agent used in this context is an application that shows real–time activities of system components such as file system, process, network and registry. Here, the monitoring agent can be developed personally to support user's special needs, or it also can be selected among existing monitoring applications like Process Monitor by Windows Sysinternals.

After executing a monitoring agent in the target virtual environment, one or more reference actions may be performed by consecutive requests from the OAS. When all the actions are completed, the OAS will send a request for terminating the active monitoring agent to the target VM through the EMS. At this moment, the monitoring result will be accessible from the shared directory between the host and virtual environment.

It should be noted that the EMS may not be required for situations, where corpus creators already know resultant artifacts changed by an action, or where the generated dataset has no connection with system event changes during the execution of actions. For those cases, this subsystem can simply be disabled so that VMPOP does not execute an event monitoring agent, and this will reduce the time of overall operations by eliminating additional processing and waiting time.

*ULS: universal logging subsystem*

The role of the universal logging subsystem (ULS) is to provide logging modules for managing detailed action and event records with timestamps. This subsystem consists of an action logging module (ALM) and an event logging module (ELM) to process requests from the OAS and EMS respectively.

First, the ALM manages behavioral information relating to user actions, which are triggered by the OAS. In detail, a request for the action logging can occur before, during or shortly after a user action, and then the request is delivered to the ALM for the integrated management and reporting.

In addition, the ELM requests managing event records that are monitored by an event monitoring agent within the virtual environment. When the monitoring agent is terminated after the execution of one or more actions, the ELM can access the monitored records from the shared directory. Afterward, the EML will try to deal with the event logs by using a configured management method such as converting a log file to a CSV file or storing the records into a normalized database.

*DES: data extraction subsystem*

The data extraction subsystem (DES) is an independent component from the other ones of the VMPOP. As shown in Fig. 4, the DES is in charge of extracting data selectively from image files. The use of the DES is not mandatory, if the purpose is to obtain reference data in the form of a disk image.

To utilize the DES, it is first necessary to prepare image files that can be identified by the image handler of the subsystem. It can be done by using an exporting method of the HIS, which is implemented to export the current snapshot of the target virtual machine as a well-known image file format including but not limited to VHD, VMDK, VDI and DD. When the target virtual machine's image file is prepared, it is entered into the DES as a primary input for the data extraction process. After that, the image handler will try to identify all volumes and file systems, and at this moment, it is possible to access and extract interesting data (i.e., sectors, clusters or files).

**Table 1**
Methods of the HIS for VirtualBox hypervisor.

| Method | Description |
|---|---|
| attach_usb_device() | Attach/Detach a device using a serial number |
| connect_to_vm() | Connect to a virtual machine |
| create_shared_dir() | Create a shared directory |
| create_user_session() | Manage the current session |
| dump_physical_memory() | Dump the physical memory to a file |
| execute_process() | Execute a process with arguments |
| export_disk() | Export a virtual disk to an image file |
| send_event_keyboard() | Send keyboard events |
| start_packet_capturing() | Start/Stop the packet capturing |
| start_video_capturing() | Start/Stop the video capturing |
| start_vm(), stop_vm() | Start/Stop a virtual machine |

*Implementation*

The *pyvmpop* is a Python package developed for automating the virtual machine population based on the proposed design concept for VMPOP.

In the current version of *pyvmpop*, the HIS that communicates with hypervisors was initially implemented with a well-known hypervisor, Oracle VM VirtualBox, which provides useful APIs that can be used for implementing the VMPOP. Note that the current

**Table 2**
Methods of the OAS for Windows.

| Method | Description |
|---|---|
| add_email_accoun() | Add an existing Microsoft e-mail account |
| add_local_account() | Add a new local account |
| attach_usb(), detach_usb() | Attach/Detach a device using a serial number |
| change_timezone() | Change the time zone |
| close_window() | Manage the current window |
| configure_eventlog() | Configure the EventLog settings |
| configure_nic_ip() | Configure IP settings of a network interface |
| connect_network_drive() | Connect a network directory |
| connect_remote_desktop() | Connect to/Disconnect from a remote desktop |
| control_web_browser() | Control web browsers |
| copy_files() | Copy files or directories |
| create_restore_point() | Create a restore point (Volume Shadow Copy) |
| create_virtual_desktop() | Create/Close a virtual desktop |
| delete_account() | Delete/Change an existing account |
| disable_uac(), enable_uac() | Disable/Enable User Account Control |
| disable_windows_update() | Disable the Windows update function |
| enable_file_history() | Enable/Configure the File History feature |
| execute_script() | Execute a PowerShell/Batch script |
| get_time(), set_time() | Get/Set the date and time |
| install_program() | Install/Uninstall a program with arguments |
| launch_program() | Launch a program with arguments |
| logon_account() | Logon with an existing account |
| map_network_drive() | Map/Unmap a network drive as a local drive |
| open_shell(), change_dirs() | Utilize the default shell (explorer.exe) |
| set_foreground_window() | Set/Get the foreground window |
| set_reg_value() | Set a registry value |
| share_directory() | Share a directory |
| shutdown(), restart() | Shutdown/Restart the current system |
| terminate_process() | Terminate a process |
| wait_for_idle() | Wait until the system goes to an idle state |

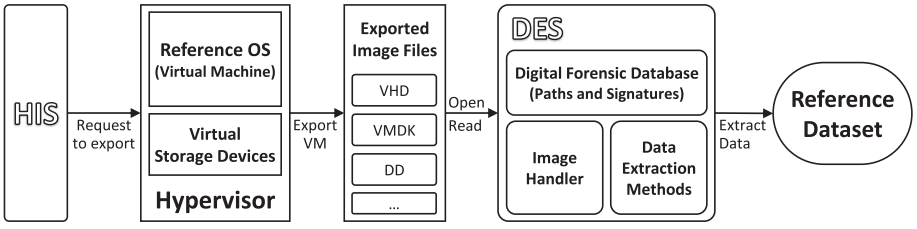


**Fig. 4.** Event flows of the data extraction subsystem (DES) inside the VMPOP.

implementation was tested with VirtualBox v5.1.8 to v5.2.6. A method list required for a virtual machine interface module (VIM) of the HIS may vary depending on available hypervisor APIs and their functionalities. As an example, Table 1 lists several methods excerpted from the HIS implemented in the current version of *pyvmpop*.

As mentioned above, the OAS is closely connected with the previous steps for defining user actions and building scenarios before the actual population step. That is, the OAS should provide various methods to automate user actions. For example, an operating system automation module (OAM) for Windows may include automation methods as listed in Table 2, which are implemented by using system commands, shell scripts and keyboard/mouse events usable in Windows. The list only shows a part of possible actions, so it needs to be continually updated.

The EMS was implemented as a switchable component, so an event monitoring module (EMM) can be optionally used by switching on and off as needed. Although the monitoring agent could be developed manually to support personal needs, the current implementation utilizes an existing application, Process Monitor for Windows. With this application, *pyvmpop* can support the monitoring feature for Windows Vista and later.

The implementation approach for the ULS relating to logging modules may vary depending on execution environments such as existing log management systems, available resources, and especially utilization plans after storing log data. Therefore, as a simple example, logging modules implemented in the current version manage action and event logs as the CSV (Comma Separated Values) format. First, the action logging module (ALM) of the ULS that is to process requests from the OAS stores all action records with timestamps into a single CSV file. Next, the event logging module (ELM) of the ULS produces an independent event log file using the CSV format for each event monitoring result.

The DES for extracting data selectively from image files was developed based on an open-source project, *dfVFS* that provides a generic interface to access various storage formats, volume systems and file systems (dfVFS, 2018). The current implementation of the HIS for VirtualBox can produce VHD images from virtual storages of a VM within the hypervisor, and then the DES will interpret image files to access all file systems.

In summary, as a result of operating all the subsystems of VMPOP, data extracted by the DES will be resultant reference data together with action and event logs saved by the ULS. As such, the current version is being developed as a scalable form, so it can be updated for other operating systems in the future. The source code of *pyvmpop* and related examples are available online (pyvmpop, 2018).

## Deployment: cfreds-2017-winreg

This section introduces a practical example of deploying TREDE and VMPOP to generate a reference Windows registry dataset, `cfreds-2017-winreg`. Note that the following sub-sections highlight primary parts of the overall work, which will be available from NIST/CFReDS project website (NIST, 2018).

**Table 3**
Considerations and TREDE classes of Windows registry dataset.

| Research and tool testing considerations | UGRD | SGRD |
|---|---|---|
| Supporting various input types[†] | ✓ | ✓ |
| Parsing normal registry hive files | ✓ | ✓ |
| Parsing corrupted registry hive files | ✓ | N/A |
| Recovering deleted registry data | ✓ | ✓ |
| Interpreting well-known registry data (artifacts) | N/A | ✓ |
| Countering anti-forensics | ✓ | N/A |

[†] Input data may be one of the follows: hive file(s), hive set(s) and disk image file(s). A hive set generally consists of SAM, SYSTEM, SOFTWARE, SECURITY and pairs of [NTUSER, USRCLASS] for each Windows account, and multiple hive sets can be found from Restore Points (XP–) as well as Volume Shadow Copies (Vista+). Here, two symbols are used for signifying earlier (−) and later (+) versions.

### *Purposes and targets*

The purpose of this example is to develop reference data for research, development, education and training activities on digital forensic techniques relating to Windows registry, and furthermore to establish ground truth data for tool testing work.

The target of this corpus generation includes the registry hive file format as well as known registry hive files that are being used in modern Windows systems. Thus, to determine which classes of TREDE are required, this initial step identifies common research and tool testing considerations on Windows registry. The Table 3 shows the considerations and related TREDE classes determined for this dataset. In summary, the final dataset will consist of both UGRD (experimentally created hive files) and SGRD (hives from Windows OSes populated by VMPOP).

### *User-generated Windows registry data*

#### *Requirements*

The TREDE steps to develop user-generated registry data should consider the following three fundamental limitations on the Windows registry hive file format (Microsoft - Windows registry, 2018; Metz, 2018; Norris, 2009; Suhanov, 2018):

- A key name has a limit of 255 characters.
- A value name has a limit of 16,383 characters.
- A registry tree can be 512 levels deep.

Based on these limitations, it is also necessary to consider UGRD-related items as marked (✓) in Table 3. In addition, the hive file format v1.3 and v1.5 will be targeted in the next steps.

#### *Environments*

Reference registry data can be generated from the built-in registry-related engine of Windows OS. In this deployment, all generation methods will be developed and tested using a workstation having Windows 7 Enterprise SP1 (64-bits). So, this work utilizes Batch and PowerShell scripting languages to develop assistance tools since they are built right into Windows. Also, because there exists a default registry management program (known as RegEdit.exe) in Windows, it is possible to conveniently create registration entry (.REG) files to add, modify, or delete registry keys and values.

In addition to these default features, this work sets up a programming environment based on Python v2.7 and v3.4, meaning that it is possible to automate complicated actions required for creating and manipulating data through writing Python scripts. Furthermore, as an alternative method for handling the registry hive format, this work also prepares a Linux environment having an open-source *hivex* library to understand differences between Windows APIs and other implementations (Libguestfs, 2018).

#### *Categorization*

This step defines four primary categories of user-generated registry data as listed in Table 4. The first 'NR' category consists of normal registry hive files, which mean general and benign data based on the hive file format. In this category, there are different classes associated with registry items such as key, value and data. More specifically, this work considers all supported data types (e.g., STRING, BINARY and DWORD) and even big data (>16,344 bytes) for creating all possible data objects. Very long key/value names and

**Table 4**
Categories for user-generated registry data.

| Code | Class | Examples of sub-class |
|---|---|---|
| NR | Data | All supported data types (total 12 types) |
| | | Big data (>16,344 bytes) |
| | Key | Tree structure for multiple keys |
| | | Tree structure with the maximum levels |
| | | Maximum key name length |
| | Value | Maximum value name length |
| | String | ASCII and Non-ASCII characters |
| NRD | Key deletion | Delete a key with values and subkeys |
| | | Delete a key without values and subkeys |
| | Value deletion | Delete a value with normal/big data |
| | | Delete multiple values in a key |
| | Data change | Change data and remain original size |
| | | Change data to smaller/larger size |
| | Key name change | Change key name and remain original size |
| | | Change key name to smaller/larger size |
| | Value name change | Change value name and remain original size |
| | | Change value name to smaller/larger size |
| CR | Valid header | Four sub-classes as shown in Fig. 5 |
| | Missing header | Three sub-classes as shown in Fig. 5 |
| MR | Data hiding | Hide keys, values and data |
| | Infinite loop | key loop |
| | Invalid data size | Integer/Binary/String data too large |
| | Version mismatch | Big data management depending on the hive format version indicator |
| | Ambiguous encoding | Key/Value name's encoding flag mismatch |
| | | String data with different encodings. |

simple/complicated tree structures are also included based on the requirement. In addition, the string class is used to handle string objects (key/value name and string data) containing ASCII or non-ASCII characters.

The next 'NRD' category includes normal registry hive files with deleted registry data. That is, a hive file of this category will have unallocated areas as a result of deleting registry items. Here, this work considers 'deletion' and 'change' operations as user activities relating to the deletion of registry data. Based on these operations, various sub-classes are derived with different sub-conditions. Users may not only be able to delete a key with or without sub-items (values or subkeys), but also to delete a value with normal or big data. Users can also change data streams and names with various size conditions.

Each hive file included in the 'CR' category will have one or more corrupted blocks. The corruption on a registry hive file can occur in various situations. For example, when Windows has abnormally shut down, when hive files are carved incompletely from unallocated and unused areas of a file system or binary dump, when there are partial registry data in the physical memory related areas, when a storage media is damaged, etc. With these points of view, Fig. 5 shows the detailed classes of this category. Here, two primary conditions, valid and missing hive header (base block), are used as the key factor of corrupting registry hive files. The first four types of the figure are to corrupt one or more hive blocks except the hive header, and the other types try to corrupt hive blocks including the hive header.

The last 'MR' category includes manipulated registry hives, which are related to plausible anti-forensic activities in order to confuse digital forensic tools and registry analysis works. There are many methods because it is similar to find unpredictable vulnerabilities from unknown executables, so this work defines several feasible activities for proposing simple automated methods rather than developing sophisticated fuzzing algorithms to find all possible manipulation (or attack) points. As listed in Table 4, this category consists of five primary classes; data hiding, infinite loop, invalid data size, version mismatch, and ambiguous encoding. Although it is very difficult or sometimes not possible to perfectly identify manipulated registry data, they can still be identified in many imperfect cases. This is done through the verification process on internal structures due to a special characteristic of the hive format that multiple structures manage similar or same values (mainly size and count) in different positions. That is to say, if users can get any alert messages from registry-related tools when abnormal values or structures are detected, it would be possible to lead them to perform a more detailed analysis.

*Development of generation methods*

This step designs and implements actual generation methods for each UGRD class listed in Table 4. Here, the following five types of assistance tools are considered for enabling fully or semi-automated operations (T1) Registration entry (.REG) file, (T2) PowerShell script,

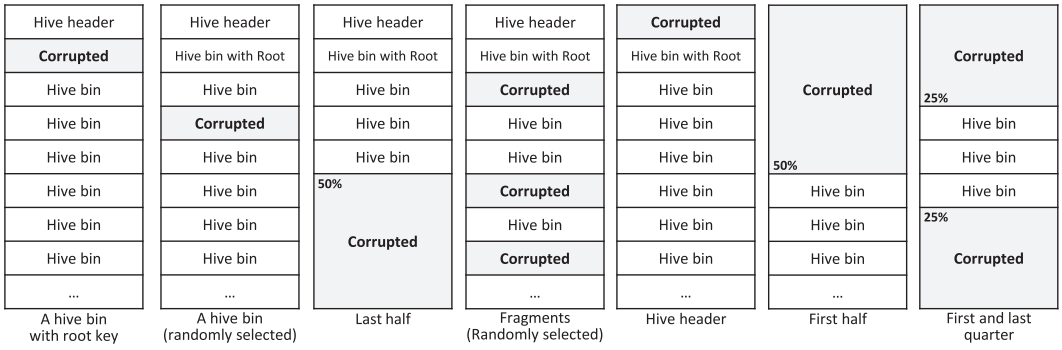


**Fig. 5.** Detailed corruption types of the CR category.

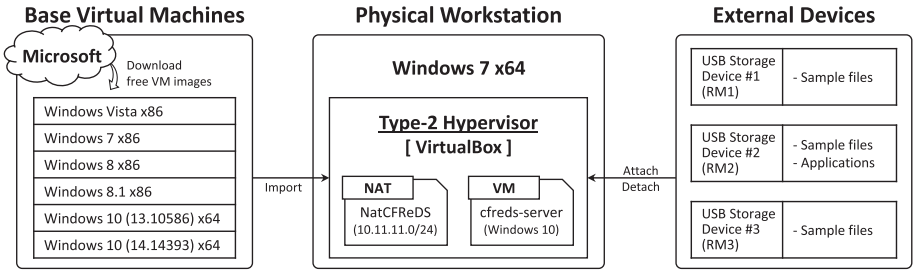


**Fig. 6.** Environments for developing system-generated registry data.

(T3) Batch script (for assisting manual operations with RegEdit.exe), (T4) Python script in Windows, and (T5) Python script using *hivex* in Linux. The first three types (T1~T3) based on Windows registry APIs are applied for 'NR' and 'NRD' classes, and (T5) is used for the two categories in order to create experimental data which Windows does not support, as well as to understand differences between Windows APIs and another implementation. In addition, (T4) is utilized only for the generation of 'CR' and 'MR' classes.

In this step, it was possible to write.REG files for representing most of the generation methods relating to 'NR' and 'NRD' classes. In the case of several 'NRD' classes, it was necessary to develop PowerShell scripts and perform manual operations along with Windows registry editor (RegEdit.exe) for renaming registry keys and values, because the renaming feature is not supported by.REG engine. As a result, 74 assistance tools were developed including 17.REG files, 6 PowerShell scripts, 1 Batch script, 33 Python scripts for Windows, and 17 Python scripts for Linux.

*Execution of assistance tools and manual operations*

The final step for UGRD is to execute developed assistance tools and manual operations to obtain user-generated registry data. For the efficient generation, this step first developed integrated execution support tools (Windows Batch and Linux Bash scripts) to conveniently execute all the assistance tools, except for a tool for assisting manual operations. Section Summary of Resultant Dataset will summarize the resultant dataset generated by the above UGRD steps.

For more information, the details on assistance tools, manual operations and integrated execution support tools prepared by this work can be found online (Park, 2018) (NIST, 2018).

*System-generated Windows registry data*

*Requirements*

The TREDE steps to develop system-generated registry data should be aware of SGRD-related considerations as marked (✓) in Table 3 and the following requirements as well:

- The reference systems should consist of various versions of Windows, and they should not contain any valid license information for commercial software.
- A common scenario should be developed considering a variety of forensically meaningful user actions in order to populate reference systems, and the common scenario should be applied to all the reference systems to identify differences between various versions of Windows. Note that some actions may be valid only in specific versions.
- The reference data should include various well-known registry hive files from the reference Windows systems. It should also include those files from a system partition and its backups (volume shadow copies) as well.

*Environments*

Fig. 6 illustrates execution environments established for the overall generation steps, which include a physical workstation, a type-2 hypervisor, three USB storage devices and base virtual machine images.

First, a physical workstation with Windows 7 (v6.1.7601) is prepared for executing VirtualBox hypervisor and running *pyvmpop* developed for the automated virtual machine population. As default parts relating to the hypervisor, a NAT network is configured to enable communications between running virtual machines, and a common virtual machine (*cfreds-server*) as a server system is created to enable a network drive and remote desktop features. Next, several virtual machine images from Microsoft are used as sources of base virtual machines (Microsoft - Virtual machines, 2018). As shown in Fig. 6, this work prepares six different versions from Windows Vista to 10, and then imports all the images into VirtualBox before proceeding to the next steps. Finally, to support user actions relating to external devices, three USB thumb drives are prepared with sample files and applications without license information.

*User action list*

This step defines possible user actions associated with Windows registry to develop a reference scenario at the next step. As listed in Table 5, a variety of actions are used for representing user behaviors

**Table 5**
User actions related to Windows registry.

| Action class | Example actions |
|---|---|
| Common | Start/Shutdown/Restart |
| | Close/Maximize a window |
| | Create/Close a virtual desktop [Win 10+] |
| Input Device Configuration | Send keyboard/mouse events |
| | Set date, time and time zone |
| | Disable/Enable/Configure NICs |
| | Configure audit policy and Eventlog |
| | Disable/Enable UAC |
| | Enable/Configure File History [Win 8+] |
| Account | Add/Delete/Change a local account |
| | Add/Delete/Change an email account [Win 8+] |
| | Logon/Logoff |
| Process | Launch a program |
| | Terminate a process |
| | Control a web browser |
| Application | Install/Uninstall a program |
| File system | Open a shell (Windows Explorer) |
| | Change directory (Windows Explorer) |
| | Copy files and directories |
| Portable Device | Attach/Detach a USB device |
| Search | Search a keyword (Windows Search) |
| Share | Share a directory |
| Network Drive | Connect to a network drive |
| | Map/Unmap a network drive as a local drive |
| Remote Desktop | Connect to a remote desktop |
| System Backup | Create a restore point (Volume shadow copy) |

that include not only basic classes such as restarting a system and sending keyboard events, but also special classes for Windows such as enabling File History and creating a volume shadow copy. It is important to note that this work primarily considers typical features of Windows rather than user applications, so the action list needs to be updated if corpus creators need to consider other user actions.

Here, for enabling fully automated operations with VMPOP, all the required actions should be supported by the HIS and OAS (Tables 1 and 2, respectively) of the framework, so related modules of the subsystems need to be updated if needed.

*Reference scenario*

With actions defined earlier, this step develops a common scenario depicting user behaviors to create forensically meaningful artifacts. As a result, eight action stages (except for a pre-requirement stage) were established for grouping associated actions into several categories, including OS configuration, account, external device, application, special feature, and anti-forensics. Each action stage is composed of multiple user actions with detailed descriptions, and consequently the developed scenario for this work contains a series of 243 actions as briefly listed in Table 6. It should be note that several user actions such as '*Launch a program and then terminate the process*' and '*Search keyword*' will be executed multiple times with different parameters, so the total number of actions to be performed can be more than 243 according to versions of Windows. For example, 467 actions were performed while the population process for Windows 10.

*Target system population and data extraction*

This step implements executable codes based on the *pyvmpop* for the developed scenario. Hereinafter, a set of these executable codes is referred to as a VMPOP scenario.

More specifically, Appendix A shows simplified Python codes excerpted from the implemented VMPOP scenario for this work. As shown in the `__init__()` method, this scenario has a pre-

**Table 6**
Brief excerpts from a scenario for system-generated registry data.

| Action | Windows version | Description |
|---|---|---|
| [ACTION STAGE 0] Pre-Requirement (13 actions) | | |
| Add a local account | * | Account (CFTT), PW (cftt@nist) |
| Logoff and then Logon | * | Account (CFTT) |
| [ACTION STAGE 1] OS Configuration (10 actions) | | |
| Change time zone | * | From UTC-8 to UTC-05 (Eastern Time) |
| Restart and then Logon | * | Account (CFTT) |
| Enable File History | 8+ | Turn on File History with a local drive |
| [ACTION STAGE 2] Account (18 actions) | | |
| Add a local account | * | Account (Forensics), PW (forensics@nist) |
| Logoff and then Logon | * | Account (Forensics) |
| Add an email account | 8+ | Account (cftt.user1@outlook.com) |
| [ACTION STAGE 3] External Device (13 actions) | | |
| Attach a USB device | * | USB storage drive (RM1) |
| Open a shell | * | Open the default shell (Windows Explorer) |
| Copy files | * | Copy a directory ([RM1+Samples]) to Desktop |
| [ACTION STAGE 4] Application Part I (91 actions) | | |
| Install a program | 7+ | Install a program (Google Chrome)<br>- [RM2]\#1_web-browser\ChromeStandaloneSetup.exe |
| Launch a program | 7+ | Launch a web browser (Chrome) |
| Control a web browser | 7+ | Visit a web site (www.cftt.nist.gov) |
| Terminate a process | 7+ | Terminate a web browser (Chrome) |
| Logoff and then Logon | 8.1+ | Account (cftt.user1@outlook.com) |
| Install a Store app | 8.1+ | Install a Windows Store app (ZIP Opener) |
| Launch a Store app | 8.1+ | Launch a Windows Store app (ZIP Opener) |
| Logoff and then Logon | 8.1+ | Account (CFTT) |
| [ACTION STAGE 5] Application Part II (47 actions) | | |
| Launch a program and then terminate the process | * | Launch a program (VLC media player)<br>Repeat for 10 sample files including:<br>- [Desktop]\RM2+Samples\dir-1\dir-1-1\audio1.mp3 |
| Launch a program and then terminate the process | * | Launch a program (Adobe Reader)<br>Repeat for 2 sample files including:<br>- [Desktop]\RM2+Samples\dir-1\dir-1-2\document1.pdf |
| [ACTION STAGE 6] Special Feature Part I (12 actions) | | |
| Search keyword | * | Search keywords using 'Windows Search'<br>- hello (English), ¡Hola! (Spanish), 안녕하세요 (Korean), etc. |
| Shared a directory | * | Share a directory (C:\\welcome) |
| Create a virtual desktop | 10+ | Create a new virtual desktop |
| [ACTION STAGE 7] Special Feature Part II (14 actions) | | |
| Map a network drive | * | Map a network drive as a local drive<br>- Drive Letter (W), URL (\\10.11.11.127\NETWORK_DIR) |
| Connect to a remote desktop | * | Connect to a desktop using 'mstsc.exe'<br>- URL (10.11.11.127), ID (cfreds-server1), PW (cs1nist) |
| [ACTION STAGE 8] Anti-Forensics (25 actions) | | |
| Uninstall a Store app | 8.1+ | Uninstall a Windows Store app (Dropbox) |
| Set date and time | * | Set the date and time: +24h |
| Delete an account | * | Delete an existing account (Temporary) |
| Uninstall a program | * | Uninstall a program (qBittorrent) |
| Create a restore point | * | Volume Shadow Copy |
| Launch a program | * | Launch a program (CCleaner) |
| Shutdown | * | Shutdown the current system |

assigned target OS list that includes VMs registered at VirtualBox. The `start()` method is the entry point function of the class, so there is a for-loop for repeating the population and extraction process with all the target machines. To populate each VM, a newly created instance of 'VmPop' class is configured by using the `basic_config()` method of the instance, and then the `scenario()` method tries to call the `action_stage_#()` methods after connecting to the target machine through the current VmPop instance. In addition, during all these processes, displaying each VM is recorded as a video file using the `start_video_capture()` method provided by HIS. As an example of implemented action stages, the `action_stage_1()` method shows sample codes for various user actions including changing the time zone, configuring the IP address, restarting the system and logging on to the system. After finishing all population processes within the `scenario()` method, the next few lines are in charge of exporting virtual storages of the target machine to VHD files by using the `get_disk_list()` and `export_disk()` methods from HIS, and then extracting interesting data (hive files in this case) from the exported image files by calling the `open_image()` and `extract()` methods from DES. Finally, the for-loop repeats until all of the target systems are processed.

*Summary of resultant dataset*

As a result of all TREDE steps above, 144 hive files (about 341 MB) as UGRD were created as listed in Table 7, which shows the count, data size and types of assistance tools used for each UGRD category (NR, NRD, CR and MR). Note that the 'Count' column indicates the number of hive files only, excluding transaction log files associated with them. In addition, six different sets of SGRD (645 hive files with a total size of approx. 5.51 GB) were extracted from six different versions of Windows. Table 8 shows a summary of data produced by *pyvmpop*, which includes action/event log files, screen capture videos, and registry hive files extracted from a system partition and volume shadow copies (VSC) of each populated virtual machine. The Windows registry dataset and associated resources developed in this work will be available at NIST/CFReDS project website (NIST, 2018).

## Evaluation

This section mentions the efficiency and potential value of this work's proposal applied for generating a reference Windows registry dataset as an example. Note that this work considers some distinct attributes for sustaining the usefulness of the proposal, since it is hard to objectively evaluate a newly proposed methodology without directly comparable studies.

**Table 7**
Summary of user-generated Windows registry data.

| Code | Class | Count | Size | Types of assistance tools |
|---|---|---|---|---|
| NR | Data | 6 | 5.5 MB | [Windows] RegEdit (.REG) |
| | Key | 11 | 102 MB | [Linux] Python with hivex |
| | Value | 4 | 1.0 MB | |
| | String | 3 | 0.8 MB | |
| | All in one | 3 | 5.1 MB | |
| NRD | Key deletion | 9 | 15.4 MB | [Windows] RegEdit (.REG) |
| | Value deletion | 6 | 11.2 MB | [Windows] RegEdit (manual) |
| | Data change | 12 | 21.6 MB | [Windows] PowerShell script |
| | Key name change | 12 | 32.2 MB | [Linux] Python with hivex |
| | Value name change | 12 | 22.5 MB | |
| CR | Valid header | 8 | 15.0 MB | [Windows] Python script |
| | Missing header | 6 | 11.2 MB | |
| MR | Data hiding | 36 | 67.5 MB | [Windows] Python script |
| | Infinite loop | 2 | 3.8 MB | |
| | Invalid data size | 6 | 11.2 MB | |
| | Version mismatch | 2 | 3.8 MB | |
| | Ambiguous encoding | 6 | 11.2 MB | |

*Usefulness of assistance tools and log files*

In all the TREDE steps, various assistance tools were developed to support the fully or semi-automated generation. Like the results from other general research and development activities, developing automated tools has valuable benefits in terms of enhancing verification and customization. That is, these tools reinforce repeatability and reproducibility of a corpus. As introduced in Section TREDE: Two-class Reference Data Development, TREDE induces corpus creators to make data generation methods shareable through developing various assistance tools for both UGRD and SGRD. Especially, in the case of SGRD, because the target systems may include operating systems or user applications, more sophisticated tools are required for handling all associated resources efficiently. For supporting this requirement, this work developed and utilized VMPOP to prove feasibility of the proposed concepts. The results of an example deployment on Windows registry show that this practical framework provides a high level of convenience by automation, and it also provides a high usefulness to populate target systems as well as extract desired data.

In addition to these assistance tools, a variety of log files that describe what happened were also created during the TREDE steps especially for SGRD. As mentioned above, VMPOP produces various log files automatically such as a progress log that contains *pyvmpop*'s logging messages for debugging purposes, an action log that manages timestamped information relating to user actions, event logs that store event records monitored within target virtual machines, and screen capture files that contain all operations of virtual machines. More specifically, the action log and screen capture files will be helpful for verification of user actions performed during the automated population process, because they consist of a detailed timeline and screen captures having actual flows of operations. Regarding the event log files created by the monitoring agent within virtual machines, they are also valuable for identifying various artifacts created, modified or deleted by user behaviors, similar to the concept being used by existing automated malware analysis systems. As such, these log files for verification and validation of resultant datasets can be used as ground truth data for multiple purposes.

*Effectiveness of automatic approaches*

The proposed concept for the use of integrated automated approaches has advantages in terms of the accuracy, consistency and efficiency. These automated approaches using assistance tools can dramatically reduce (or eliminate) the possibility of unintended results by human mistakes. In other words, corpus creators can produce accurate results according to a planned strategy whenever they want. Therefore, these kinds of data generation approach also have consistency, because performing a fully automated action will produce same results when it is performed again if there is no exception.

To prove feasibility and efficiency of the proposed methodology, this work introduced a practical example including multiple categories for UGRD and a relatively complicated scenario for SGRD as briefly explained in Section Deployment: cfreds-2017-winreg. Especially, the scenario for SGRD was built using a variety of user actions, and then it was implemented as a Python class based on *pyvmpop*. For providing more information about the target system population and data extraction, Table 9 shows the time taken by the implemented VMPOP scenario to obtain system-generated Windows registry data. The total time took about 2–4 h according to the versions of Windows. Except for logging and waiting

**Table 8**
Summary of system-generated Windows registry data.

| Windows | Data type (count) | Count | Size |
|---|---|---|---|
| Vista | Action log (1), Event log (347) | 348 | 637 MB |
| | Screen capture video (7) | 7 | 452 MB |
| | Hive from a system partition (24) and 4 VSCs (96) | 120 | 1.25 GB |
| 7 | Action log (1), Event log (409) | 410 | 468 MB |
| | Screen capture video (7) | 7 | 577 MB |
| | Hive from a system partition (19) and 4 VSCs (72) | 91 | 675 MB |
| 8 | Action log (1), Event log (416) | 417 | 561 MB |
| | Screen capture video (7) | 7 | 706 MB |
| | Hive from a system partition (29) and 3 VSCs (84) | 113 | 829 MB |
| 8.1 | Action log (1), Event log (433) | 434 | 605 MB |
| | Screen capture video (7) | 7 | 785 MB |
| | Hive from a system partition (29) and 3 VSCs (84) | 113 | 1.00 GB |
| 10 | Action log (1), Event log (467) | 468 | 763 MB |
| | Screen capture video (7) | 7 | 886 MB |
| | Hive from a system partition (26) and 3 VSCs (78) | 104 | 922 MB |
| 10RS1 | Action log (1), Event log (467) | 468 | 806 MB |
| | Screen capture video (7) | 7 | 803 MB |
| | Hive from a system partition (26) and 3 VSCs (78) | 104 | 902 MB |

**Table 9**
Time taken[†] (HH:MM:SS) by VMPOP to obtain a system-generated Windows registry dataset.

| Windows | Total | EMS | ULS (ALM only) | Waiting for idle |
|---|---|---|---|---|
| Vista | 02:11:13 (100%) | 00:50:52 (38.8%) | 00:14:22 (10.9%) | 00:03:10 (2.4%) |
| 7 | 02:43:53 (100%) | 01:01:48 (37.7%) | 00:16:55 (10.3%) | 00:02:49 (1.7%) |
| 8 | 03:22:02 (100%) | 01:07:12 (33.3%) | 00:18:43 (9.3%) | 00:07:48 (3.9%) |
| 8.1 | 03:42:25 (100%) | 01:15:10 (33.8%) | 00:19:48 (8.9%) | 00:10:01 (4.5%) |
| 10 | 04:12:25 (100%) | 01:24:08 (33.3%) | 00:21:47 (8.6%) | 00:18:26 (7.3%) |
| 10RS1 | 03:47:31 (100%) | 01:12:06 (31.7%) | 00:21:50 (9.6%) | 00:13:52 (6.1%) |

[†] The population was performed on a desktop system with Intel Core i5-4460, 8 GB RAM and a solid-state drive.

operations, it took about 1–2 h for performing all actions automatically. This means that the proposed automatic population has effectiveness, in terms of carrying out boring and repetitive tasks with precision without mistakes, obtaining ground truth data and enabling concurrent executions for multiple target systems. If a corpus creator manually performs the population process, it might take approx. 10 h even without logging operations.

As listed in Table 9, a large part of the total time was related to VMPOP's logging features: EMS (Event Monitoring Subsystem) and ULS (Universal Logging Subsystem). Firstly, the average time taken by the EMS was around 34.8% of the total time. The primary reason for this relatively high rate was that an event monitoring request took about 9.7 s, in order to execute an event monitoring agent shortly before an action and terminate the agent just after the completion of the action. In the current implementation of *pyvmpop*, a certain waiting time had to be added to ensure proper operations for loading/unloading kernel modules of the event monitoring agent (Process Monitor for Windows). Consequently, the currently implemented EMS for Windows was not very effective, but it could be improved by future works such as finding more effective monitoring policies or developing customized monitoring agents. Secondly, the average time taken by the ALM (Action Logging Module) was about 9.6% of the total time. As mentioned in Section ULS: Universal Logging Subsystem, a request for the action logging can occur before, during or shortly after an action, and for example, the action logging function was called 2631 times for the population of Windows 10 system. Overall, creating an action log entry took an average of 0.497 s, for internal operations such as calling several hypervisor APIs and executing a tiny program to get the current date/time and timezone information from a running VM. Meanwhile, the ELM (Event Logging Module) of the ULS had little impact on the performance, because the current implementation of the ELM for Windows simply supports copying and renaming CSV files created by an event monitoring agent.

Note that the last column of the table shows the time taken for waiting for the completion of specific actions including '*Logon*' and '*Install a Store app*'. Section Automation Challenges will discuss about this waiting issue in detail.

### *Applicability*

The TREDE steps induce generating multi-purpose datasets, and thus datasets generated by this work's proposal have wide applicability to a variety of purposes, especially since they consist of both UGRD and SGRD through systematic steps as well as VMPOP.

In general, the experimentally created UGRD, in compliance with the specification of target digital data, will be useful for education, practice, training and research for the understanding of internal formats and detailed storing mechanisms of associated software. Because the UGRD can be composed of not only normal but also infeasible data beyond the limitations of a pre-defined specification, it would be also valuable for algorithm validation, detection of potential vulnerabilities, equipment checking, tool development and testing, especially verifying fault tolerance on exceptions.

Next, SGRD that is generated naturally by target operating systems or applications can also support various purposes, since it is developed based on realistic scenarios mimicking actual human. Especially, the concept of VMPOP provides advantages in terms of applicability by making user actions automatable as well as programmable. It means that corpus creators can develop each scenario as executable codes with the proposed framework, and they can simply execute the codes to create a synthetic dataset whenever the need arises. Thus, this approach can be used for generating not only entirely different datasets through continuous endeavors of scenario development and implementation, but also similar (non-identical) datasets by simply adding random factors to a scenario having a shared pool of actions and associated resources.

Such conveniences of development provided by VMPOP might help corpus creators generate datasets for everywhere where it is necessary to reflect the latest updates as well as technology trends, including but not limited to research, development, education and tool testing.

Through the above-mentioned reasons, the proposed strategy would be applied and utilized for diverse purposes in the field of security and digital forensics. Furthermore, as an example of those purposes, the NIST/CFTT is planning to design a tool testing work using the registry dataset developed by this work.

## Discussion

### *Automation challenges*

Automation definitely has a lot of benefits, but there are several challenges that need to be overcome. In this context, this work developed *pyvmpop* as a proof-of-concept that is designed for automatically producing SGRD, and then successfully tested the implementation through an example pertaining to Windows registry. Nevertheless, there were several challenges that should be considered to make the mission possible.

The first automation challenge is to handle *waiting issue* (or *delay issue*), meaning that a VMPOP scenario should wait until the current action is completed before proceeding the next action. Hence, clear methods had to be prepared for checking the latest status of a target operating system or application. A simple method for this may be just adding a time delay of a few seconds or minutes according to operations of each action, and actually this work used the method for ensuring each action's minimum execution time. However, it is not enough because it may not only lead to a waste of time if operations of an action are finished before a pre-defined delay, but also fail to wait if those operations take more time than expected due to any issues such as the read/write speed of storage devices and the status of running processes within a host operating system. So, it was necessary to use additional methods such as waiting until the system becomes idle (i.e., the current CPU usage gets below a threshold for a few seconds) and checking the text of a specified window's title bar. Note here that the above-mentioned solutions are just related to specific operating systems or user applications, so corpus creators may need to research and develop proper methods for each target of automation.

The next challenge is to deal with *continuous software updates*, meaning that it is necessary to disable automatic updates as well as to handle a sudden pop-up window for the update notification. Since these updates can happen from both operating systems and applications, they should be identified prior to implementing a VMPOP scenario in order to automate all actions smoothly.

In addition, the other challenge can occur because of *unexpected errors* from operating systems or user applications. Actually, during the deployment explained in Section Deployment: cfreds-2017-winreg, the execution of *pyvmpop* had to be repeated several times, due to errors occurred by the hypervisor, operating systems and user applications.

### *Interactions between users and technologies*

To enhance the realism of synthetic scenarios, it is essential to understand many types of user interactions with various sources of potential digital evidence. Since traces created by a sequence of actions are just limited to the actions pre-defined by corpus creators, it is required to define meaningful actions as detail as possible based on practical knowledge about digital forensics and security.

As an example of defining those actions, this work defined several user actions relating to Windows registry as listed in Table 5. They of course are part of all possible actions, so the action list needs to be updated in accordance with the purpose of corpus generation. For doing that properly, it is necessary to perform continuous research on a variety of interactions between users and sources of potential digital evidence such as hardware, software, network, cloud services, etc.

### *Distribution and sharing*

As mentioned in Section Research Objectives, a common concern relating to data corpora is how to distribute datasets to the public or within a closed group.

In the case of distributing or sharing whole disk images having operating systems, several methods proposed in existing studies may be useful for reducing the compressed size of data through redacting or scrubbing unrelated data. A distribution concept proposed by Scanlon et al. may also be a possible solution because it enables efficient storage and distribution of differences between a base (previously distributed) image and a resultant image populated by corpus creators (Scanlon et al., 2017).

Although these methods to reduce the data size are valuable and applicable to many use cases, there is still demand for a more general and efficient way. That is why this work proposed detailed steps on how to create synthetic data. More specifically, it will be effective if the forensic science community can build an ecosystem, where stakeholders are sharing fine-grained steps, scenarios and assistance tools developed for corpus generation activities.

## Conclusion and future direction

This paper introduced a procedural methodology, TREDE (UGRD + SGRD), to improve the efficiency of forensic corpora development, and developed a new autonomous framework, VMPOP, to assist data generation processes especially for SGRD by utilizing a virtualized environment. This work also demonstrated their effectiveness through a Windows registry dataset as an example.

To build a systematic and productive environment where can cultivate a sustainable development of corpora, it is important to not only document the entire development process in detail, but also share implementation results for automated operations. This work's proposal as an endeavor to step forward will create shareable resources that are easy to modify and deploy, so it will help construct an ecosystem for sharing of ideas, methods and tools to develop data corpora.

There are several plans for enhancing VMPOP via various updates such as adding modules to automate various operating systems except for Windows, visualizing logging results for the effective analysis and reinforcing automation features for a variety of user applications through understanding meaningful interactions between users and technologies. The developed Windows registry dataset will also be updated to include hive files and their transaction log files from the latest versions of Windows 10. In addition, the propagation of newly emerging technologies like IoT (Internet of Things) and cloud-based services may require other forms of dataset development in order to support research, education, practice and tool testing reflecting pervasive trends. Thus, future works will establish fundamental concepts and practical approaches for that kind of new targets.

## Acknowledgments

The author thanks Barbara Guttman, Jim Lyle and Susan Ballou at NIST for their valuable comments and suggestions.

## Appendix A

```
class VmPopScenarioCFReDS2017WinReg:
    def __init__(self):
        self.os_list.append(("Win10RS1_14393_IE11+Edge_(CFReDS)", VmPopOSType.Windows10_64))
        self.os_list.append(("Win_7_IE09_(CFReDS)", VmPopOSType.Windows7))
        … skip …

    def start(self):
        for vm_name, os_type in self.os_list:
            vmpop = VmPop()
            vmpop.basic_config(hv_type=self.hv_type, os_type=os_type, shared_dir=self.shared_dir,
                               start_mode=self.hv_start_mode, log_dir=log_dir)

            # Populate a VM with user actions
            self.scenario(vmpop, vm_name, self.default_id, self.default_pw)

            # Export virtual storages to VHD image files
            for d in vmpop.hypervisor.get_disk_list():
                output_path = log_dir + PtUtils.gen_disk_file_name(d)
                vmpop.hypervisor.export_disk(d.get('id'), output_path, VmPopImageFormat.VHD)
                images.append(output_path)

            # Extract forensically interesting data (Windows registry) from image files
            for image in images:
                vmpop.extractor.open_image(image)
                vmpop.extractor.extract(data_class=[VmPopDataClass.WINDOWS_REGISTRY])

    def scenario(self, vmpop, vm_name, user_id, password):
        vmpop.connect_to_vm(vm_name=vm_name, user_id=user_id, password=password)
        vmpop.hypervisor.start_video_capturing("{}.webm".format(vm_name))
        self.action_stage_0(vmpop)
        self.action_stage_1(vmpop)
        … skip …

    def action_stage_1(self, vmpop):
        vmpop.automation.change_timezone("Eastern Standard Time", VmPopActionMethod.WIN_PS)
        vmpop.automation.configure_nic_ip(mode=VmPopNICMode.STATIC, address='10.11.11.77',
                                          mask='255.255.255.0', gateway='10.11.11.1')
        vmpop.automation.restart(mode=VmPopFunctionMode.HV)
        vmpop.automation.logon_account(user_id="CFTT", user_pw="cftt@nist")
        vmpop.automation.configure_eventlog(log_name='Security', max_size='80MB')
        vmpop.automation.enable_file_history()
        … skip …
```

## References


Carrier B. Digital Forensics Tool Testing Images. http://dftt.sourceforge.net (Accessed May 2018).

Digital Corpora. Govdocs1 – (nearly) 1 million freely-redistributable files. http://digitalcorpora.org/corpora/files [Accessed May 2018].

Digital Forensics Virtual File System (dfVFS). https://github.com/log2timeline/dfvfs [Accessed May 2018].

Flandrin, F., 2012. A Platform for Digital Forensics Evaluation of Tools (D-fet). M.S. thesis. School of Computing, Edinburgh Napier Univ, Edinburgh, Scotland.

ForGe. https://github.com/hannuvisti/forge (Accessed May 2018).

ForGeOSI. https://github.com/maxfragg/ForGeOSI (Accessed May 2018).

Garfinkel, S., 2007. Forensic Corpora: A Challenge for Forensic Research, pp. 1–10. http://simson.net/ref/2007/Forensic_Corpora.pdf [Accessed May 2018].

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.

Libguestfs, hivex. http://libguestfs.org (Accessed May 2018).

Metz J. Windows NT Registry File format specification. https://github.com/libyal/libregf/tree/master/documentation (Accessed May 2018).

Microsoft Virtual machines for testing browsers. https://developer.microsoft.com/en-us/microsoft-edge/tools/vms (Accessed May 2018).

Microsoft Windows registry Information for advanced users. https://support.microsoft.com/en-us/kb/256986 (Accessed May 2018).

Moch, C., Freiling, F.C., 2009. The forensic image generator generator (Forensig$^2$). IMF 2009, 78–93.

Moch, C., Freiling, F.C., 2011. Evaluating the forensic image generator generator. ICDF2C 2011, 238–252.

NIST. Computer Forensic Reference Data Sets (CFReDS). https://www.cfreds.nist.gov (Accessed May 2018).

Norris, P., 2009. The Internal Structure of the Windows Registry. Cranfield Univ., UK (M.S. thesis).

Park J. Assistance tools for cfreds-2017-winreg. https://github.com/jungheum/cfreds-2017-winreg (Accessed May 2018).

pyvmpop: A Python implementation of VMPOP (Virtual Machine POPulation) framework. https://github.com/jungheum/pyvmpop (Accessed May 2018).

Russell, G., Macfarlane, R., Ludwiniak, R., 2012. A forensic image description language for generating test images. Cfet, 2012.

Scanlon, M., Du, X., Lillis, D., 2017. EviPlant: An efficient digital forensic challenge creation, manipulation and distribution solution. Digit. Invest. 20, S29–S36.

Suhanov, M., 2018. Windows Registry File Format Specification. https://github.com/msuhanov/regf [Accessed May 2018].

Vidas, T., 2011. MemCorp: An open data corpus for memory analysis. Hicss, 2011.

Woods, K., Lee, C.A., Garfinkel, S., Dittrich, D., Russell, A., Kearton, K., 2011. Creating realistic corpora for security and forensic education. ADFSL 2011, 123–134.

Yannikos, Y., Winter, C., Schneider, M., 2012. Synthetic data creation for forensic tool testing: improving performance of the 3LSPG framework. Ares 2012, 613–619.

Yannikos, Y., Winter, C., 2013. Model-based generation of synthetic disk images for digital forensic tool testing. Ares 2013, 498–505.

Yannikos, Y., Graner, L., Steinebach, M., Winter, C., 2014. Data corpora for digital forensics education and research. Adv. Dig. Forensics 10, 309–325.