Xanthus: Push-button Orchestration of Host Provenance Data Collection

Xueyuan Han
Harvard University
Cambridge, MA, USA
hanx@g.harvard.edu

James Mickens
Harvard University
Cambridge, MA, USA
mickens@g.harvard.edu

Ashish Gehani
SRI International
Menlo Park, CA, USA
ashish.gehani@sri.com

Margo Seltzer
University of British Columbia
Vancouver, BC, Canada
mseltzer@cs.ubc.ca

Thomas Pasquier
University of Bristol
Bristol, UK
thomas.pasquier@bristol.ac.uk

ABSTRACT

Host-based anomaly detectors generate alarms by inspecting audit logs for suspicious behavior. Unfortunately, evaluating these anomaly detectors is hard. There are few high-quality, publicly-available audit logs, and there are no pre-existing frameworks that enable push-button creation of realistic system traces. To make trace generation easier, we created Xanthus, an automated tool that orchestrates virtual machines to generate realistic audit logs. Using Xanthus' simple management interface, administrators select a base VM image, configure a particular tracing framework to use within that VM, and define post-launch scripts that collect and save trace data. Once data collection is finished, Xanthus creates a self-describing archive, which contains the VM, its configuration parameters, and the collected trace data. We demonstrate that Xanthus hides many of the tedious (yet subtle) orchestration tasks that humans often get wrong; Xanthus avoids mistakes that lead to non-replicable experiments.

CCS CONCEPTS

• **Security and privacy → Usability in security and privacy**; *Intrusion detection systems*; Penetration testing.

KEYWORDS

computer security, data provenance, data replicability

ACM Reference Format:
Xueyuan Han, James Mickens, Ashish Gehani, Margo Seltzer, and Thomas Pasquier. 2020. Xanthus: Push-button Orchestration of Host Provenance Data Collection. In *3rd International Workshop on Practical Reproducible Evaluation of Computer Systems (P-RECS '20), June 23, 2020, Stockholm, Sweden.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3391800.3398175

P-RECS '20, June 23, 2020, Stockholm, Sweden

ACM ISBN 978-1-4503-7977-9/20/06…$15.00
https://doi.org/10.1145/3391800.3398175

1 INTRODUCTION

Host-based intrusion detectors sift through audit data for signs of attack. Training and evaluating such detectors requires trace data. Unfortunately, the security community suffers from a lack of publicly-available, high-quality datasets [49]. For example, DARPA's IDEVAL traces [28] are publicly available but suffer from well-known deficiencies that hurt the realism of the traces [30, 31, 33]. However, academics continue to use these traces [10, 21] (which are over 20 years old!) due to a lack of alternative public datasets.

The rise of provenance-based intrusion detection [16, 17, 19, 20, 34, 50] has emphasized the dearth of realistic, openly-available traces. Data provenance [18] is a particular type of audit data that uses a graph to describe the interaction histories of host objects such as files, processes, and network connections. The typical workflow to evaluate such detection systems consists of three steps: 1) trace benign and attack workloads to construct a training dataset, 2) build a model based on the training traces, and 3) trace new scenarios on which to test the model. In theory, publicly released datasets are the output of the first step.

While evaluating Unicorn, our own provenance-based intrusion detection system (IDS) [16], we repeatedly found that released traces were insufficient for our purposes. For example, DARPA's Transparent Computing dataset [1] contains only attack scenario traces; the StreamSpot dataset [32] was pre-processed, removing key information. These are all symptoms of a fundamental problem: each IDS typically requires a specific kind of trace data, and published traces are specific to the system for which they were originally designed. Unlike conventional machine learning applications, where training data consists of a set of samples and their labels, the "samples" in this case are large, complex, non-standard traces. To address this mismatch we facilitate faithful replication of both the training and test data. In other words, Xanthus enables *replicability* [38] of both training and test workloads for the evaluation of provenance-based IDSes.

Xanthus is a framework for collecting host-based provenance datasets, which automates: (1) configuring a data collection framework, (2) recording data using that framework, and (3) publishing the results. During the configuration stage, Xanthus creates VMs with a deterministic set of initial states defined by user-provided scripts and a specific provenance tracking framework (e.g., SPADE [11] or CamFlow [40, 41]). Xanthus saves these images for repeated

use. Then, in the recording phase, Xanthus runs a specified workload, which can include hooks for additional scripts that control the monitoring infrastructure in real time. When execution completes, Xanthus bundles the data, the Xanthus scripts, and the Xanthus configuration files into a single archive and publishes the archive (e.g., on a configured data repository such as `Dataverse` or on `GitHub`). Other researchers can download the archive to validate correctness of the collected traces, replay the workloads with different auditing systems and experimental settings (e.g., with or without attacks), or replay the saved traces to an analysis tool. For large-scale experiments, it works seamlessly with Amazon Elastic Compute Cloud (EC2).

2 MOTIVATION

Intrusion detection introduces a number of challenges not encountered in other replicability scenarios. Provenance systems interoperate in specific ways with the host operating system, and each attack scenario relies on operating system, library, and application versions. We use our experience trying to evaluate Unicorn using public datasets to motivate Xanthus' key design features.

2.1 Provenance-Based Intrusion Detection

Provenance-based IDSes [16, 34, 50] often perform graph analysis on provenance graphs, in which vertices represent processes, users, or kernel resources (e.g., inodes) and edges represent system-call-induced interactions. However, depending on factors such as security end goals, analysis scope, and runtime performance concerns, detection systems adopt different capture mechanisms and assume distinct graph semantics; those that use the same capture infrastructure might focus on different subsets of data that they deem relevant to their analysis. Meanwhile, security researchers are still developing new graph models [1] describing abstract execution semantics, hoping to facilitate future analysis with better system visibility. Consequently, effectively sharing data and enabling data reuse becomes challenging, which is why Xanthus facilitates replication of the workload that generates the data instead.

Unicorn. Unicorn [16] is a host IDS that uses provenance graphs as input. It leverages the state-of-the-art system-level provenance tracing frameworks [11, 40, 43] to model data flows across an entire system via kernel resources such as inodes and sockets. These frameworks not only interpose on system call invocations, but also understand the *semantics* of system calls. For example, CamFlow [40] can trace how the contents of an incoming network packet flow into a process via `recv()` and out of the process via a subsequent `write()` to a disk file. Unicorn summarizes benign system execution through efficient graph compression to model normal host behavior and defines a similarity metric that quantifies the deviation of the host's current execution from its model. It detects anomalies when the system behaves significantly differently from its norm. We use Unicorn as an example throughout § 3.

2.2 404: Data Not Found

Provenance-based intrusion detection has been studied for a decade. However, we were surprised by the scarcity of publicly-available provenance traces; even compiling and running prior tracing frameworks was challenging.

Provenance Tool	Open Source	Actively Maintained	Tagged Release	Binary Release	Documentation Support	Xanthus Supported
PASS [36]	×	×	×	×	×	×
Story Book [48]	×	×	×	×	×	×
Burrito [13]	✓	×	×	×	×	×
Hi-Fi [43]	✓	×	✓	×	✓	×
LPM [5]	✓	×	×	×	✓	×
SPADE [11]	✓	✓	✓	×	✓	✓
PVM [4] (w/ DTrace [12])	×	✓	×	×	×	✓
CamFlow [40]	✓	✓	✓	✓	✓	✓

Table 1: The provenance tools we examined and selected (below the middle bar) that are supported by Xanthus

2.2.1 DARPA's Transparent Computing Dataset. DARPA's Transparent Computing program sponsors a wide variety of provenance research. A primary goal is to use provenance to detect and analyze advanced persistent threats (APT), i.e., attacks that spread their activity across a long period of time, hiding malicious behavior amid normal system events. DARPA conducted simulated attacks on realistic servers to generate public datasets for researchers. For example, in 2018, DARPA ran a two-week-long engagement [1] in which red teams launched APT attacks on victim machines running five different provenance tracking frameworks. Prior to the engagement, DARPA deployed scripts to generate innocuous background activity (e.g., simulated user logins to ssh daemons). Although the collected provenance traces are publicly accessible, DARPA did not release the data captured from the innocuous background activity, which makes it difficult to evaluate anomaly-based intrusion detectors, since anomalies are defined relative to normal behavior. We petitioned DARPA for details about the background activity but were unable to obtain the scripts that generated the activity.

2.2.2 StreamSpot's Dataset. StreamSpot [32] is an academic project that introduced a fast streaming analysis on provenance graphs. The authors made their evaluation dataset public. However, like the DARPA dataset, it lacks a description of non-anomalous behavior. The dataset is also pre-processed: it contains only the provenance information useful to StreamSpot's algorithm. Hiding raw trace information diminishes the value of a dataset, since different analytics systems might examine different kinds of provenance states.

2.2.3 Other Datasets. We surveyed other academic frameworks for tracking and analyzing provenance [19, 20, 22, 29, 47], but none were accompanied with public datasets. The associated papers did make sincere attempts to describe attack scenarios evaluated by the authors; however, our attempts to replicate even well-described attacks were time-consuming, labor-intensive, and often ended in failure. For example, Jiang et al. [22] used a virtualization environment called vGround [23] to isolate worms in a realistic but confined environment. Unfortunately, neither vGround nor the experimental setup scripts are publicly available.

2.3 An Ideal Framework

We struggled to locate a high-quality, public dataset to evaluate Unicorn, while our subsequent manual efforts to create our own datasets were equally frustrating. Often times, we were unable to repeat the same experiment using a different tracing framework due to, e.g., unexpected environmental changes, missing packages that existed in prior runs, or even lost references to the experiment due to our own carelessness. Based on our experience, we designed Xanthus with the following properties in mind:

- **Replicability**: The framework must collect enough information to allow a third party to recreate an experiment so that different graph semantic models can be adopted to describe identical system execution (§ 2.1). For example, it must capture the discrete events or generative models associated with both malicious behavior and innocuous background activity. It also needs to capture environmental features such as version information for the operating system and user-level binaries that were running during an experiment.
- **Flexibility**: The framework should not make assumptions about the downstream data consumers. When possible, it should emit raw, unprocessed data. Storage is cheap; thus, it should err on the side of collecting too much data, not too little.
- **Longevity**: The framework must collect and publish data in a way that is not dependent on a particular hosting server or distribution technology. An ideal dataset is self-hosting in the sense that, once a researcher has downloaded the bytes in the dataset, minimal additional infrastructure should be necessary to analyze the data or recreate the experiments that generated the data.
- **Usability**: The framework should provide explicit interfaces that allow easy scripting to generate host behavior, collect trace data, and so on. To the greatest extent possible, configuring the software inside the system to trace should be automated. Creating a self-hosting archive should also be automated.
- **Shareability**: Researchers should be able to exchange entire experimental environments. Shareability is enabled by flexibility, longevity, and usability.

3 XANTHUS FRAMEWORK

Xanthus assumes that the downstream analytics system requires as input host audit data, but it is agnostic to the specific tracing system used. Currently, we focus on capturing system-level provenance data (for Unicorn). Table 1 outlines a set of criteria we used to compare and select provenance tools supported by Xanthus.

Xanthus is written in Ruby and can be easily installed through Ruby's package manager RubyGems. Fig. 1 shows the three high-level stages that comprise Xanthus. In the remainder of this section, we elaborate on each stage with simple code snippets to demonstrate concepts and design decisions.

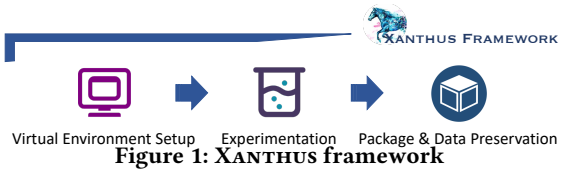

Figure 1: Xanthus framework

3.1 Virtual Environment Setup

It is tempting to believe that the script that executes an experiment is a long-lived artifact. While a script may provide detailed specifications about a particular environment, software versions used, and instructions that automate the experimental setup, its correct execution depends on the availability of those artifacts. If some version becomes unavailable, replication becomes impossible.

One solution is to provide virtual machine (VM) images encapsulating the correct environment and software dependencies. Those materialized images enable immediate replication of an identical working environment.

```
1  config.vm :server do |vm|
2      vm.box = 'ubuntu/trusty64'
3      vm.ip = '192.168.33.3'
4      vm.boxing = [:box_config]
5  end
6  config.script :box_config do
7      %q{%{
8        sudo apt-get install vim apache2
9        gem install json rgl mqtt rake bundler
10     }}
11 end
```

Listing 1: Set up a user-configured VM in Xanthus

Xanthus leverages `Vagrant` to manage VMs. Before running an experiment, it creates the necessary VM image(s), which can be stored locally or shared on VagrantCloud [3], an online box repository where users share public boxes. Xanthus also supports pre-existing images hosted on VagrantCloud, which can be further customized through scripts. For example, in Listing 1, we use the `ubuntu/trusty64` image and customize it with the `box_config` script. `vm.ip` defines the virtual IP address used during the experiment. Xanthus boxes the VM once in the first run and uses the materialized VM afterwards; this also provides a more efficient out-of-box experience for those wishing to use the artifact since they do not have to configure the machine for each experiment. Users can upload the resulting Vagrant box to VagrantCloud.

```
1  config.vm :aws do |vm|
2    vm.on_aws = true
3    vm.aws_env_key_id = 'AWS_ACCESS_KEY_ID'
4    vm.aws_env_key_secret = 'AWS_SECRET'
5    vm.aws_key_pair_name = 'AWS_KEY_PAIR'
6    vm.aws_region = 'eu-west-2'
7    vm.aws_ami = 'ami-xxxxxxxxxxxx'
8    vm.aws_instance_type = 'i3.large'
9    vm.aws_security_group = 'AWS_SG_1'
10 end
```

Listing 2: Set up VMs on AWS

To enable large-scale, multi-host experiments, Xanthus works seamlessly on Amazon EC2 (Listing 2); users simply need to switch to the AWS mode (Line 2) and provide their EC2 credentials in the configuration file to set up VMs in the cloud.

3.2 Specifying an Experiment

Each experiment is called a *job*, which consists of instantiation(s) of VM image(s), execution of user-defined tasks assigned to particular instances, and management of outputs (e.g., to retrieve audit logs). A Xanthus workflow is composed of one or more jobs that can be executed multiple times.

```
1  config.job :attack do |job|
2      job.iterations = 2
3      job.tasks = {server: [:server], client: [:pre, :camflow_start,
         :attack, :camflow_stop, :post]} # VM tasks
4      job.inputs = {server: ['ipscan_3.5.5_amd64.deb', 'exploit.py'
       ]} # VM inputs
5      job.outputs = {client: {config: '/etc/camflow.ini', trace: '/
       tmp/audit.log'}} # VM outputs
6  end
```

Listing 3: Configure a job in Xanthus

Listing 3 is an example of a job configuration, in which a job called `attack` is configured to run twice. During each iteration, two VMs, `server` and `client`, are instantiated and run their respective task(s). In Line 3, `server` runs a single task, defined in

`config.script :server` (similar syntax as in Listing 5), while `client` has multiple tasks that run sequentially. A Xanthus *task* allows users to logically encapsulate a single step in the experiment. Line 4 defines two inputs to `server`, a Debian package and a Python script, while Line 5 shows that we expect two outputs from `client`, a configuration file and trace data.

Xanthus is a framework for cybersecurity experiments, so it is important to ensure easy integration with popular security tools. We show how users can readily use Xanthus to retrieve traces during penetration testing using Metasploit, Armitage, and Cortana. Without Xanthus, a researcher would have to manually 1) set up an attacker and a victim machine, 2) log onto the attacker machine to configure Metasploit and Armitage, 3) log onto the victim machine to configure its audit system, 4) execute the attack, and 5) extract audit data from the victim machine.

Security Example. Metasploit [25] is a well-known penetration testing framework that helps security experts verify vulnerabilities, manage security assessments, and improve security awareness. Armitage [25] is a scriptable cyberattack management tool for Metasploit and a force multiplier (i.e., creates synergy) for red team operations. Cortana is the scripting language behind Armitage that automates the Metasploit framework and creates long running bots.

```
config.vm :victim do |vm|
    vm.box = 'e314c/Metasploitable2'
    vm.version = '0.0.1'
    vm.ip = '192.168.33.8'
end
config.vm :attacker do |vm|
    vm.box = 'Sliim/kali-2018.1-amd64'
    vm.version = '1'
    vm.ip = '192.168.33.10'
end
```

Listing 4: Configure attacker and victim VMs

In Listing 4, we configure an intentionally vulnerable version of Ubuntu Linux VM called Metasploitable [35]. The Metasploitable VM is designed specifically for testing security tools and demonstrating vulnerabilities. We then configure a Kali Linux machine, a security-oriented Linux distribution that pre-installs many useful penetration testing tools, including Metasploit and Armitage. As we are using existing images from the VagrantCloud, setting them up is trivial, as illustrated in the listing.

```
config.job :cortana do |job|
    job.iterations = 1
    job.tasks = {victim: [:actions], attacker: [:attack]}
    job.inputs = {attacker: ['local.prop', 'demo.cna']}
end
config.script :attack do
    %q{%{
      teamserver 192.168.33.10 password &
      java -jar cortana.jar local.prop demo.cna
    }}
end
```

Listing 5: Configure the adversarial scenario

Now, let's assume that we wish to simulate an adversarial scenario where the attacker exploits the FTP vulnerability in Metasploitable and uses the `vsftpd_234_backdoor` module in Metasploit to install a backdoor and create a reverse shell payload to remotely control Metasploitable. Listing 5 describes the experiment in Xanthus. The attacker consists of a single task, `attack` that launches the attack with a Cortana script. To run Cortana as a stand-alone script, the attacker needs to set up an Armitage `teamserver` locally on the VM. The user then specifies properties of the team server by placing them in the file `local.prop`. The file `demo.cna` is the Cortana script that runs the attack (Listing 6). It creates a virtual Metasploit console that prepares the exploit and configures the payload (e.g., setting up the remote host IP address through `RHOST`). To show that our attack succeeds, the script registers two listeners, one for when a reverse shell session is open and one for when the shell responds to the `whoami` command. When the `session_open` event triggers the listener, the attacker automatically sends a `whoami` command to the victim and prints victim's response on its console.

```
on shell_whoami {
  println("[ $+ $1 $+ ] I am: $3");
}
on session_open {
  if (!-isshell $1)
    return;
  s_cmd($1, "whoami");
}
$console = console();
cmd($console,"use exploit/unix/ftp/vsftpd_234_backdoor");
cmd_set($console, %(RHOST => "192.168.33.8", RPORT => "21", TARGET
      => "0",PAYLOAD => "cmd/unix/interact"));
cmd($console, "exploit -j");
```

Listing 6: The Cortana script

Xanthus allows a researcher to easily run similar experiments multiple times with different capture mechanisms and share precise configurations with others. Xanthus' modularized design allows researchers to reuse their experimental setup, simply changing e.g., Metasploit's exploit module to create new experiments.

3.3 Package and Data Preservation

Xanthus enables push-button execution of the framework. The artifacts of the workflow, including user-supplied scripts and packages (as defined in `job.inputs`) and experimental results and datasets (as defined in `job.outputs`), are all bundled and archived locally.

```
config.github do |github|
  github.repo = '<ADD GITHUB REPO>'
  github.token = '<ADD GITHUB TOKEN>'
end
config.dataverse do |dataverse|
  dataverse.server = '<ADD DATAVERSE BASE URL>'
  dataverse.repo = '<ADD DATAVERSE NAME>'
  dataverse.token = '<ADD DATAVERSE TOKEN>'
  dataverse.subject = '<ADD DATAVERSE SUBJECT>'
end
```

Listing 7: Configure GitHub and Dataverse

Xanthus allows users to automatically share the collected experimental data. For example, if the user provides a GitHub repository address and an access token, it pushes the archive to GitHub automatically using Git Large File Storage (Listing 7). Xanthus also supports automatic sharing via Dataverse [26], and we are working on providing more archiving options. We have made an example archive available at https://github.com/margoseltzer/wget-apt. The archive contains a `.xanthus` file for push-button replicability. The `.xanthus` file is the central orchestration file that controls the entire pipeline described in this section. It contains metadata describing the experiments and actionable instructions to 1) generate VM images, 2) schedule tasks, 3) setup experiments, and 4) store and upload data.

4 RELATED WORK

We are not the first to observe that cybersecurity research is threatened by a lack of high-quality, easily-accessible datasets. For example, Ghorbani et al. [45] evaluated 11 publicly-available traces used by intrusion detection researchers and concluded that none of the traces were comprehensive or reliable. Ghorbani et al. introduced their own dataset (CICIDS2017) that leveraged their prior work on systematic generation of IDS traces [46]. However, the collection of the CICIDS2017 trace was manually orchestrated (and thus non-replicable).

Despite the power of host-based intrusion detection, the security community has traditionally paid more attention to network traces than host ones. This bias may reflect the fact that host-based IDSes are more recent inventions. DARPA IDEVAL is a well-known host trace, but it has various deficiencies, such as poor diversity of executed programs [30]. We know of only one more host-based dataset that is widely used—the University of New Mexico dataset [2]. However, this dataset suffers from similar problems that hurt the realism of the trace [42]. Other publicly-available host traces are either application-specific [37] or suffer from low attack diversity and coarse-grained trace information [7]; these datasets are studied by only a few papers [14, 15]. Due to the lack of high-quality datasets, many evaluations of host-based IDSes use private datasets [6, 27, 47] or a mixture of public and private datasets [8, 30].

Prior critiques of network traces are equally applicable to host traces. For example, several papers bemoan how a lack of documentation prevents replicable generation of traces [39, 44, 49]. Deelman et al. [9] discuss how best-practices in cybersecurity, e.g., applying patches to address vulnerabilities, can change system functionality in ways that might affect replicability, e.g., by changing the code paths in the kernel that execute. From their discussion, we can conclude that a replicable experiment must record not only the software that was used, but also the set of patches and updates that were applied to that software. Xanthus neatly sidesteps these problems by implicitly recording them by packaging the entire environment into a virtual machine.

Other practical frameworks [24] exist, but these systems focus on re-running a computation to produce an identical output. Xanthus' power lies in replicating a computation (i.e., the training and test workloads) specifically *not* to produce an identical output, but to produce a different trace of the same computation. To the best of our knowledge, Xanthus is the first general framework that enables replication of workloads that interact in complex ways with host operating systems. While we focus here on its use for evaluating IDSes, Xanthus can also be used to replicate results from experimental computer systems.

5 DISCUSSION

Our Xanthus prototype is fully functional, and we have already used it to evaluate Unicorn. For example, we used Xanthus to generate an APT trace (§ 2.2.1). The traced APT attack exploited a `wget` vulnerability (CVE-2016-4971) to install a corrupt Debian package; once installed, the package contacted a command-and-control server, and slowly exfiltrated data. We were able to evaluate Unicorn's operation with three different provenance collection infrastructures by changing only a few lines of code in the configuration script. Xanthus achieved the desired goals from Section 2.3:

- **Replicability:** Xanthus archives all of the information needed to recreate a trace. For example, Xanthus records the environmental scripts provided by the user, and the contents of the corrupt Debian package. The Vagrant boxes that Xanthus outputs are sufficient for a third party to replicate the original tracing conditions.
- **Flexibility:** In Xanthus, the selection of an audit framework is orthogonal to the selection of the environmental conditions that drive the system behavior in the trace. This flexibility makes it easy to generate multiple datasets that use different logging mechanisms to observe the same environmental setup. This is the feature that allowed us to use different provenance systems in our evaluation.
- **Longevity**: Currently, VMs are considered best practice for long-term digital preservation as the only requirement for running them is a compatible hypervisor. Xanthus captures all necessary information inside a VM.
- **Usability:** Xanthus' script-based interface encourages the design of incremental, modular experiments. For example, as shown in § 3.1, Xanthus scripts enable a user to directly configure a VM and its applications. Xanthus also directly integrates with popular penetration testing tools such as Metasploit (§ 3.2), allowing off-the-shelf attacks to easily be added to a trace. Xanthus re-runs an experiment using a single command.
- **Shareability:** Xanthus automatically pushes VM images to VagrantCloud and the rest of a dataset archive to GitHub.

The interested reader can find our APT dataset at https://github.com/tfjmp/xanthus.

Xanthus improves dataset replicability, but does not automatically improve dataset *realism*. Xanthus users are responsible for ensuring that environmental scripts and VM configurations reflect plausible real-life scenarios. Prior work on dataset fidelity [33, 45] can help Xanthus users to create high-quality traces.

6 CONCLUSION

Xanthus is a practical tool for generating and sharing provenance traces. By automating the minutiae of trace collection and bundling, Xanthus enables the replicable evaluation of host-based intrusion detectors.

AVAILABILITY

Xanthus is an open-source project, available at https://github.com/tfjmp/xanthus under an MIT license. Xanthus's Ruby `gem` is freely distributed at https://rubygems.org/gems/xanthus. The CamFlow provenance capture system is available at http://camflow.org (GPL v2 license). The SPADE provenance capture system is available at https://github.com/ashish-gehani/SPADE (GPL v3 license).

ACKNOWLEDGMENTS

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG). This material is based upon work supported by the National Science Foundation under Grants ACI-1440800, ACI-1450277 and ACI-1547467. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Transparent computing engagement 3 data release, accessed May 12, 2020. https://github.com/darpa-i2o/Transparent-Computing.

[2] University of New Mexico system call dataset, accessed May 12, 2020. https://www.cs.unm.edu/~immsec/systemcalls.html.

[3] VagrantCloud, accessed May 12, 2020. https://app.vagrantup.com/boxes/search.

[4] Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, and Andy Hopper. Opus: A lightweight system for observational provenance in user space. In *Workshop on the Theory and Practice of Provenance*. USENIX, 2013.

[5] Adam Bates, Dave Jing Tian, Kevin RB Butler, and Thomas Moyer. Trustworthy whole-system provenance for the linux kernel. In *Security Symposium*, pages 319–334. USENIX, 2015.

[6] Suresh N Chari and Pau-Chen Cheng. Bluebox: A policy-driven, host-based intrusion detection system. *Transactions on Information and System Security*, 6(2):173–200, 2003.

[7] Gideon Creech and Jiankun Hu. Generation of a new ids test dataset: Time to retire the kdd collection. In *Wireless Communications and Networking Conference (WCNC)*, pages 4487–4492. IEEE, 2013.

[8] Gideon Creech and Jiankun Hu. A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns. *Transactions on Computers*, 63(4):807–819, 2014.

[9] Ewa Deelman, Victoria Stodden, Michela Taufer, and Von Welch. Initial thoughts on cybersecurity and reproducibility. In *International Workshop on Practical Reproducible Evaluation of Computer Systems*, pages 13–15, 2019.

[10] Hussein M Elshafie, Tarek M Mahmoud, and Abdelmgeid A Ali. Improving the performance of the snort intrusion detection using clonal selection. In *International Conference on Innovative Trends in Computer Engineering (ITCE)*, pages 104–110. IEEE, 2019.

[11] Ashish Gehani and Dawood Tariq. Spade: Support for provenance auditing in distributed environments. In *International Middleware Conference*, pages 101–120. ACM/IFIP/USENIX, 2012.

[12] Brendan Gregg and Jim Mauro. *DTrace: Dynamic tracing in Oracle Solaris, Mac OS X, and FreeBSD*. Prentice Hall Professional, 2011.

[13] Philip J Guo and Margo Seltzer. Burrito: Wrapping your lab notebook in computational infrastructure. In *Workshop on the Theory and Practice of Provenance*. USENIX, 2012.

[14] Waqas Haider, Gideon Creech, Yi Xie, and Jiankun Hu. Windows based data sets for evaluation of robustness of host based intrusion detection systems (ids) to zero-day and stealth attacks. *Future Internet*, 8(3):29, 2016.

[15] Waqas Haider, Jiankun Hu, Jill Slay, Benjamin P Turnbull, and Yi Xie. Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network and Computer Applications*, 87:185–192, 2017.

[16] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *Symposium on Network and Distributed System Security (NDSS)*, 2020.

[17] Xueyuan Han, Thomas Pasquier, Tanvi Ranjan, Mark Goldstein, and Margo Seltzer. Frappuccino: Fault-detection through runtime analysis of provenance. In *Workshop on Hot Topics in Cloud Computing (HotCloud)*. USENIX, 2017.

[18] Xueyuan Han, Thomas Pasquier, and Margo Seltzer. Provenance-based intrusion detection: opportunities and challenges. In *Workshop on the Theory and Practice of Provenance*. USENIX, 2018.

[19] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *Symposium on Network and Distributed System Security (NDSS)*, 2019.

[20] Wajih Ul Hassan, Mark Lemay, Nuraini Aguse, Adam Bates, and Thomas Moyer. Towards scalable cluster auditing through grammatical inference over provenance graphs. In *Symposium on Network and Distributed System Security (NDSS)*, 2018.

[21] Poulmanogo Illy, Georges Kaddoum, Christian Miranda Moreira, Kuljeet Kaur, and Sahil Garg. Securing fog-to-things environment using intrusion detection system based on ensemble learning. In *Wireless Communications and Networking Conference (WCNC)*, pages 1–7. IEEE, 2019.

[22] Xuxian Jiang, AAron Walters, Dongyan Xu, Eugene H Spafford, Florian Buchholz, and Yi-Min Wang. Provenance-aware tracing of worm break-in and contaminations: A process coloring approach. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 38–38. IEEE, 2006.

[23] Xuxian Jiang, Dongyan Xu, Helen J Wang, and Eugene H Spafford. Virtual playgrounds for worm behavior investigation. In *International Workshop on Recent Advances in Intrusion Detection*, pages 1–21. Springer, 2005.

[24] Ivo Jimenez, Michael Sevilla, Noah Watkins, Carlos Maltzahn, Jay Lofstead, Kathryn Mohror, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. The popper convention: Making reproducible systems evaluation practical. In *Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1561–1570. IEEE, 2017.

[25] David Kennedy, Jim O'gorman, Devon Kearns, and Mati Aharoni. *Metasploit: The penetration tester's guide*. No Starch Press, 2011.

[26] Gary King. An introduction to the dataverse network as an infrastructure for data sharing, 2007.

[27] Peter Lichodzijewski, A Nur Zincir-Heywood, and Malcolm I Heywood. Host-based intrusion detection using self-organizing maps. In *International Joint Conference on Neural Networks*, volume 2, pages 1714–1719. IEEE, 2002.

[28] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, 2000.

[29] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. Towards a timely causality analysis for enterprise security. In *Symposium on Network and Distributed System Security (NDSS)*, 2018.

[30] Federico Maggi, Matteo Matteucci, and Stefano Zanero. Detecting intrusions through system call sequence and argument analysis. *Transactions on Dependable and Secure Computing*, 7(4):381–395, 2010.

[31] Matthew V Mahoney and Philip K Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 220–237. Springer, 2003.

[32] Emaad Manzoor, Sadegh M Milajerdi, and Leman Akoglu. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1035–1044. ACM, 2016.

[33] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, 2000.

[34] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, R Sekar, and VN Venkatakrishnan. Holmes: real-time apt detection through correlation of suspicious information flows. In *Symposium on Security and Privacy (SP)*, pages 1137–1152. IEEE, 2019.

[35] HD Moore. Metasploitable 2 exploitability guide. *Retrieved June*, 27:2013, 2012.

[36] Kiran-Kumar Muniswamy-Reddy, David A Holland, Uri Braun, and Margo I Seltzer. Provenance-aware storage systems. In *Annual Technical Conference*, pages 43–56. USENIX, 2006.

[37] Syed Shariyar Murtaza, Wael Khreich, Abdelwahab Hamou-Lhadj, and Mario Couture. A host-based anomaly detection approach by representing system calls as states of kernel modules. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 431–440. IEEE, 2013.

[38] National Academies of Sciences, Engineering, and Medicine et al. *Reproducibility and replicability in science*. National Academies Press, 2019.

[39] Joshua Ojo Nehinbe. A critical evaluation of datasets for investigating idss and ipss researches. In *International Conference on Cybernetic Intelligent Systems (CIS)*, pages 92–97. IEEE, 2011.

[40] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. Practical whole-system provenance capture. In *Symposium on Cloud Computing*, pages 405–418. ACM, 2017.

[41] Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David Eyers, Jean Bacon, and Margo Seltzer. Runtime analysis of whole-system provenance. In *Conference on Computer and Communications Security (CCS)*. ACM, 2018.

[42] Marcus Pendleton and Shouhuai Xu. A dataset generator for next generation system call host intrusion detection systems. In *Military Communications Conference (MILCOM)*, pages 231–236. IEEE, 2017.

[43] Devin J Pohly, Stephen McLaughlin, Patrick McDaniel, and Kevin Butler. Hi-Fi: Collecting high-fidelity whole-system provenance. In *Annual Computer Security Applications Conference*, pages 259–268. ACM, 2012.

[44] Haakon Ringberg, Matthew Roughan, and Jennifer Rexford. The need for simulation in evaluating anomaly detectors. *SIGCOMM Computer Communication Review*, 38(1):55–59, 2008.

[45] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *International Conference on Information Systems Security and Privacy (ICISSP)*, pages 108–116, 2018.

[46] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, 2012.

[47] Xiaokui Shu, Danfeng Daphne Yao, Naren Ramakrishnan, and Trent Jaeger. Long-span program behavior modeling and attack detection. *Transactions on Privacy and Security (TOPS)*, 20(4):12, 2017.

[48] Richard P Spillane, Russell Sears, Chaitanya Yalamanchili, Sachin Gaikwad, Manjunath Chinni, and Erez Zadok. Story book: An efficient extensible provenance framework. In *Workshop on the Theory and Practice of Provenance*. USENIX, 2009.

[49] Mahbod Tavallaee, Natalia Stakhanova, and Ali Akbar Ghorbani. Toward credible evaluation of anomaly-based intrusion-detection methods. *Transactions on Systems, Man, and Cybernetics*, 40(5):516–524, 2010.

[50] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, C Gunter, et al. You are what you do: Hunting stealthy malware via data provenance analysis. In *Symposium on Network and Distributed System Security (NDSS)*, 2020.