The Penguin Operating System Forensic Recorder (POFR): Reliably collecting, storing and processing large amounts of live forensic evidence using data differencing

Georgios Magklaras, Steelcyber Scientific & Norwegian Meteorological Institute
georgios@mail.steelcyber.com

Abstract

The fields of live digital forensics and computer monitoring play an increasing role in securing digital infrastructures. A plethora of tools cover a part of the digital forensics and incident response requirements spectrum. As digital infrastructures evolve, they produce an ever increasing amount of volatile information. Reliably collecting, storing and analyzing volatile forensic digital evidence presents certain challenges. This paper reviews these challenges in order to propose sixteen essential functional requirements of a modern live forensics tool. It then presents POFR, a Linux based Open Source next generation live forensic tool that reliably collects, processes and organizes the evidence in a manner that meets these functional requirements. Success in this process means access to a representation of events that occurred on the monitored computer systems without impacting accuracy or computing performance, as well as dealing effectively with the Big Data nature of live forensics.

1. Introduction

The option of not stressing the importance of the digital forensics and incident response (DFIR) field in a world that produces and consumes an ever increasing amount of data would simply not be a realistic one. From the petabyte driven world of clinical genomics [1] to the insatiable data storage appetite of the Internet of Things (IoT) [2], accounting for cyber attacks to vehicles [3] and considering vulnerabilities that target the very fundamental control blocks of vital industrial infrastructures [4], one starts sketching the multifaceted and data driven scope of modern digital forensics.

Practitioners and researchers combine static and live analysis techniques to accommodate for that wide scope of DFIR analysis requirements [5]. Static analysis involves techniques that halt the system of interest and obtain a validated copy of its storage media for further examination. In contrast, live analysis gathers information from the system of interest without halting it, in a near continuous or discrete data sampling fashion, depending on the sampled metric. Despite the strong and weak points of each approach, it is widely accepted [5][6] that live forensic analysis provides a more complete picture of the events that lead to the acquisition of a particular artifact. As an example, the discovery of a partially erased image in the filesystem of a device by a static forensic tool is important. However, it says nothing or often very little about the way (applications, sequence of events) the picture was stored.

In addition to static and live forensic tools, DFIR practitioners employ computer monitoring techniques. Tools such as computer log aggregators [7][8] or general system state query tools [9] are examples of this trend. Logs constitute valuable information but their role is always complementary to that of forensic tools in an incident response. The same can be said for tools like osquery [9] because although they span an entire infrastructure and able to provide a rather detailed view of a system state, they cannot be easily used to detect notable trends over time, neither they are always designed to correlate events in a forensic manner.

The challenges that the modern arsenal of forensic tools is facing are well documented in literature. Casey [10] highlights the difficulty in establishing facts from incomplete data generated by both static and live forensic tools. He also mentions the difficulty of educating forensic experts. Garfinkel [11] mentions the difficulty of transitioning academic research tools to an end user, due to tool complexity, lack of documentation and long term maintenance of academic research tools. In addition, the same paper [11] mentions the lack of suitable abstractions for representing forensic data, as well as the difficulty of consistently representing a timeline of events in an evidence based manner. Grajeda et al [12] focused on the problems of producing and making available forensic datasets. Wu et al [13] has performed a comprehensive categorization of a representative sample of modern forensics tools. The paper echoes the same concerns to those mentioned in [11] when it comes to tool maintenance, complexity and usability, indicating that the problem of forensic tools is systemic.

Even the most promising live forensics tools [5],[6] suffer from fundamental issues that concern the data acquisition process [14]. One of these issues is the “observer effect” which concerns the ways an investigator can alter the state of the system being investigated and thus impede the accuracy of the acquired data. Other equally important factors concern the integrity of the collected volatile data. For instance, if the data acquisition mechanism was com-

promised, how could an analyst detect signs of such a compromise in a reproducible manner when examining the collected data?

1.1. Fundamental questions on Live Data Forensics

Addressing all of the previously discussed challenges leads inevitably to a process of formulating fundamental questions with the purpose to review the functional requirements of live forensics tools. Asking the right questions is a good starting point in the process of attempting to meet these challenges.

A first category of questions should be concerned with the data acquisition process. Having a representation of events in a time line in mind [11], one should think about the following:

- What information to sample to answer useful questions?
- How often to sample that kind of information, so that we accurately represent a picture of what happened to a system?
- How to ensure that the data is collected in a manner that minimizes the possibility of intentional or accidental corruption?

A second category of questions relates to the storage and processing of the collected data and in particular:

- How can we store the data in a manner that can aid storage management, as well as the ability to search/interrelate the events?
- How to efficiently process large amounts of data?

Both question categories work in tandem and they inevitably influence each other. For example, the amount of information to sample and the frequency of sampling will inevitably challenge the storage management and processing efficiency tasks. The next section presents the argumentation behind concrete answers and will shape a number of live forensics functional requirements.

1.2. Live forensics functional requirements

The key aspect of shaping the answer to the fundamental questions of Section 1.1 of this paper is the preservation of evidence in a manner that can help the examiner answer key questions about an incident. A data forensic specialist would expect from an efficient forensic process to sample enough information to represent a sequence of events. Too little information would make the representation of these events unclear. In contrast, too much information would convey accurately the events at the cost of making the tasks processing/searching difficult or impractical.

However, prior quantifying the amount of information to sample, it is important to examine what is to be sampled. Garfinkel [11] justified the significance of presenting a timeline of events in a evidence based manner.

The importance of reliably maintaining the sequence of events is an essential element of digital forensic evidence [16]. Using the example of a partially erased image that is discovered by static analysis forensic tools, a forensic examiner will have a more relevant and complete case overview if she could prove that the image was associated to a specific program. In addition, the sampled forensic evidence should not only provide proof that the application accessed the discovered image but also provide linkage of other related facts about that access. For instance, did the application download the image by means of a network connection that was established or not? What was the user account that ran the program that accessed that image?

These latter questions can only be answered by live analysis tools. One can then form the first two fundamental qualitative requirements of an evidence oriented live forensics tool:

- R1:A live forensics tool should enable the forensic examiner to relate:

  -File access

  -Process

  -User accounts

  -Network connections

Requirement R1 should relate all four pieces of information in a way that can answer useful questions and exclude non relevant events to the greatest possible extent. For the purposes of clarity, we define the term process as that of an instance of a computer program that is being executed by one or many threads [17].

In addition, as timing information is of essence [16]:

- R2: A live forensic tool should provide a record format that can maintain the sequence of the recorded events.

Having sketched the qualitative aspect of a desirable live forensic data collection scheme, the next issue to be examined is the quantitative one. How much information one needs to collect, in order to portray accurately the evidence in an incidence response?

Adelstein [18] was one of the earliest researchers that identified the challenges of live data forensics, focusing among other things on an important aspect of data collec-

tion, that of data volatility. In the context of live data forensics, data volatility refers to the ephemeral existence of the collected data (processes, file access descriptors, network connections). In direct contrast to static forensics, relevant live forensic data have short lifespans. Thus, knowing the data sampling speed/frequency is of crucial importance for preserving the evidence chain.

Sandvik et al [19] provides a comprehensive quantification of data volatility in live forensics. The study focuses on filesystem operations for resource constrained IoT devices. Nevertheless, it remains relevant because it proves that estimating the useful lifetime of volatile filesystem data is a process that can be complex and specific to the filesystems in question. It also proves that the relevant data lifetimes can vary in quantity from microseconds to years. This adds two important functional requirements to an evidence based live forensics process:

- R3: The ability to complete data sampling loops in an adjustable microsecond scale.
- R4: The ability to provide data recording sessions that could facilitate continuous data sampling periods that last months or years.

Latter paragraphs of this paper will revisit the live forensic data quantitative issue in terms of processing the collected information. However, at this point we shall return to the qualitative matters and address the critical issue of live forensic data reliability. Adelstein [18] mentions a good practice of a forensic examiner, that of not trusting the data collection programs or binaries of the target system due to risk of being maliciously compromised. Law et al [14] discusses the difficulties of verifying the integrity of volatile data, in particular when it comes to target system memory. In addition, an experiment based study by Jones et al [20] concludes that the best place to store live forensics evidence is on another computer on the local network, whereas the worst option would be to save the evidence on a cloud storage device directly from the evidence machine.

Field experience verifies that although most of these concerns are true, it is not always possible to fully address them. On the other hand, it is possible to equip a live forensics tool with functionality that minimizes the likelihood of these undesirable effects.

In response to the compromised target system binaries mentioned by Adelstein [18], modern live data forensics capture does no longer rely on operating system binaries to obtain data. A number of modern tools mine instead core kernel operating system structures directly to obtain results [13], following a modular software stack hierarchy. For the popular Linux operating system for example, a lot of early rootkit examples were maliciously modifying commands such as ‘ps’ and ‘lsof’ to effectively hide malware processes from system administrators, users and security experts [21]. However, a more up-to-date practice is for the utilities to parse directly procfs [22], a virtual filesystem that allows access to process metadata, as well as other kernel and in-memory data structures.

Using procfs in Linux or other core kernel mechanisms of other operating systems is not a panacea against malware. The evolution of rootkits has forensic examiners facing scenarios of malware that targets these core kernel mechanisms [23]. The only available measure to defend against these scenarios is an examination of the target system kernel module space (including device drivers) with static forensic tools, prior commencing live data forensics capture. This is usually part of field forensic examination practices [24]. The usage of Unified Kernel Images and other device driver signing mechanisms should be a recommendation of forensic examiners to guide system users and architects to prevent occurrences of these undesired situations [25]. Thus, a fifth functional requirement of a live forensics tool is added:

- R5: The tool’s data sampling process installation and runtime requirements should not impede or cause overhead with kernel module complexity as part of a Unified Kernel Image or similar driver signing procedures.

An equally important question for the reliability of the collected live forensics data is what happens once all the info is gathered, especially in the context of problems discussed by Jones et al. [20]. The authors of the study conclude that it is best that the collected data do not stay in the target system. The more volatile data remain in the target system, the greater the probability of being modified accidentally (filesystem, memory corruption, power loss) or intentionally. Overhead issues (capacity, system overhead) of storing the collected data in the target system storage media should also be considered. If a live forensics data acquisition process continuously modifies filesystems of the target system, the “observer effect” [5],[6],[14] could modify the storage media and this could have consequences for both live and static forensic data acquisition. This leads to the addition of specific functional requirements for a live data forensics tool:

- R6: The tool must provide ways to reliably transport the live forensic data off the target system.
- R7: The tool must provide functionality to help the examiner tag and possibly exclude from the collected data its own data sampling activities.

How to transport the collected data to a system for processing and analysis is also of crucial importance. Live

forensic data contain confidential information about a system and its general infrastructure. As a result:

- R8: The data transfer process of a live forensics tool should encrypt the data prior transmitting them from the target systems.

In addition to data encryption, the in-transit data need to be protected from accidental or intentional alteration. Adelstein [18] proposed the idea of fingerprinting the collected data by employing cryptographic hash functions [26], in order to help the forensic examiner prove that the evidence is intact. Evidence assurance is a top priority for every forensic process. The only thing that should be added is that the hashing of the collected data should be atomic for every data sampling loop mentioned by Requirement R3. Hashing a block of data that contains info from many sampling loops provides a certain attack space for an anti-forensic tool [27] aiming to alter specific records on a target system. A process that hashes data on every individual sampling cycle would in comparison provide a smaller attack surface and increase data collection reliability. In conclusion:

- R9: The data transmission process of a live forensics tool should hash the collected data on every data sampling cycle prior transmitting them from the target system.

Having justified the need for data encryption and hashing, a modern live forensics tool will have to traverse the complexity of modern network topologies, if it is to send forensic data back to an analysis server. Well defined corporate/institutional data networks allow network administrators to define firewall rules that regulate data transmission between monitored systems and servers. As a result of this, a substantial portion of infrastructure monitoring, management and security software employ an agent based architecture [8][9]. An agent is software installed on a client system for the purposes of regulating the operation and data transmission from clients to servers. Normally, this implies that every client agent needs to open a specific network port for that purpose, in order to facilitate bidirectional information transfer between clients and servers.

Agent based architectures would be challenged in modern networking environments, where mobile, IoT and embedded devices [2],[3] operate on WLAN and cellular telephony telecommunication infrastructures [28]. Mobility roaming would change for example the IP address of client systems. Wireless networks have varying measures of latency and packet loss. Additional challenges arise when combining these facts with other networking mechanisms, such as NAT traversal [29], VPN usage [30] and the large exposure of a target system in IoT/IPv6 based architectures [31].

Under this modern telecommunications landscape, an agent based architecture starts showing distinct disadvantages over an agent-less architecture, where the target system would have no dedicated open ports exposed on the network in the process of transmitting evidence back to a server. Moreover, the process of transmitting volatile data within certain time ranges implies that the tool needs to handle reliably data re-transmissions to the greatest possible extent. Hence, three additional functional requirements are added:

- R10: The data transmission process of a live forensics tool should be able to reliably traverse networks where stateful firewalls, mobile roaming,NAT and VPN technologies are required for the target system to operate.
- R11: The data transmission process of a live forensics tool should provide the minimum possible exposure from a network security point of view.
- R12: For unreliable wireless networks, the data transmission process of a live forensics tool should be able to perform data transfer in the same way databases offer atomicity in transactions. The data will either be transmitted in their entirety unaltered, or they will be discarded.

The previously discussed live forensic tool functional requirements have focused on the target system issues. The discussion of the remaining paragraphs of this section should address requirements that concern the processing of the collected data (Requirements R6 and R7).

Requirements R1 and R2 emphasized the need for evidence to be collected and stored in a way that can enable the forensic examiner to relate processes, file and network endpoint access events with user entities in sequence. Garfinkel [11] mentions explicitly that evidence based design, as part of the “Visibility, Filter and Report” model, has tools use external SQL databases to organize the collected information.

An example of such a tool is discussed by Magklaras et al. [32]. LUARM was not strictly speaking designed as a live forensics tool. It is an audit log engine that addresses IT misuse. However, it provides a number of user action data such as file access, program execution and network endpoint user activities, all organized in easily searchable relational tables [33]. The Structured Query Language (SQL) is a well standardized mechanism [34] that allows an examiner to ask questions like:

- Which programs accessed file A at a particular time and which were the user accounts that were involved in that process?

- Did a particular application accessed a network endpoint and if yes, how many times and in what sequence within a specific time frame?

In addition, relational database management systems (RDBMS) excel in providing transactional consistency, through the properties of atomicity, consistency, isolation and durability (ACID) [35]. These properties are highly desirable for evidence preservation, so any process that processes volatile data should provide some degree of these properties to ensure that the data have been stored in a sound manner.

It should be noted that a substantial portion of forensic tools have moved into NoSQL data management approaches [15]. NoSQL approaches address the data scalability issues that an RDBMS might experience and can be a more efficient mechanism to process large amounts of unstructured data. However, at the time of writing, not all NoSQL systems comply with the previously discussed ACID properties. Thus, a more conservative approach would choose a traditional RDBMS and favor evidence preservation over speed and efficiency. In addition, such a conservative approach should impose a data structure on the volatile data, in order to minimize processing overheads and inefficiencies that concern the processing of unstructured data by RDBMS approaches. NoSQL is also an approach that does not exclude SQL standard queries (Not only SQL) [36]. In conclusion, the following requirements are added:

- R13: A live forensics data tool should store and process the volatile data by means of a mechanism that can issue standard SQL queries.
- R14: A live forensics data tool should impose structure into the collected data.
- R15: A live forensics data tool should store and process the data by means of a mechanism that can provide some or all the properties of atomicity, consistency, isolation and durability (ACID), in order to ensure that volatile data are stored in a consistent manner.

Given the Big Data requirements of modern digital forensics, the previous volatile storage and data processing requirements have not really addressed the core issue of dealing with the sheer volume of data. Finding a way to perform a forensic grade data reduction [37] is essential, especially as the previous paragraphs presented evidence of RDBMS scalability weaknesses. The remaining paragraphs of this section will further qualify what a forensic grade data reduction should entail.

Data reduction is not used here in the strict statistical sense [37]. It's broad context term, that of transforming a large quantity of initial data into a smaller non redundant, ordered data set is the focus of this discussion. There is a wide body of research and development that addresses data reduction in digital forensics and storage solutions. This includes efforts to use various forms of delta compression and similarity based data deduplication [38][39]. There are also notable approaches that are specific to digital forensic imaging, thus pertaining in static forensic analysis techniques. A representative sample of those includes the use of differential analysis in mobile phone imaging [40], selective imaging for data reduction [41] and the use of graph dependencies to compact audit log representation and processing [42]. Lastly, efforts that utilize spatial statistics [43] to achieve evidence data reduction were also considered.

An important consideration of any forensic data reduction technique is to achieve reduced data volume without loosing vital evidence. From the previously mentioned data reduction techniques, those that are based on differential processing [40], selecting imaging [41], and graph based log compaction claim that are lossless, in the sense that the data reduction process does not loose vital evidence. In contrast, statistical techniques such as those in [43] can be lossy. Statistical spatial distribution methods have a margin of error depending on how frequent the data change. They reduce processing time, but they can loose vital evidence. As a result, the last functional requirement is formed:

- R16: A live forensics tool should include data reduction techniques to reduce the volume and the processing time of collected data without loosing/compromising vital evidence.

At that point, we have formulated sixteen essential functional requirements of a live forensics tool, after reviewing relevant research efforts. The next section of this paper presents a tool that attempts to address these requirements.

2. The Penguin OS Forensic Recorder

The Penguin OS Forensic Recorder (POFR) [44] is an Open Source next generation live forensics tool for the Linux operating system. The tool was designed and developed on the basis of the functional requirements of Section 1.3.

POFR is a client server tool (Figure 1), with the target system being the client and the forensic examiner having a server where all the information is gathered, processed and presented. Figure 1 above illustrates its client server architecture.

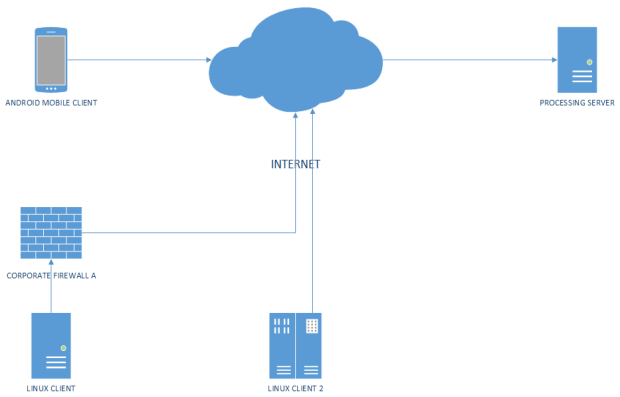

Figure 1: The POFR high-level architecture

One of the important POFR design aspects is that it provides a push based (client → server) information flow, during its basic operation. It does not employ a pull based approach, where a server would need to pull the data from the client. This feature does not only comply with Requirement R6 (data transported off the target system), but it also satisfies Requirement R11 (minimum network exposure of a target system). All that is required is outbound network connectivity so that the client systems can push data to the server. There is no need for the client to be visible via an open network port to the server (inbound client network access). This also means that the push based POFR information flow can traverse firewalls that can maintain outbound network access via NAT, VPN and mobile roaming technologies (Requirement R10).

POFR makes use of the Secure Shell (SSH) protocol [45] to achieve these features. SSH's connection oriented nature, as well as its embedded public key and symmetric encryption mechanisms preserve the confidentiality and integrity of the information across the transmission channel (Requirement R8). In addition, it provides a way to authenticate clients to the server, as well as authenticating the server to the client. Prior commencing data capture and transmission each client has to register with the POFR server. This client registration process [46] ensures a degree of isolation for each client (a POFR client can only send data to a unique server in its own dedicated communication and storage channel). Thus, if one client is compromised, it will not affect the data integrity of another client on the same server.

2.1 The POFR client volatile data collection

At the very heart of the POFR client live forensic data collection process is a sampling step of repeated high frequency parsing of the procfs filesystem [22]. Using high speed photography as a real world example, if we want to have a detailed view of a fast event in motion, we could employ a high speed camera, capable of taking several pictures per second. The largest the number of pictures taken per second, the greater the detail gained of the fast occurring event. The same principle applies for the sampling of the entities of procfs. A reference implementation of the parsing method is provided in the scanproc.pl script of the POFR utility [47]. In particular, snapshots of all entities of the /proc/[pid] subdirectory branch are collected and stored temporarily under the /dev/shm virtual memory based filesystem. These snapshots contain all necessary data to satisfy Requirement R1. Each /proc/[pid] entry contains all available information about running processes. The same can be said about the contents of /proc/net/tcp and /proc/net/udp in connection to network connections/endpoints.

For requirement R3 in particular, the POFR implementation [47] uses an adjustable microsecond scale sampling loop by means of the Perl based Time::HiRes [48] CPAN module. Smaller microsecond timer values will increase computational overhead. However, for most Linux systems a microsecond sampling delay of 300000 (0.3 seconds) maintains an accurate sample without imposing substantial CPU overhead on the target system. Experts can further tune the sampling delay based on specific evidence requirements.

It should be noted that the chosen volatile data collection method [47] is also compliant to Requirement R5. procfs has its own underlying performance based issues [49]. It is nevertheless a well proven standardized approach employed by many Linux monitoring tools. It does not require use of specialized kernel modules or other introspection techniques. Thus, it does not introduce complex software installation dependencies. All procfs parsing processes tasked with volatile data extraction have their software dependencies satisfied by the POFR in-built Perl interpreter environment. This environment is preconfigured by the POFR tool in a way that makes the installation of the client software an easy process [46].

Figure 2 provides a more modular description of the information flow from the POFR client to the server. This is important for clarifying various aspects that preserve the sequence of the recorded events (Requirement R2) as well as many other requirements that concern the integrity of the collected volatile data.

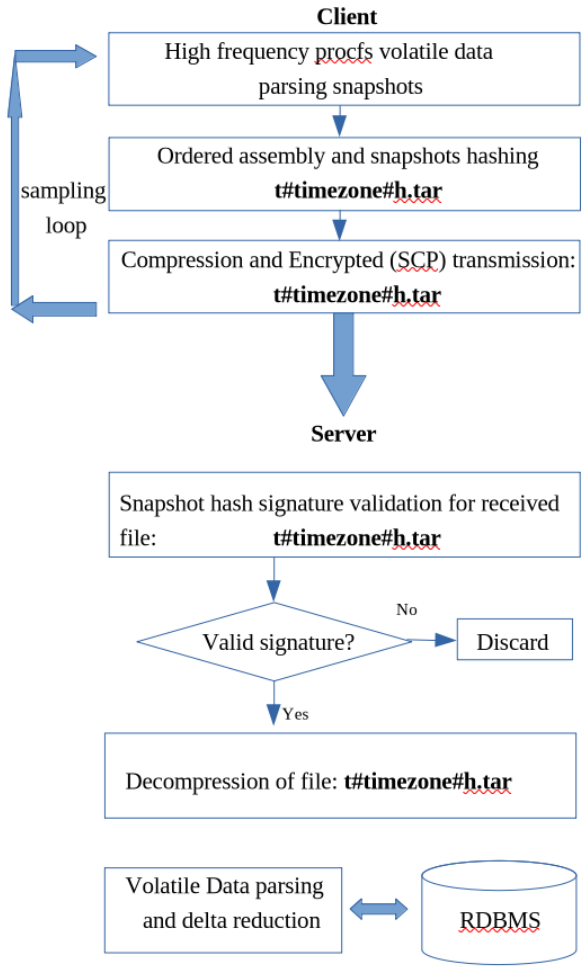

Figure 2: POFR volatile data flow

A key point of the volatile data collection process is the creation of the 't#timezone#h.tar' archive file. The timestamp t is a 16 digit string that contains the number of seconds from the Unix Epoch time [50] (10 digits) plus the elapsed microseconds (6 digits) that mark the file creation. Placing the time stamp as part of the file name is a practical way to preserve the sequence of the snapshot entries and ensure that the files will be transmitted on a FIFO basis (Requirement R2). The same t string is used at the server end to process the received files on the same FIFO basis. The 'timezone' bit represents the client's (target system) timezone so that the Epoch plus microsecond time stamp can be converted at a real date on the server (dd/mm/yyyy:hh:mm:ss:msec). Finally, h represents a suitably collision resistant SHA256 cryptographic hash of the entire file contents. Once again, encapsulating a hash signature as part of the file name provides a reasonable level of protection against the accidental or intentional (malicious) modification of the volatile data contents during their transmission (Requirements R6, R12, R15).

Having explained the naming, reliability and event sequence preservation mechanisms of the volatile data snapshot archive file, it is now time to focus on its contents. Each timestamped and cryptographically hash signed snapshot file contains a predetermined equal number of compressed process and network snapshots with the following naming convention:

t#timestamp.proc.gz

t#timestamp.net.gz

Data compression is a design choice aimed to minimize the overhead of the network bandwidth required to transfer the data to the server. As the information sent in the file is text based, it has low entropy and thus can be highly compressible. The reference implementation uses gzip compression which is a combination of the LZ77 [51] and Huffman encoding [52] algorithms. The proc.gz file contains all sensed processes and their open files at timestamp t. The net.gz file contains all TCP and UDP endpoints at timestamp t. The 't' and 'timestamp' strings are constructed as described in the previous paragraphs. This is a second level of event sequence preservation minus the cryptographic hash signature, which is applied on the entire archive file.

For the compressed process and open file snapshot data file, each process and file access record has the following form:

spid###pid###ppid###ruid###euid###rgid###egid###name###cmdline###openfile1openfile2openfilen

The string '###' is the field delimiter. The spid is a string that contains the source process and group id of the data collection process, separated by comma. This is the process that reads the procfs to take the snapshot. The reason it is recorded is to enable POFR to implement Requirement R7. By tagging the process id of the volatile data collection program, one can exclude the further analysis of its activities and its impact on the collected volatile data (observer effect). The pid and ppid strings represent the process id and parent process id of a particular process. It is also useful to log its real and effective user id data (ruid and euid), as well as its real and effective group id (rgid,egid). The difference/transition from the real to the effective user id of a process is important evidence that helps the examiner really determine whether a process has temporary access to files that require elevated privileges [53]. The 'name' string records the name of the executable application, whereas 'cmdline' contains the name of the executable application together with all the command line arguments that it was called with. The last record field

contains a white space separated list of all the files (absolute path) that the executable application is accessing.

For the latter compressed snapshot of the network endpoint data, the data format is:

spid###TCP4data###UDP4data###TCP6data###UDP6data

As with the process and file compressed snapshot file format, the string "###' is the field separator. The spid is a string that contains the source process and group id of the data collection process, separated by comma. The TCP4data, UDP4data, TCP6data and UDP6data sections represent the contents of the /proc/net/tcp, proc/net/udp, /proc/net/tcp6 and /proc/net/udp6 files respectively.

2.2 POFR data reception and difference processing at the server

The first processing step after receiving the file (encrypted SCP transfer) from the client is to check its integrity by means of verifying the cryptographic hash that forms part of the file name (Figure 2). If the hash is not verified the entire file is discarded. Anything that could accidentally or maliciously modify its contents during the transmission would compromise the integrity of the forensic process (Requirement R12). Once the integrity of the received file is verified the tar archive file will be opened with all its packed contents and the processing of the snapshot data shall begin.

The previous sections mentioned the Big Data nature of forensics [15] and imposed Requirement R16, in order to handle the ever increasing data volumes and processing requirements of forensic data sets. The previously presented procfs based snapshot data collection can create on average anything between 3-12 Gigabytes of compressed files per system on a daily basis. Each of these snapshots has to be parsed and recorded. If one collects simultaneously snapshots from tenths, hundreds or thousands of systems, it becomes evident that the storage and processing requirements can increase exponentially.

Prior choosing a technique to tackle the quantity of the data, it is critical to observe that the collected snapshot information has relatively low variance. Figure 3 projects the percentage of the generated new records over a time period of 12.5 seconds. At the start of the snapshot period, we have a lot of new records. However, the number of new process/file and network records drops quickly and with the exception of a few peaks, most of the new records are arithmetically below 20% of the overall number of collected snapshot records. The peaks are observed more often in busy target systems. For example, situations where a complex application is restarted or occasional peaks of activity that kickstart application work flows can produce such peaks of new records. The red line (Column C) represents data from a busy Linux desktop system, whereas the blue line represents a less busy web server (Column B).

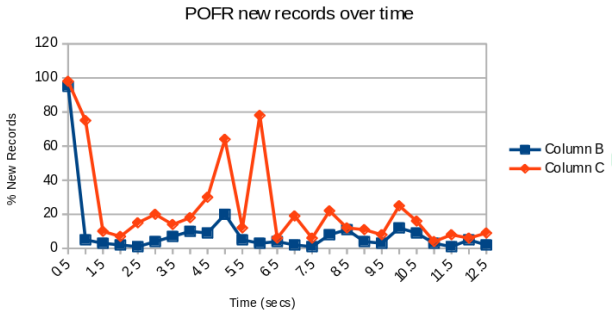

Figure 3: POFR new snapshot data over time

Delta encoding and its resulting differential compression [54] is a technique that enables storage or transmission of data in the form of differences (deltas) between sequential data rather than complete files. Its applications are widely used in data backup systems, version control software, as well as video encoding. In our problem domain, we have complete snapshot files, however we need to generate the delta from the complete files and use them to curate a non redundant, coherent version of snapshot events in a relational database (Figure 2). This is not so trivial and requires a careful orchestration of steps to achieve.

Figures 4 and 5 provide a high level algorithmic overview of how POFR performs the desired data reduction by means of data differencing.

The 'divide_data_set' algorithm of Figure 4 takes advantage of the multiple processing cores/threads of modern microprocessors. It initially constructs an ordered collection of all the validated snapshot data. This is implemented as a one dimension array (VectorCD) of all the valid files that were received by the client, in the order they were received. Each of the files contained in the Client Data (CD) contains process and network data. The next step is to break the vector CD into p different vectors if and only if we have more files than the number of vectors we wish to process simultaneously (n>=p). In such a case, each vector contains consecutive sequences of the files of the original CD vector. If n is the number of valid files collected by the client and p the number of distinct vectors we wish to make (p>=2), we continue processing the data. Once this breaking up of the original vector CD is complete, the algorithm will launch p instances of the data reduction algorithm 'deltaparse', each one executing

independently by the other. If we do not have more files than the number of vectors we wish to make (number of distinct instances) we differ the processing of the data until we do. This achieves a parallel processing approach of the dataset.

```
algorithm divide_data_set {
Pinstances=p
Nclientfiles=n
if (n>=p) {
 Vector CD={file1,file2,...filen}
 make subvectors(p) {
 VectorCD1={file1,file2,...filemaxCD1}
 VectorCD2={filemaxCD1+1,filemaxCD1+2,...file-
maxCD2}
      …
VectorCDp={filemaxCDp-1+1, filemaxCDp-1+2,…,filen}
}
for every Vector V (CD1, CD2,...CDp) {
 deltaparse(V)
}
} else {
wait_until (n>=p)
}
}
```

Figure 4: The 'divide_data_set' algorithm

Figure 5 outlines the data reduction 'deltaparse' algorithm. Each of the files divided by the 'divide_data_set' algorithm of Figure 4 provides an ordered collection of process, file and network endpoint data. They are all passed as Vector arguments (ProcessV, NetworkV) to each of the parallel instances of the deltaparse algorithm. The algorithm acts on the process data first, whereas the network data processing follows. The reason for this order is that from an operating system perspective, a network endpoint will always be created by a running process. The process will also interact with the filesystem (opening and closing ordinary and special files) before a network connection is created. Consequently, if one needs to reliably correlate network activity to users and applications, conclusions/facts about the presence of processes and files need to be established first.

```
algorithm deltaparse {
Vector V={ProcessV,NetworkV}
RefProcVector=first_file_p_data(Pfile1,ProcV)
populate_RDBMS(RefProcVector)
discard(Pfile1,ProcessV)
for every Pfile (ProcessV) {
 CProcVector=process_data_from_file(Pfile,ProcessV)
 DiffProcVector=diff_Vector(CProcVector,RefProcVector)
 if size_of_DiffProcVector == 0 {
  discard(PFile,ProcessV)
 } else {
   populate_RDBMS(DiffProcVector)
   discard(KFADFile,ProcessV)
 }
 }
RefNetVector=first_file_n_data(Nfile1,NetworkV)
populate_RDBMS(RefNetVector)
discard(Nfile1,NetworkV)
for every Nfile (NetworkV) {
 CNetVector=process_data_from_file(Nfile,NetworkV)
 DiffNetVector=diff_Vector(CNetVector,RefNetVector)
 if size_of_DiffNetVector == 0 {
  discard(PNfile,NetworkV)
 } else {
   populate_RDBMS(DiffNetVector)
   discard(PNfile,NetworkV)
 }
 }
 }
```

Figure 5: The 'deltaparse' algorithm

Starting with the process and file access data that are inside the ProcessV vector, one of the first things the algo-

rithm will do is to establish the initial picture of process and file activity. This is achieved in a First In First Out (FIFO) fashion by processing the very first of the files of the ProcessV vector (PFile1). The algorithm uses that data to construct a Reference Process Vector (RefProcVector). It will use that Vector to store the data to the Relational Database.

However, while the first file is discarded after that step, the reference vector is not. For every subsequent file in the ProcessV vector (Pfile), the algorithm will form the Current Process Vector (CprocVector) which contains all process and file data of the present process file in the loop. The heart of the data reduction step is to locate all the data of the Current ProcessVector that are not in the initially established Reference Process Vector. This is the Difference Process Vector (DiffProcVector). In set theory, this is the equivalent of the difference in two sets [55]. As the snapshot data are highly highly repetitive and thus redundant, the methodology detects easily only new process and file entries. If we have new entries, these will be entered into the database and the file will be discarded. In contrast, if the Difference Process Vector is empty, we just discard the file and continue in the loop.

The same methodology is applied to the network snapshot data files contained in the NetworkV vector.

For clarity purposes, the 'deltaparse' algorithm description excludes the details of the populate_RDBMS function. The POFR SQL table schema can be found in [56]. It can help the reader understand the level and the complexity of queries that a digital forensics expert can answer when mining the data. Samples of these queries include questions like:

- What sort of programs were executed at certain times and in what order?
- What files and/or network connections these programs opened/used and in what order?
- Did a program try to access a specific IP address and port number?
- Was a program run as a result (after) the presence of a network connection?
- Was a specific user or system account used to access certain files outside the normal access patent?
- Which countries did the connections originate from/went to in connection to certain port numbers?

Consequently, the data structure and query mechanisms satisfy Requirement R13, involving the SQL standard to store and process volatile data.

The 'divide_data_set' and 'deltaparse' algorithms are practically implemented as part of the POFR newdeltaparseproc.pl [57] module. Over a period of two months, the methodology is able to reduce 92.8 Gb of compressed snapshot data from a single client to just 146 Mb of raw SQL data. Moreover, the proposed methodology complies with Requirement R16, as it reduces data without loosing evidence.

Early deployment over hundreds of different production Linux systems has so far indicated that the greatest drawback of the 'deltaparse' algorithm is its weak performance when volatile data change frequently. However, this can only affect the processing time and never the accuracy of the collected volatile data.

Requirement R4 (length of data recording sessions to extend to months or years) has not been discussed yet. For reasons of computational efficiency (length of relational tables in terms of number of rows) the implementation of the deltaparse algorithm (Figure 5) restricts the length of the recording snapshot interval to a number of hours. If one wishes to produce recording sessions that extend to several months or years, POFR provides the mergearchive.pl utility [58]. The use of this utility is described in detail in the POFR Technical Operations and User Manual [46]. In essence, the utility merges all collected recording sessions into a single one in a non overlapping/redundant manner.

In order to demonstrate the ability of the mergearchive.pl utility to perform such a task successfully, the paper provides a public data set of such a recorded session [59]. The data set provides two months worth of POFR merged continuous recording sessions from a honeypot Linux server, together with sampled queries to demonstrate the effectiveness of the tool. Producing the merged data set from these two month data sampling period takes 10 hours on a single hypertheraded Intel Coffee Lake Xeon(R) E-2286M processor core.

Conclusions

This paper presented a systemic review of the challenges of the live digital forensics field. After reviewing relevant literature, sixteen functional requirements were proposed. These requirements ensure that volatile data can be collected in a reliable, evidence oriented manner, without overloading the target systems. The goal is to produce data sets that can help the forensic examiner to easily relate sequences of process execution, file access and network endpoint creation events.

POFR is an Open Source live forensics implementation for the Linux Operating system that attempts to meet all the

proposed functional requirements. It implements an easily installed tool that can provide reliable data sets for a forensic examiner. In addition, POFR can deal with the Big Data nature of live forensics by providing demonstrably adequate data reduction levels without compromising the accuracy of the collected evidence.

Acknowledgments

Steelcyber Scientific has donated employment time and infrastructure equipment to this Open Source project. The author would also like to acknowledge the support of the Norwegian Meteorological Institute's IT Infrastructure team.

References

1. Eisenstein, M. Big data: The power of petabytes. Nature 527, S2–S4, 2015: https://doi.org/10.1038/527S2a

2. Y. Sasaki, "A Survey on IoT Big Data Analytic Systems: Current and Future," in IEEE Internet of Things Journal, vol. 9, no. 2, pp. 1024-1036, 15 Jan.15, 2022, doi: 10.1109/JIOT.2021.3131724.

3. Mahdi Dibaei, Xi Zheng, Kun Jiang, Robert Abbas, Shigang Liu, Yuexin Zhang, Yang Xiang, Shui Yu, Attacks and defences on intelligent connected vehicles: a survey,Digital Communications and Networks, Volume 6, Issue 4,2020, Pages 399-421, SSN 2352-8648, https://doi.org/10.1016/j.dcan.2020.04.007.

4. Thiago Alves, Rishabh Das, Aaron Werth, Thomas Morris, Virtualization of SCADA testbeds for cybersecurity research: A modular approach, Computers & Security,Volume 77, 2018, Pages 531-546, ISSN 0167-4048,https://doi.org/10.1016/j.cose.2018.05.002

5. B. Hay, M. Bishop and K. Nance, "Live Analysis: Progress and Challenges," in IEEE Security & Privacy, vol. 7, no. 2, pp. 30-37, March-April 2009, doi: 10.1109/MSP.2009.43.

6. Sutherland, Iain et al. "Acquiring volatile operating system data tools and techniques." Operating Systems Review 42 (2008): 65-73.

7. A. Vazão, L. Santos, M. B. Piedade and C. Rabadão, "SIEM Open Source Solutions: A Comparative Study," 2019 14th Iberian Conference on Information Systems and Technologies (CISTI), Coimbra, Portugal, 2019, pp. 1-5, doi:10.23919/CISTI.2019.8760980.

8. P. J. Divya, R. S. George, G. Madhusudhan and S. Padmasree, "Organization-wide IOC Monitoring and Security Compliance in Endpoints using Open Source Tools," 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2022, pp. 1-6, doi: 10.1109/GCAT55367.2022.9971999.

9.The osquery tool online documentation,https://osquery.readthedocs.io/en/stable/

10. Casey, E., 2017. The broadening horizons of digital investigation. Editor. Digital Investigation, 21, 1e2.

11. Garfinkel, S.L.. "Digital forensics research: The next 10 years", Digit. Invest, 2010, Pages S64-S73, https://doi.org/10.1016/j.diin.2010.05.009

12. Grajeda, C., Breitinger, F., & Baggili, I. "Availability of datasets for digital forensics–And what is missing". Digital Investigation, 22, 2017, https://doi.org/10.1016/j.diin.2017.06.004

13. Wu, T., Breitinger, F., O'Shaughnessy, S. "Digital forensic tools: Recent advances and enhancing the status quo". Digital Investigation, 34,2020, https://doi.org/10.1016/j.fsidi.2020.300999

14. F. Y. W. Law, K. P. Chow, M. Y. K. Kwan and P. K. Y. Lai, "Consistency Issue on Live Systems Forensics," Future Generation Communication and Networking (FGCN 2007), Jeju, Korea (South), pp. 136-140, 2007: doi: 10.1109/FGCN.2007.93.

15. M. Qi, Y. Liu, L. Lu, J. Liu and M. Li, "Big Data Management in Digital Forensics," 2014 IEEE 17th International Conference on Computational Science and Engineering, Chengdu, China, 2pp. 238-243, 2014: doi: 10.1109/CSE.2014.74.

16. Raincock S. "Sometimes it's all about timing", Forensic Focus portal, 2010: https://www.forensicfocus.com/articles/sometimes-its-all-about-timing/

17. Bach M.J. "The Design of the Unix Operating System", Prentice-Hall International, 1986, ISBN: 0-13-201757-1, page 10 provides the defition of the term process.

18. Adelstein F. "Live Forensics: Diagnosing your system without killing it first", Communications of the ACM, Vol 49, No. 2, https://doi.org/10.1145/1113034.1113070

19. Sandvik J.P, Franke K., Abie H., Årnes A. "Quantifying data volatility for IoT forensics with examples from Contiki OS", Forensic Science International: Digital Investigation, Vol. 40, Supp, https://doi.org/10.1016/j.fsidi.2022.301343

20. J. Jones and L. Etzkorn, "Analysis of digital forensics live system acquisition methods to achieve optimal evidence preservation," SoutheastCon 2016, Norfolk, VA, USA, 2016, pp. 1-6, doi: 10.1109/SECON.2016.7506709.

21. Pogue C., Altheide C., Haverkos T. “UNIX and Linux Forensic Analysis DVD Toolkit”, SYNGRESS, 2008: ISBN: 978-1-59749-269-0

22. The procfs wikipedia page: https://en.wikipedia.org/wiki/Procfs

23. Paganini P. “Experts spotted Syslogk, a Linux rootkit under development”, Security Affairs website, 2022: https://securityaffairs.co/132232/malware/syslogk-linux-rootkit.html

24. Malin C. H., Casey E., Aquilina J. M. “Malware Forensics Field Guide for Linux Systems”, SYNGRESS, 2014: ISBN: 978-1-59749-470-0

25. The Userspace API comminity group Unified Kernel Image specification git repository: https://github.com/uapi-group/specifications/blob/main/specs/unified_kernel_image.md

26. Hash function Wikipedia page: https://en.wikipedia.org/wiki/Hash_function

27. Conlan K., Bagilli I., Breitinger F. “Anti-forensics: Furthering digital forensic science through a new extended, granular taxonomy”, Digital Investigation, 18, 2016: http://dx.doi.org/10.1016/j.diin.2016.04.006

28. Koivusalo E. “Converged Communications: Evolution from Telephony to 5G Mobile Internet”, Wiley-IEEE Press, 2022: ISBN: 978-1119867500

29. D. Wing, "Network Address Translation: Extending the Internet Address Space," in IEEE Internet Computing, vol. 14, no. 4, pp. 66-70, July-Aug. 2010, doi: 10.1109/MIC.2010.96.

30. A. Alshalan, S. Pisharody and D. Huang, "A Survey of Mobile VPN Technologies," in IEEE Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1177-1196, Secondquarter 2016, doi: 10.1109/COMST.2015.2496624.

31. E. Schiller, A. Aidoo, J. Fuhrer, J. Stahl, M. Ziörjen, B. Stiller, Landscape of IoT security,Comp. Sci. Rev., 44 (2022),https://doi.org/10.1016/j.cosrev.2022.100467

32. Magklaras G, Furnell S., Papadaki M. “LUARM: An Audit Engine for Insider Misuse Detection.”, Int. J. Digit. Crime Forensics, 2011,https://doi.org/10.4018/jdcf.2011070103

33. Codd, E.F. “The relational model for database management: Version 2”, Addison-Wesley, 1990, ISBN: 9780201141924.

34. ISO/IEC 9075-1:2016 Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework): https://www.iso.org/standard/63555.html

35. Bernstein P., Newcomer E. “Principles of Transaction Processing (2nd ed.). Morgan Kaufmann (Elsevier), 2002. ISBN 978-1-55860-623-4

36. Cattell R. “Scalable SQL and NoSQL data stores”, SIGMOD Record, vol. 39, no. 4, pp. 12-27, 2010, https://dl.acm.org/doi/10.1145/1978915.1978919

37. Ehrenberg, Andrew S. C. “A Primer in Data Reduction: An Introductory Statistics Textbook.” New York: Wiley, 1982. ISBN 0-471-10134-6.

38. Xia W., Jiang H., Feng D., Tian L., Fu M., Zhou Y. “Ddelta: A deduplication-inspired fast delta compression approach”, Performance Evaluation 79, pp 258-272, 2014, http://dx.doi.org/10.1016/j.peva.2014.07.016

39. Dongyang Li et al., "Hardware accelerator for similarity based data dedupe," 2015 IEEE International Conference on Networking, Architecture and Storage (NAS), Boston, MA, USA, 2015, pp. 224-232, doi: 10.1109/NAS.2015.7255198.

40. Guido M., Buttner J., Grover J. “Rapid differential forensic imaging of mobile devices”, Digital Investigation, Vol 18, pp. 46-54, 2016, https://doi.org/10.1016/j.diin.2016.04.012

41. Quick, D., Choo, KK.R. Big forensic data reduction: digital forensic images and electronic evidence. Cluster Comput 19, 723–740 (2016). https://doi.org/10.1007/s10586-016-0553-1

42. Hossain M. N., Wang J., Sekar R., Stoller S. D., “Dependence-Preserving Data Compaction for Scalable Forensic Analysis, Proceedings of the 27th USENIX Security Symposium, Baltimore, MD, USA, 2018, ISBN 978-1-931971-46-1

43. Peng T., Jiang M., Yang X., Song K., Hu M., Wei X., Liang J. (2016), “Evidence-obtaining data reduction method based on spatial statistics”, UNIV WUHAN TEXTILE, Patent Application Number CN106228173 (A) — 2016-12-14

44. POFR - Penguin OS Forensic (or Flight) Recorder github repository: https://github.com/gmagklaras/POFR

45. Network Working Group of the IETF (2006), RFC 4251, The Secure Shell (SSH) Protocol Architecture, URL: https://tools.ietf.org/html/rfc4251

46. POFR – Penguin OS Forensic Recorder Technical Operations and User Manual: https://github.com/gmagklaras/POFR/blob/main/doc/POFRmanual.pdf

47. POFR scanproc.pl script: https://github.com/gmagklaras/POFR/blob/main/client/scanproc.pl

48. The Perl Time::HiRes CPAN module: https://perl-doc.perl.org/Time::HiRes

49. sudonull website. “How effective is the procfs virtual file system and can it be optimized?”, 2019: https://sudonull.com/post/13135-How-effective-is-the-procfs-virtual-file-system-and-can-it-be-optimized

50. Matthew N., Stones R. "The Linux Environment", Beginning Linux Programming, Indianapolis, Indiana, US: Wiley, page 148, 2008, ISBN 978-0-470-14762-7

51. Ziv J., Lempel A. (1977), "A Universal Algorithm for Sequential Data Compression”, IEEE Transactions on Information Theory, 23 (3), pages 337–343. DOI:10.1109/TIT.1977.1055714.

52. Huffman D. (1952), "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the IRE, 40 (9), pages: 1098–1101, DOI:10.1109/JRPROC.1952.273898

53. Kerrisk M. (2010), “The Linux Programming Interface, A Linux and Unix Programming Handbook”, No Starch Press, ISBN 978-1-59327-220-3

54. Delta encoding Wikipedia page: https://en.wikipedia.org/wiki/Delta_encoding

55. Devlin K.J. (1979), “Fundamentals of contemporary set theory”, Springer, ISBN 0-387-90441-7

56. The POFR SQL schema: https://github.com/gmagklaras/POFR/blob/main/server/itpslschema.sql

57. The POFR newdeltaparseproc.pl module at github: https://github.com/gmagklaras/POFR/blob/main/server/newdeltaparseproc.pl

58. The POFR mergearchive.pl utility:https://github.com/gmagklaras/POFR/blob/main/server/mergearchive.pl

59. The POFR reduced data reference dataset: https://www.steelcyber.com/georgioshome/other/refdata/