PANDAcap: A Framework for Streamlining Collection of Full-System Traces

Manolis Stamatogiannakis
Vrije Universiteit Amsterdam
The Netherlands
manolis.stamatogiannakis@vu.nl

Herbert Bos
Vrije Universiteit Amsterdam
The Netherlands
h.j.bos@vu.nl

Paul Groth
University of Amsterdam
The Netherlands
p.groth@uva.nl

ABSTRACT

Full-system, deterministic record and replay has proven to be an invaluable tool for reverse engineering and systems analysis. However, acquiring a full-system recording typically involves significant planning and manual effort. This represents a distraction from the actual goal of recording a trace, i.e. analyzing it. We present PANDAcap, a framework based on PANDA full-system record and replay tool. PANDAcap combines off-the-shelf and custom-built components in order to streamline the process of recording PANDA traces. More importantly, in addition to making the setup of one-off experiments easier, PANDAcap also caters to the streamlining of systematic repeatable experiments in order to create *PANDA trace datasets*. As a demonstration, we have used PANDAcap to deploy an *ssh honeypot* aiming to study the actions of brute-force *ssh* attacks.

KEYWORDS

framework, PANDA, record and replay, docker, honeypot, dataset

ACM Reference Format:
Manolis Stamatogiannakis, Herbert Bos, and Paul Groth. 2020. PANDAcap: A Framework for Streamlining Collection of Full-System Traces. In *13th European Workshop on Systems Security (EuroSec '20), April 27, 2020, Heraklion, Greece*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3380786.3391396

1 INTRODUCTION

Full system record and replay has proven to be a powerful tool for a variety of tasks including debugging [6, 12, 18], intrusion detection [11, 23], security fault analysis [6] and data provenance [26]. In the context of security, full system recordings enable rare conditions to be detected and potential attack vectors to be studied more deeply. Typically, full system recordings have been used in scenarios where detailed analysis is necessary and the potential insights gained are worth the effort involved in setting up the recording environment. In this work, we aim to *dramatically lower the effort* to 1) provision an environment where full system record and replay is enabled; and 2) analyze the resulting recordings. By reducing this barrier, we are able to *expand the areas* where full system record and replay can be potentially applied. In particular, we look at the systematic collection of trace datasets for the purposes of security analysis, which we illustrate with a classic honeypot case study.

EuroSec '20, April 27, 2020, Heraklion, Greece

In addition, we believe the framework can provide both an environment for computational reproducibility as well as a foundation for capturing systems oriented datasets. Lack of reproducibility and (verifiable) datasets are hurting security research at large [21, 28]. Moreover, the ability to replay *all* events in arbitrary operations rather than logging specific features for a specific experiment allows the use of data sets for multiple use cases, in more versatile ways.

The contributions of this paper are as follows:

- the PANDAcap framework for streamlining the process of recording PANDA based traces;
- a dataset of ssh honeypot traces, analytics over those traces and the procedures to expand the dataset.

The rest of this paper is organized as follows. In §2 we briefly present some work that inspired us. Next, in §3 we present PANDAcap, followed by a case study in §4. Finally, in §5 we discuss some issues related to our system and we conclude in §6.

2 RELATED WORK

Our work is inspired by work on environments for computational reproducibility [21, 27]. ReproZip [5] automatically analyzes system calls to create bundles of files and programs that can be executed in a virtual machine or Docker image. Likewise, [20] describes a system for building Docker based virtual machines for R environments. Indeed, Docker has become a foundation for a number efforts to ensure computational reproducibility (e.g., [3, 14, 16]). Beyond just capturing computational artifacts in a container, other approaches, such as the WholeTale[4] and CWLProv [15], aim to incorporate datasets, papers into larger so called research objects [1] to further facilitate reproducibility. More broadly reproducibility is becoming an increasing concern in cybersecurity research [8]. For a wider view of computational reproducibility and particular its relevance to notions of data provenance, we recommend the recent survey by Pimentel et al. [22]. Our work builds on these approaches by adding full system recording capability. Additionally, our work sees the outcome not to be about reproducibility but to provide datasets for *analytics*.

3 PANDACAP

PANDA offers an excellent platform for deep, full-system analysis [25, 26]. In addition, PANDA is well-suited for creating and sharing systems-related datasets, offering researchers a platform to iterate and improve their methods using a common baseline. However, this potential is not fully realized in practice and the sharing of datasets created with PANDA is lacking. We developed PANDAcap to fill this gap and help PANDA to gain traction as

the de-facto tool for creating datasets for systems and security research. The target user group of PANDAcap is researchers who are already familiar with working in a Unix environment, but are not intimate with Docker or PANDA so they can quickly spin-up a dataset collection pipeline.

PANDA traces contain a precise log of the code and data that the program accessed during recording. Because of their serial nature these traces may be time-consuming to analyze without having any context. For this purpose, we decided to also separately record the PANDA VM *disk delta*. While the disk deltas will not add any execution-related information that is not also contained in the traces, they may give useful hints for how to proceed with the analysis of an unknown trace.

We envision several applications for PANDAcap. These applications are mostly related to the creation of *security-related datasets*, but also extend outside the strict borders of security research. For instance, we can use PANDAcap for applications such as:

- **Analyzing samples from malware datasets.** PANDAcap can be used to automate the collection of traces from existing datasets, similar to, but more versatile than, Malrec's full-trace recording of malware [25]. PANDAcap offers a streamlined way to build such systems by simply plugging in any experiment-specific customizations.
- **Honeypot deployment.** Sometimes it is desirable to deploy honeypot systems for studying attack trends in a network. Typically, full-system honeypots require a fair amount of effort to be prepared. Moreover, they typically permit mostly post-mortem analysis. With the help of PANDAcap, it now becomes easier to deploy PANDA based honeypots. We present an example of such a deployment in §4.
- **User studies.** PANDAcap can also find applications in domains other than systems and security. For example, it can be used to conduct a study on users in order to gain insights on their use of tools and data. Traces collected from such a study could be used in the fields of data provenance and process mining.

To support such different use cases, PANDAcap provides support for containerized system deployment, recording and replaying facilities, and powerful control primitives. In the remainder of this section, we describe the components comprising PANDAcap, the corresponding control primitives, the bootstrap procedure, and finally the PANDAcap support wrapper.

3.1 Off-the-shelf components

Intended as a low-maintenance, multi-purpose framework for the community rather than a one-off academic prototype, we developed PANDAcap on top of two widely-used, off-the-shelf components.

3.1.1 The PANDA record and replay framework. First, PANDAcap builds on PANDA [9, 10], an open-source, full-system record and replay tool based on QEMU [2]. PANDA records traces comprised of *a*) an initial dump of the QEMU guest memory, and *b*) a log file containing all the non-deterministic inputs used during the recorded execution. This information allows one to re-enact the exact execution during the replay of the trace. To facilitate the analysis of traces, PANDA features a *plugin architecture* which allows writing analysis modules in C, C++, and Python. Analysis plugins can insert instrumentation at different execution points of the analyzed trace: per instruction, per memory access, per context switch etc. PANDA also offers a communication mechanism for plugins to interact with each other, thus building complex functionality from many simple analysis modules.

3.1.2 Docker. Second, PANDAcap uses Docker [17] for easy deployment and containment. Docker is currently the de facto standard for lightweight virtualization. Adoption of Docker in research is already high, driven by the need for reproducibility. We create a *Docker image* to bundle the runtime dependencies of PANDA, and instantiate identical environments for PANDA VMs to run. Furthermore, PANDAcap uses Docker to virtualize the networking of PANDA VMs as well as the storage space used when running an experiment.

3.2 The *recctrl* plugin

The *recctrl* plugin is the key building block we developed for PANDAcap. Typically, starting recording a PANDA trace involves typing the `begin_record` command on the QEMU monitor prompt. This interaction can be scripted using the *QEMU Machine Protocol* (QMP), but there's still need for manual intervention to properly time the start and the end of the recording. This may be acceptable for one-off recordings, but becomes a problem when one wants to collect a large number of recordings.

The *recctrl* plugin addresses this problem, by adding a special *hypercall* to the PANDA VM that allows guest programs to signal when recording should start or stop. This allows to automate recording of traces, removing the need for manual intervention or scripting. Moreover, the captured traces will only contain the desired part of the execution.

Unlike typical PANDA analysis plugins, *recctrl* is meant to operate exclusively on a *live VM*, rather than a VM replaying a trace. Its implementation piggybacks on existing, unprivileged instructions of the emulated architectures—`CPUID` for x86, `MCR` for ARM. The plugin requires a magic value to be set in one of the instruction input registers in order to process it. Because, overall, the frequency of the instructions triggering *recctrl* is very low, the performance impact on the live VM is negligible.

To cater for different types of experiments, *recctrl* offers several runtime configuration options. First, it implements two modes of operation. In *toggle mode*, recording starts on the first hypercall and stops on the following. In *semaphore mode*, hypercalls can be made to either increase or decrease a session counter. Recording starts/stops when the counter raises from/falls back to zero. Moreover, *recctrl* allows to limit the number of recordings and the duration of each recording. When the set number of recordings is exceeded, the PANDA VM is powered off.

Finally, to avoid having to modify programs to include the new hypercall, the plugin comes with a small utility program called *recctrlu*, providing a command-line interface to the hypercall. The utility can be easily invoked from existing hooks offered by the guest operating system, catering for a variety of scenarios. E.g. *recctrlu* can be hooked with the Linux PAM authentication framework in order to start recording when a user logs in via *ssh*. Or, it can be hooked to the `iptables` network firewall to start recording on an

incoming HTTP connection. The code of both the plugin and the utility is portable and should work on Linux and Windows.

3.3 Bootstrapping PANDAcap

PANDAcap offers the capability to insert bootstrapping code at different phases of preparing an experiment. This gives a lot of flexibility in designing and building the experiment workflow. Bootstrapping has been automated using GNU Make where possible. A central configuration file called Makefile.vars is used to define options in a single location. In order to propagate the options to scripts written in other languages, we use the *Jinja templating language*[1] as a preprocessor for those scripts. Next, we describe the different bootstrapping phases used by PANDAcap.

3.3.1 Docker image bootstrapping. The instructions for building a Docker image are typically contained in a template file called *Dockerfile*. The file uses a domain-specific language to list the instructions. The main PANDA repository included a Dockerfile, however we found it to be mostly suited for automated testing of PANDA builds, rather than deploying the tool inside a container. The reason is that the template pulls inside the container all the PANDA *build-time dependencies*. This result in unnecessary bloat in the resulting image.

Figure 1 shows the process of creating the PANDAcap Docker image. We can see that PANDA is added to the image as a precompiled tar archive and only its runtime dependencies are installed, resulting in a slimmer image. Bootstrap scripts are also installed as a tar archive and run in a single step. The scripts from the archive are executed in lexicographical order. Using this process instead of using multiple RUN commands in the Dockerfile helps us avoid creating redundant image checkpoints. Moreover, it helps to make the Dockerfile easier to reuse, as any experiment-specific customization can be applied through the bootstrap scripts.

Finally, we should mention that the PANDAcap Docker image uses baseimage-docker[2] as its base. This provides us with some desired features, such as a proper init process and the *ssh* access to the container. But more importantly, it provides us with a hook for our runtime bootstrapping scripts (see §3.3.3), so that they run when we instantiate a container from the image.

3.3.2 PANDA VM customization. In addition to any customization required for running a specific experiment, the only customization required for using an existing PANDA VM image with PANDAcap is to *a*) copy a single script somewhere in the filesystem, and *b*) add it to rc.local startup script, so it will be executed at the end of the VM boot process Optionally, the *recctrl* utility can also be copied to the VM at this phase, although this is not strictly required.

The copied script will take care of the runtime bootstrapping when we run the VM. The script looks whether a virtual USB drive has been inserted. If one is found, it is mounted and the bootstrap.sh script will be executed from it (see §3.3.3). Finally, the script will unmount and disconnect the USB drive, cleanup rc.local and remove itself from the system. This is a low-effort attempt to obscure the presence of PANDAcap components in the VM. We should note that currently, the boot-time VM customization script has only been implemented for Linux.

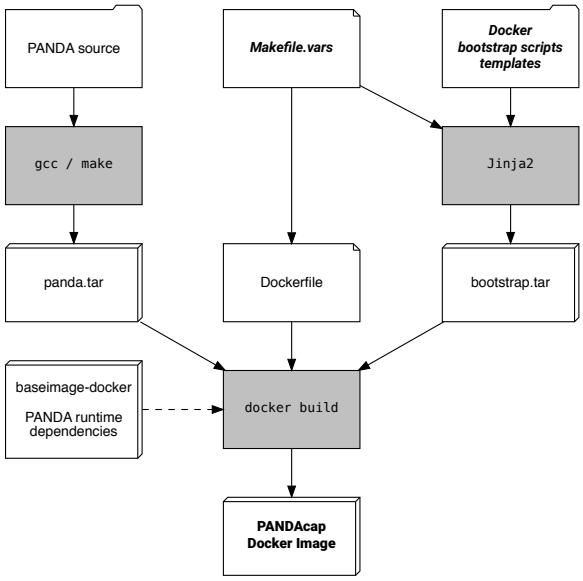

Figure 1: PANDAcap Docker image creation process. A dashed edge is used for components downloaded externally. Typically, only the nodes with labels in bold-italic print require customization before an experiment.

3.3.3 Runtime bootstrapping. PANDAcap offers support for runtime customization for both the PANDA VM used to capture a trace and the Docker container it runs in. As we mentioned in §3.3.1 and §3.3.2, this is implemented by hooking to the baseimage-docker startup and the rc.local script respectively. Similar to what we described in §3.3.1, the startup scripts and any files they need to generate have access to the configuration options in Makefile.vars, can be generated using the Jinja template language.

PANDAcap provides a sample Makefile for quickly creating a set of bootstrap scripts. However its use is not mandated. Furthermore, users are free to choose whether the runtime bootstrap scripts used for capturing different samples will be batch-generated at the start of the experiment or if they will generated at the beginning of each run.

3.4 The PANDAcap wrapper

While PANDA and Docker are very feature-rich, combining them to record a trace for a specific experiment can be a significant challenge. This is because of the sheer amount of options they offer: The PANDA binary lists *over 170 command-line flags and switches*, most of them inherited from QEMU. Adding to that, Docker's run subcommand alone adds *over 90 flags and switches*. Even after finding the right options to use, the resulting command line is extremely long and unwieldy[3].

For this we created the *pandacap.py* wrapper around the two tools. The goal of the wrapper is to help researchers to set up a

[1] The Jinja templating language: https://jinja.palletsprojects.com/
[2] baseimage-docker: https://github.com/phusion/baseimage-docker

[3] The command line for the case study in §4 spans 10 lines in a standard 80col terminal.

PANDA-based experiment, without requiring them to have extended experience with QEMU, Docker and Linux system-administration. The wrapper script is written in Python and employs the powerful argparse module of the language. It provides reasonable defaults and shortcuts for the most commonly used options, and handles the intermediate steps required to launch an experiment. Following, we briefly describe the main amenities offered by *pandacap.py*.

Transparently handles the creation of VM derived images. This is an essential part for efficiently supporting running multiple VM instances concurrently. The derived images are essentially deltas over the initial VM image. Without using them, we would have to provide each PANDA VM instance with a separate copy of the VM image. Thus, the use of derived images both saves storage space and makes launching a new VM instance faster.

Transparently handles the creation of the virtual disk used for VM bootstrapping. As we mentioned in §3.3.3, PANDAcap supports bootstrapping a VM at runtime via a virtual USB drive. *pandacap.py* handles the creation of this virtual drive from the contents of a directory. This removes the need for the user to learn about aspects of administration of Linux filesystems.

Supports naming each run. This is important for experiments involving the collection of many individual samples. The run name is propagated to the names of the support files generated by *pandacap.py*, making their management easier.

Python integration. PANDAcap can be imported as Python module, so it can be used as a building block for more complex setups. E.g. it can be integrated with a Flask[4]web application to enable controlling a PANDAcap workflow over the web.

Use of Docker is optional. We believe that using Docker makes it easier to manage multiple PANDA instances running concurrently. However, for simple cases its use may be seen as an unnecessary complication. For example, when debugging a PANDA plugin, one may prefer to run everything directly on the host. For this, *pandacap.py* also works without Docker and will produce an appropriate command line if no Docker-specific options are supplied.

4 CASE STUDY: SSH HONEYPOT

To demonstrate the use of PANDAcap, we used it to configure a PANDA VM to operate as an ssh honeypot. *Brute-forcing of ssh passwords* is still a very common type of attack today [13]. Attackers identify reachable ssh servers on the internet and repeatedly try to guess the password of a user—typically root—to gain access to the server. All recent OS distributions come with the ssh service configured with secure defaults that would prevent these attacks. However, for reasons of convenience, users often revert to insecure configurations. As a result, ssh brute-force attacks still remain viable and popular. Despite their low sophistication, a study of such attacks would be interesting, in order to gain an understanding of the tools used as well as the incentives of the attackers. As attackers often remove their traces, using PANDAcap to capture a series of incidents could provide us with much more insight than a simple post-mortem analysis on a host.

[4]Flask micro web framework: https://flask.palletsprojects.com/

Table 1: Collected samples per *ssh* port. No attempts to gain access to the VM listening on port 2200 were made.

port	samples	nondet	nondet-gz	disk-delta
22	50	9.61 GiB	2.75 GiB	11.49 GiB
2222	13	0.99 GiB	0.28 GiB	3.00 GiB

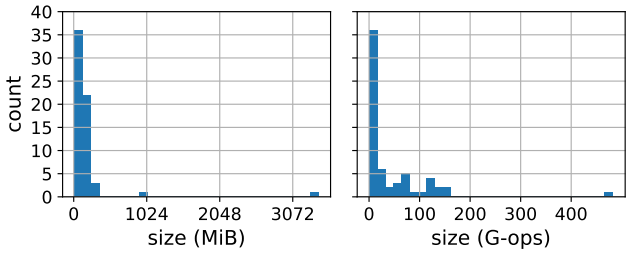

Figure 2: Trace size and instruction count distributions.

4.1 Experiment setup

We deployed 3 instances of our honeypot VM on a publicly accessible host for a period of approximately 3 days. The VMs were listening for ssh connections on ports 22, 2222, and 2200 respectively. Waiting for attackers to actually guess a password was deemed impractical. For this, the Linux PAM authentication framework was tuned to *a*) accept any password for user root, and *b*) run *recctrlu* on each successful ssh login. This reconfiguration was performed at runtime, as described in §3.3.3. The Docker container needed no particular runtime configuration for this experiment.

recctrl was configured to run in semaphore mode (see §3.2) and capture a single trace, for 30min after the first successful login. We used this setup because our initial tests showed that attackers used a single command per *ssh* connection. Under this usage pattern, the regular use of semaphore mode would result in many tiny traces.

We used supervisord[5]to reset and restart each VM after it finished recording a trace. Additionally, supervisord provided a logging facility for the messages emitted by the running VMs. In order to prevent the case where the same small group of hosts would intrude our VMs again and again, we periodically blocked the IP addresses we found listed in the *ssh* access logs.

4.2 Collected dataset

During the course of our experiment, we gathered a total of *63 traces*, and their associated VM disk deltas. Links for downloading the dataset are provided at the end of the paper. The size of the samples collected per port can be seen in Table 1. We should mention that non-determinism logs also include much of the information in disk deltas. We decided to preserve disk deltas in order to allow for direct analysis with existing disk forensic tools. In Figure 2 we show the distributions of the size and the instruction count of the traces. We observe that the distributions are heavy on their left side, but they do not match exactly. This indicates that there are variations in the workload inside the traces.

[5]Python supervisor: http://supervisord.org/

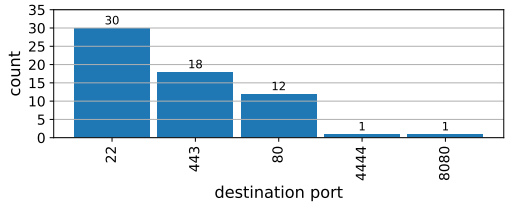

Figure 3: Top target ports for outgoing connections. In one trace, there were no outgoing connections.

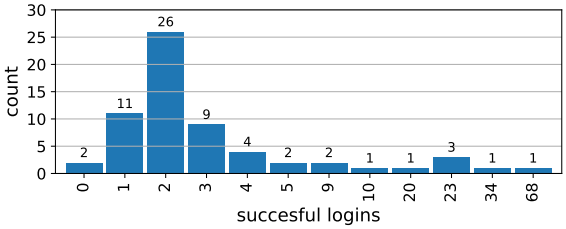

Figure 4: Succesful logins attempts in auth.log.

4.3 Trace analysis

Following, we make a quick analysis of some aspects of the collected traces. The analysis is not meant to be exhaustive. Our aim is *to present different types of analysis* that are possible using the data collected, and give some starting pointers on what can be done with the released dataset.

4.3.1 Scanning activity. Hosts compromised by *ssh* brute-force attacks are often used for scanning for more hosts with weak passwords. We would like to know how often this happened in our dataset. For this, we used the PANDA network plugin on the collected traces to extract their network activity as pcap traces.

Querying the pcap traces with tcpdump, we counted the top destination port for outgoing packets that have the SYN flag set. The distribution of these ports is shown in Figure 3. We can see that around half of the honeypot VMs started scanning for more weak hosts within 30 min from getting compromised.

4.3.2 Number of successful root logins. To demonstrate the usefuleness of having VM disk deltas along PANDA traces, we used them to inspect how many times we had a successful root login on each VM. The results are presented in Figure 4.

Inspecting the figure can help us to quickly infer information about the collected traces, before processing them with some PANDA plugin. For example, if we want to study what type of cleanup an intruder performs, we can see that two traces have aggressively cleaned-up all log entries for root. These would be a good candidate for further inspection.

Similarly, if we are interested to study a rootkit that leaves behind a known file, we can identify which traces were infected by it using familiar command-line tools. Then we can switch to PANDA for a more comprehensive analysis.

4.3.3 Cleanup methodology. Wanting to investigate the cleanup methodology used by the intruder, we wrote a plugin that for each system call prints *a*) the process id and name that ran the call, *b*) the system call name, and *c*) the values of all of its string arguments. We ran it on one of the two traces with the thorough cleanup. Unsurprisingly, the auth.log file was among the files touched by the rm command and the unlinkat system call. However, many of the other files affected by the same command/system call were known to be non-existent. This indicates a blind cleanup action, rather than targeted hidding of the intruder's trails.

5 DISCUSSION

5.1 Comparison with Malrec

PANDAcap and Malrec [25] both leverage PANDA to capture high-fidelity execution traces. However the two projects differ in scope. Malrec was designed with a single application in mind: converting a feed of malware samples to PANDA traces. The disclaimer on the project's GitHub page[6] notes that heavy modifications would be needed to adapt to other scenarios. On the other hand, PANDAcap has been designed to be reusable. Towards this end, it supports integration with Docker that allows virtualization of storage and networking, and includes a streamlined bootstrapping mechanism for easy customization for different use cases.

5.2 PANDAcap as a honeypot solution

In their survey, Nawrocki et al. [19] present different honeypot taxonomies. A honeypot based on PANDAcap would rank as a *high-interaction*, *virtual* honeypot. Depending on the bootstrapping, it can operate either as *server* or *client* honeypot.

The high-interaction honeypots listed in [19] typically focus on a single type of analysis. For example, Argos [24] focuses on taint analysis of network inputs. Moreover, transient effects on the system typically will not be captured. Contrasting to this, honeypots based on PANDAcap offer significant advantages because of the underlying PANDA VM record and replay capabilities. Specifically, *a*) the analyst doesn't have to decide *a priori* on the type of analysis, thus providing for flexible analysis, and *b*) all the transient effects of an attack are captured, thus resulting in higher *accuracy* of the analysis.

5.3 Detectability

Offensive tools and malware have evolved to avoid analysis by changing their behaviour in the presence of an analysis environment [7]. For the case of an analysis sandbox based on PANDAcap, the QEMU emulator underlying PANDA would be the main source of environment artifacts that can be fingerprinted by malware. For this, we did not attempt to fully conceal the presence of components specific to PANDAcap, like *recctrlu*. In principle, if we ever identify malware that probes for the presence of *recctrlu*, we can use the same tools used by malware authors (packers, code injectors etc.) to hide its presence inside existing binaries. Moreover, to prevent malware from running the *recctrl* hypercall itself, we can randomize the magic number it uses at runtime. Finally, we should mention that the APIs provided by PANDA make it possible to identify and react to attempts of malware to fingerprint QEMU. However, this is a topic for future research.

[6] Malrec GitHub repository: https://github.com/moyix/panda-malrec

5.4 Privacy Issues

Sharing datasets of traces captured with PANDAcap could help to boost reproducibility of research results in the area of systems analysis. However, full system traces capured using PANDA or a similar system, will inevitably include any privacy-sensitive information used during recording. This introduces privacy concerns that need to be thought through. Taking a technical approach and sanitizing the trace before sharing is considered infeasible. Sanitization has been investigated in the past for much simpler types of traces without reaching to a generally acceptable solution to the problem.

Since we do not consider not sharing trace datasets as an acceptable option, any privacy issues would need to be addressed with non-technical means. First, users or organizations participating in a study, should be asked for consent to share their data. An additional step would be to share the collected datasets under some form of *non-disclosure agreement* (NDA). This would provide strong guarantees for keeping any identified privacy leaks contained. However, the procedure of signing an NDA is typically a cumbersome process that requires the involvement of legal professionals. For this, we favor more lightweight approaches. For example, each released dataset can come with a well defined retraction procedure for individual samples. This would allow to minimize the damage from any privacy leak, while maintaining the desired flexibility for researchers that use the dataset.

6 CONCLUSION

We presented PANDAcap, a framework that aims to streamline the collection of PANDA full-system traces. Furthermore, we presented an experimental deployment of an *ssh* honeypot with the help of our framework and made publicly available the collected dataset. The source code for PANDAcap as well as links to download the collected dataset can be found on GitHub:

https://github.com/vusec/pandacap

We aspire for PANDAcap to lower the bar for the creation of similar dataset and promote reproducibility of research results in the field of systems and security.

REFERENCES

[1] Sean Bechhofer, Iain Buchan, David De Roure, Paolo Missier, John Ainsworth, Jiten Bhagat, Philip Couch, Don Cruickshank, Mark Delderfield, Ian Dunlop, Matthew Gamble, Danius Michaelides, Stuart Owen, David Newman, Shoaib Sufi, and Carole Goble. 2013. Why linked data is not enough for scientists. *Future Generation Computer Systems* 29, 2 (Feb. 2013), 599–611. https://doi.org/10.1016/j.future.2011.08.004

[2] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of USENIX ATC'05*. Anaheim, CA, USA.

[3] Carl Boettiger. 2015. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review* 49, 1 (2015), 71–79.

[4] Adam Brinckman, Kyle Chard, Niall Gaffney, Mihael Hategan, et al. 2019. Computing environments for reproducibility: Capturing the "Whole Tale". *Future Generation Computer Systems* 94 (2019), 854 – 867. https://doi.org/10.1016/j.future.2017.12.029

[5] Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. 2016. Reprozip: Computational reproducibility with ease. In *Proceedings of the 2016 international conference on management of data*. 2085–2088.

[6] Jim Chow, Tal Garfinkel, and Peter M. Chen. 2008. Decoupling Dynamic Program Analysis from Execution in Virtual Environments. In *Proceedings of USENIX ATC'08*. Boston, MA, USA.

[7] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti. 2018. Understanding Linux Malware. In *2018 IEEE Symposium on Security and Privacy (SP)*. 161–175.

[8] Ewa Deelman, Victoria Stodden, Michela Taufer, and Von Welch. 2019. Initial Thoughts on Cybersecurity And Reproducibility. In *Proceedings of the 2nd International Workshop on Practical Reproducible Evaluation of Computer Systems* (Phoenix, AZ, USA) *(P-RECS '19)*. Association for Computing Machinery, New York, NY, USA, 13–15. https://doi.org/10.1145/3322790.3330593

[9] Brendan Dolan-Gavitt, Josh Hodosh, Patrick Hulin, Tim Leek, and Ryan Whelan. 2014. *Repeatable Reverse Engineering for the Greater Good with PANDA*. Technical Report CUCS-023-14. Columbia University. https://doi.org/10.7916/D8WM1C1P

[10] Brendan Dolan-Gavitt, Josh Hodosh, Patrick Hulin, Tim Leek, and Ryan Whelan. 2015. Repeatable Reverse Engineering with PANDA. In *Proceedings of PPREW'15*. Los Angeles, CA, USA. https://doi.org/10.1145/2843859.2843867

[11] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. 2002. ReVirt: Enabling Intrusion Analysis Through Virtual-machine Logging and Replay. In *Proceedings of USENIX OSDI'02*. Boston, MA, USA. https://doi.org/10.1145/1060289.1060309

[12] Zhenyu Guo, Xi Wang, Jian Tang, Xuezheng Liu, Zhilei Xu, Ming Wu, M. Frans Kaashoek, and Zheng Zhang. 2008. R2: An Application-Level Kernel for Record and Replay. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation* (San Diego, California) *(OSDI'08)*. USENIX Association, USA, 193–208.

[13] Rick Hofstede, Luuk Hendriks, Anna Sperotto, and Aiko Pras. 2014. SSH Compromise Detection Using NetFlow/IPFIX. *SIGCOMM Comput. Commun. Rev.* 44, 5 (Oct. 2014), 20–26. https://doi.org/10.1145/2677046.2677050

[14] Project Jupyter, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, Kyle Kelley, Gladys Nalvarte, Andrew Osheroff, M Pacer, Yuvi Panda, Fernando Perez, Benjamin Ragan-Kelley, and Carol Willing. 2018. Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. In *Proceedings of the 17th Python in Science Conference*. SciPy. https://doi.org/10.25080/majora-4af1f417-011

[15] Farah Zaib Khan, Stian Soiland-Reyes, Richard O Sinnott, Andrew Lonie, Carole Goble, and Michael R Crusoe. 2019. Sharing interoperable workflow provenance: A review of best practices and their practical application in CWLProv. *GigaScience* 8, 11 (Nov. 2019). https://doi.org/10.1093/gigascience/giz095

[16] Christian Knoth and Daniel Nüst. 2017. Reproducibility and practical adoption of geobia with open-source software in docker containers. *Remote Sensing* 9, 3 (2017), 290.

[17] Dirk Merkel. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014, 239, Article 2 (March 2014), 1 pages. https://dl.acm.org/doi/10.5555/2600239.2600241

[18] Mozilla.org. 2014. rr. https://rr-project.org/.

[19] Marcin Nawrocki, Matthias Wählisch, Thomas C. Schmidt, Christian Keil, and Jochen Schönfelder. 2016. A Survey on Honeypot Software and Data Analysis. arXiv:cs.CR/1608.06249

[20] Daniel Nüst and Matthias Hinz. 2019. containerit: Generating Dockerfiles for reproducible research with R. *Journal of Open Source Software* 4, 40 (2019), 1603.

[21] Thomas Pasquier, Matthew K Lau, Ana Trisovic, Emery R Boose, Ben Couturier, Mercè Crosas, Aaron M Ellison, Valerie Gibson, Chris R Jones, and Margo Seltzer. 2017. If these data could talk. *Scientific data* 4 (2017).

[22] João Felipe Pimentel, Juliana Freire, Leonardo Murta, and Vanessa Braganholo. 2019. A Survey on Collecting, Managing, and Analyzing Provenance from Scripts. *ACM Comput. Surv.* 52, 3, Article 47 (June 2019), 38 pages. https://doi.org/10.1145/3311955

[23] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. 2010. Paranoid Android: Versatile Protection for Smartphones. In *Proceedings of ACSAC'10*. Austin, TX, USA. https://doi.org/10.1145/1920261.1920313

[24] Georgios Portokalidis, Asia Slowinska, and Herbert Bos. 2006. Argos: An Emulator for Fingerprinting Zero-day Attacks for Advertised Honeypots with Automatic Signature Generation. In *Proceedings of EuroSys'06*. Leuven, Belgium. https://doi.org/10.1145/1217935.1217938

[25] Giorgio Severi, Tim Leek, and Brendan Dolan-Gavitt. 2018. Malrec: Compact Full-Trace Malware Recording for Retrospective Deep Analysis. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer International Publishing, Cham, 3–23. https://doi.org/10.1007/978-3-319-93411-2_1

[26] Manolis Stamatogiannakis, Elias Athanasopoulos, Herbert Bos, and Paul Groth. 2017. PROV2R: Practical Provenance Analysis of Unstructured Processes. *ACM Trans. Internet Technol.* 17, 4, Article 37 (Aug. 2017), 24 pages. https://doi.org/10.1145/3062176

[27] Victoria Stodden, Marcia McNutt, David H Bailey, Ewa Deelman, Yolanda Gil, Brooks Hanson, Michael A Heroux, John PA Ioannidis, and Michela Taufer. 2016. Enhancing reproducibility for computational methods. *Science* 354, 6317 (2016), 1240–1241.

[28] Erik van der Kouwe, Gernot Heiser, Dennis Andriesse, Herbert Bos, and Cristiano Giuffrida. 2019. SoK: Benchmarking Flaws in Systems Security. In *EuroS&P*. https://download.vusec.net/papers/benchmarking-crimes_eurosp19.pdf