Evaluating system-level provenance tools for practical use

Samuel Grayson, Reed Milewicz

August 25, 2023

1 Introduction

The Oxford English Dictionary defines **provenance** as "a record of the ultimate origin and passage of an item through its previousowners." In a scientific context, the origin of an artifact is some experimental procedure, so provenance is a description of that; each input used in the procedure has its own provenance, which might be included in the final product, depending on the depth requested. **Computational provenance** refers to the software programs and input data used to generate the artifact can serve as a description of the experimental procedure [14]. This provenance is either **prospective**, which describes the computational procedure one would need to take to generate an analogous artifact, or **retrospective**, which describes the computational procedure that the authors took to generate the artifact [55]. Somewhat independently from the retrospective/prospective classification, provenance can be collected at the application-level, workflow-level, or system-level [38, 14].

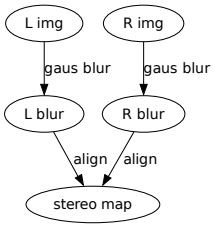
- To collect **application-level provenance**, one would modify each application to emit provenace data. This is the most semantically rich but least general, as it only enables collection by that particular modified applicaiton [38].
- To collect **workflow-level provenance**, one would modify the workflow engine, and all workflows written for that engine would emit provenance data. Workflow engines are only aware of the dataflow not higher-level semantics, so workflow-level provenance is not as semantically rich as application-level provenance. However, it is more general than application-level provenance, as it enables collection in any workflow written for that modified engine [14].
- To collect **system-level provenance**, one uses operating system facilities to report the inputs and outpus that a process makes. This is the least semantically aware because it does not even know dataflow, just a history of inputs and outputs, but it is the most general, because it supports any process (including any application or workflow engine) that uses watchable I/O operations [14].

The workflow-level provenance graph (Figure 1b) knows what data goes where, but not what the data represents, so it uses arbitrary labels A, B, C, D for data and X, Y for programs.

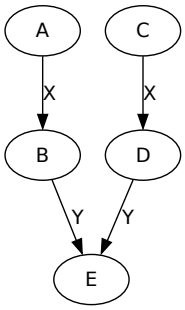
If X and Y are called as functions from within one process, system-level provenance would see Figure 1c. D does not really depend on A, but the system-level graph (Figure 1d) would have no way of knowing that from just the input/output log. The system-level graph also does not know that E only depends on the information in A and C which is also present in B and D. Most applications (e.g., reusing cached results) would prefer false positives than false negatives to the question, "does this depend on that?", so we conservatively assume an input effects any future output. The system-levle graph also does not know the any of the transformations (e.g., from A to B). However, if X and Y are called as subprocesses, the system-level graph may be closer to the workflow-level provenance graph.

Computational provenance is useful for anumber of applications:

- While provenance cannot guarantee reproducibility, it can serve as a starting point for one seeking to reproduce a computational artifact.
- It helps scientists know when their outputs are out-of-date or not (reusing cached results).



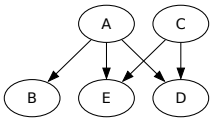
(a) Application-level provenance



(b) Workflow-level provenance

```
read A
write B
read C
write D
write E
```

(c) System-level input/output log



(d) System-level provenance

Figure 1: Several provenance graphs collceted at different levles.

- Retrospective provenance could simplify the task of producing artifact descriptions that are often required when submitting to conferences.
- It helps scientists debug differences in outputs by viewing differences in their methods.
- It helps administrators know where data is being used in a complex system.

There are many possible applications for provenance. Moreover, users collect system-level retrospective provenance (henceforth, **SLRP**) quite easily, reaping these benefits without having to modify the code. They would just need to run their experiment in a system that records the inputs and outputs of its child processes. However, there is no SLRP tool used in common practice. We believe this is because they have potentially large performance overheads and are difficult to use. This study aims to quantify the performance overhead and report user-experiences with SLRP tools. Based on these two properties, we will determine the suitability of current SLRP tools for computational work at Sandia National Laboratories.

We hope this study, prior work, and future work will eventually help SLRP tools over the wall between research and practice at Sandia and elsewhere.

1.1 Prior work

There are many provenance tools and primary studies that we will find in Section 3.1. However, these studies do not use the same benchmarks between themselves. We also want to provide additional evaluation of "ease-of-use" and appropriateness for our specific use-case, not a generic benchmark.

Like this study, ProvMark [10] seeks to evaluate SLRP tools. However, ProvMark evaluates the *completeness* of each tool not performance. As performance is not their goal, ProvMark does not use a realistic application as the benchmark.

Interested readers should consult Freire's survey [14] to learn the conceptual design space for computational provenance, including retrospective/prospective, capture mechanisms, provenance models, storage formats, and query languages.

2 Methodology

We began a rapid review to identify the research state-of-the-art tools for automatic SLRP.

2.1 Rapid Review

Rapid Reviews are a lighter weight alternative to systematic literature reviews with a focus on timely feedback for decision-making. Several studies show [**todo**] that Rapid Reviews can yield substantially similar results to a systematic literature review, albeit less detailed. Several other studies [**todo**] show that Rapid Reviews, although developed in the field of medicine, are useful to inform software engineering design decisions.

We conducted a rapid review with the following parameters:

- **Objective**: identify system-level retrospective provenance collection tools.
- **Search terms**: "system-level" AND "provenance". Note that "retrospective" is less used in prior literature; system-level provenance is usually retrospective.
- **Search engine**: Google Scholar
- **Number of results**: 50 (in order to complete the review in a timely manner). This threshold is the point of diminishing returns as no new toosl came up in the results numbered 40 – 50.

2.2 Performance analysis

We will run a set of real applications with and without SLRP collection and measure their overhead. SLRP has to intercept the I/O operation and log them; interception has to happen on the critical path, but logging can happen on the critical path or in a separate process[1]. We follow the "EMP" experimental procedure [48] which dictates deactivating non-essential daemons and activating the NTP daemon (to correct realtime clock drift). We run our experiment within BenchExec [5], which creates cgroup with a fixed CPU allocation and pins each process to a single CPU core. We use Kalibera's algorithm [24] to determine the number of executions and derive a confidence interval of our results. We will measure maximum resident set size, CPU time of the original and additional processes, wall time, and storage size of outputs.

For our benchmark applications, we use typical workloads that a data analysts would actually run:

- compiling a complex package from source
- running a data science Jupyter notebook

3 Results

First, we oultine the result of our rapid review. Second, we filter the candidates to those which are feasible to test on our system and suitable for our use-case at Sandia. Finally, we run a performance analysis on them.

3.1 Candidates from Rapid Review

Several other tools came up, especially workflow-level provenance implementations, but we restrict this report to SLRP tools. Of the search results, we list the primary and secondary studies which came up in Tables 1 and 2. The union of these leads us to large set of SLRP tools Table 3, for which we found the original publication and characterized the collection method.

[1]For the purposes of this paper, we will call POSIX "threads" as a kind of process which shares certain resources (virtual memory map, file descriptor table signal handler table). They are both created by the `clone(2)` syscall in Linux.

[2]SPADE can use multiple backends, including other provenance collectors. On Linux, SPADE can use: Auditd, CamFlow, FUSE; On MacOS: OpenBSM, MacFUSE, Fuse4x; On Windows: ProcessMonitor; On any platform: import static data (e.g., from logs on disk), applications instrumented with API, applications compiled with LLVM pass. Given our requiremnts, SPADE+auditd and SPADE+FUSE are the only suitable SPADE configurations.

Table 1: Secondary studies in our search results

Study	SLRP Tools mentioned	Summary
Li et al. [32]	SPADE, CamFlow, BEEP, Ma et al. [36], MPI, ProTracer, UIScope, Winnower, LDX, MCI, RAIN, RTAG, libdft, OmegaLog	To survey SLRP tools for threat-dection
Berrada et al. [4]	SPADE, OPUS	To establish baselines results for anomoly detection algorithms
Braun et al. [7]	PASS, GenePattern, TREC, Lineage FS	To summarize issues in automatic provenance collection
Chen et al. [9]	SPADE on auditd, OPUS, PASS, Hi-Fi, LPM	To compare the expressiveness of OPUS and SPADE
Carat et al. [8]	PASS, PASSv2, SPADE	To survey provenance collection and usage
Zipperle et al. [56]	ETW, Lineage FS, DTrace, PASS, PASSv2, Panorama, SPADE, Hi-Fi, BEEP, LogGC, Sysmon, LPM, Provmon, DataTracker, PROV-Tracer, INSPECTOR, ProTracer, RecProv, MPI, RAIN, CamFlow, LPROV	To survey provenance-based intrusion-detection systems
Lee et al. [30]	PASS, SPADE	To survey secure provenance collection in the cloud

Table 2: Primary studies in our search results

Study	Tools introduced
Tariq et al. [50]	SPADE on LLVM pass
Sultana et al. [49]	FiPS
Muniswamy-Reddy et al. 2009 [38]	PASSv2
Muniswamy-Reddy et al. 2006 [39]	PASS
Gehani et al. [17]	SPADE
Pohly et al. [41]	Hi-Fi
Pasquier et al. [40]	CamFlow
Rupprecht et al. [43]	Ursprung
Fadolakarim et al. [13]	PANDDE
Wang et al. [52]	LPROV

Table 3: SLRP Tools mentioned in primary and secondary studies in our search results. "ins." stands for instrumentation

Tool	Suitable? Sec. 3.2	Collection method	Collection tool	Notes
SPADE [17]	Yes	Audit, filesystem, compile-time, **or** source ins.	Multiple[2]	Can use multiple low-level sources
OPUS [2]	Yes	Library instrumentation	libc instrumentaiton	
OmegaLog [19]	No source code	Static binary ins., audit	Angr, auditd	Finds relationships in logs in mulitple layers
FiPS (File Provenance System) [49]	No source code	Filesystem ins.	VFS	
LogGC	No source code	Audit	Auditd	The contribution is deleting unnecessary parts of the log
RecProv [21]	No source code	Tracing	rr, ptrace	
PANDDE [13]	kernel modif.	Filesystem ins.	Custom VFS	
Winnower [20]	Wrong platform	Audit	auditd, SELinux, SPADE, Docker Swarm	Specific to Docker Swarm
URSpring	Wrong platform	Audit and file-system ins.	auditd and IBM Spectrum Scale	Specific to IBM Spectrum Scale FS
Event Tracer for Windows [12]	Wrong platform	Audit	NT Kernel	Implemented for Windows
UIScope [53]	Wrong platform	Audit	ETW and accessibility service	Tracks grpahical user interaction
TREC [51]	Wrong platform	Audit	Proc filesystem	Implemented for Solaris
DTrace [1]	Wrong platform	Audit	Respective kernels	Can do event processing in kernel-space
Sysmon [37]	Wrong platform	Audit	NT Kernel	Implemented for Windows
BEEP (Binary-based Execution Partition) [31]	Hardware-specific	Dynamic and static binary ins.	Intel Pin and, PEBIL	
libdft [25]	Hardware-specific	Dynamic binary ins.	Intel Pin	
RAIN (Refinable Attack INvestigation) [23]	Hardware-specific	Library, kernel-level, dynamic binary ins.	libc ins., custom kernel module, Intel Pin	Records syscalls during runtime, replays offline under a DIFT
DataTracker [46]	Hardware-specific	Dinamic binary ins.		
INSPECTOR	Hardware-specific	Library ins., ISA extensions	libthreads ins., Intel PT ISA extensions	
MPI (Multiple Perspectives attack Investigation) [35]	Unsuitable method	Compile-time ins.	LLVM pass	Requires manual input
LDX (Lightweight Dual Execution) [28]	Unsuitable method	Compile-time ins.	LLVM pass	
MCI (Modeling-based Causality Inference) [29]	Unsuitable method	Compile-time ins.	LDX (LLVM pass)	
Ma et al. [36]	Unsuitable method	Kernel ins.	ETW	
S2Logger [47]	Unsuitable method	Kernel ins.	Custom kernel module or Linux Security Module	
ProTracer [34]	Unsuitable method	Kernel ins.	Linux tracepoints, custom kernel module	
Hi-Fi [41]	Unsuitable method	Kernel ins.	Linux Security Module	
Lineage FS [44]	Unsuitable method	Kernel ins.	Modified kernel	
PASS/Pasta [39]	Unsuitable method	Kernel ins., filesystem ins.	Modified kernel, VFS	
PASSv2/Lasagna [38]	Unsuitable method	Kernel ins., filesystem, library ins.	Modified kernel, instrumented libc, VFS	
RTAG [22]	Unsuitable method	Kernel ins.	Modified kernel	Record/reply like RAIN
LPM/ProvMon [3]	Unsuitable method	Kernel ins.	Modified kernel, kernel module, NetFilter	
CamFlow [40]	Unsuitable method	Kernel ins.	Linux Security Module and NetFilter	
LPROV [52]	Unsuitable method	Kernel, dynamic binary, static binary ins.	Custom kernel module, custom loader, BEEP	
Panorama [54]	Unsuitable method	VM ins.	QEMU	
PROV-Tracer [45]	Unsuitable method	VM ins.	QEMU, PANDA	

3.2 Suitability review

Our resulting candidates use a variety of different methods of collecting provenance events. We describe the collection methods below and assess their suitability in Sandia's context:

- **Tracing**: The Linux kernel also exposes ways for the user to trace a running program (often to implement debuggers) called 'ptrace(2)' [42]. Unprivileged users can use the `ptrace(2)` syscall [42] which can modify and trap one of their own processes. These do not require super-user and are scoped to a subset of the processes and filesystem. Tracing services are **suitable**, provided their performance overhead is small, which the performance study aims to evaluate.

- **Audit service**: The Linux kernel exposes multiple ways to log input and output operations for security auditing purposes (intrusion detection, digital forensics, etc.). Two of which are: Linux Auditing Framework (also called 'auditd') and enhanced Berkeley Packet Filter (eBPF) [6]. These methods generally require super-user access, and while that is a security risk, the attack surface can be minimized by encapsulating the audit service in a privileged daemon exposed to unprivileged users (same way Docker works). These methods are **suitable**.

- **Filesystem instrumentation**: Many kernels support software file systems, which pass through I/O calls to an underlying file system after modifying or logging the request. For example Linux offers a Filesystem in User SpacE (FUSE) interface [15] and and an older kernel-space Virtual File System (VFS) interface [18]. Filesystem-level provenance collection is **suitable, if they do not involve kernel modification**.

- **Library instrumentation** (also called library interposition and the `LD_PRELOAD` trick): Dynamically-linked programs ask the system to find an implementation of a library/application binary interface (ABI) at runtime. Library instrumentation supplies a wrapper library which implements that ABI by logging and passing certain function calls to an underlying implementation. For example, one could write a wrapper around the libc's `open` function, so when programs open a file, that request gets logged. It is an open question whether the performance overhead is low enough. It is also an open question whether HPC applications in the wild use libc and load it dynamically. Some applications (e.g., all Go programs, by default) do not use libc at all, some may use link against libc but do I/O directly, and some may link against libc statically (especially MUSL libc). Library instrumentation would not be able to know about the I/O from those programs, but we suspect this is not common for computational experiments 'in the wild'. Library instrumentation method is **suitable**, at least as a candidate to the performance test.

- **Static or dynamic binary instrumentation**: Binary instrumentation rewrites a binary executable before it runs (static) or while it is running (dynamic). This method approaches the power of VM extensions without the overhead and the power of kernel-level syscall trapping but at the user-level. However, dynamic instrumentation often requires an Intel CPU tool called Pin [33], which would go against the grain of reproducibility, which wants as many users as possible (especially cloud users) to be able to use the provenance system. Static instrumentation is simpler than dynamic binary instrumentation, as it would only require a specific instruction-set (e.g., x86_64). Binary instrumentation is **suitable, if it is not hardware dependent**.

- **Compile-time instrumentation**: A compiler pass can analyze and emit provenance data, especially intra-program control flow, yielding a finer granularity with more precise dataflow. LLVM is the natural choice for this, because its modular makes writing a new pass simpler. However, this would require HPC system adminstrators to maintain independent builds of the entire software stack. Compile-time instrumentation is **too expensive** to be suitable.

- **Kernel instrumentation**: All input, output, and process launching has to go through the kernel, so the kernel can be modified to emit provenance data. This method involves modifying the kernel to wrap or hook into syscall handlers. However, modifying the kernel directly or through a kernel module increases the attack surface to an extent that it may be difficult to get approval on classified networks. Kernel-level provenance collection poses **too much security risk**.

- **VM instrumentation**: Virtual machines, such as QEMU, can run programs with dynamic taint tracking often implement as shadow memory e.g., Panorama[54]. The performance overhead of QEMU with shadow memory is quite great (20x for Panorama). VM extensions for provenance collection are **too slow** to be suitable.
- **ISA extensions**: Intel released an extension called Processor Trace [27] which can trace the intra-process control-flow at a fine granularity, but it is **too hardware-specific** to be suitable (see the discussion for dynamic binary analysis).

Within these tools, we filter based on the following criteria:

1. We rule out tools based on whether their collection method is suitable for use in Sandia's context.
2. We look for collection methods with caveats (i.e., hardware specific, platform specific, and methods involving kernel modification)
3. We check for source code in the original paper, Google search (the first 20 results), GitHub (all results), and BitBucket (all results).

This leaves us with just three candidates: **SPADE+auditd**, **SPADE+FUSE**, and **OPUS**. . However, it appears OPUS is fatally broken in a way that prevents it from recalling provenance data it stored, and debugging that issue is out-of-scope for this project. Since the ProvMark artifact is available, we study how they use OPUS, but the ProvMark aftifact does not actually contain the code which sets up the OPUS tool.

This leaves several suitable SLRP collection methods untested! At the very least, even though there is no provenance tool, we want to test at least a "mock" examples of every suitable SLRP method. Once we know the order-of-magnitude overhead of the collection method itself, we can decide which would be worth using in a new SLRP tool.

- To test the library instrumentation overhead, we use **Darshan** and **fsatrace**. Darshan is an I/O performance analysis tool, whereas fsatrace is a barebones logging tool. Both use `LD_PRELOAD` to interpose libc I/O calls. We expect a provenance tool based on library instrumentation would have runtime characteristics somewhere between Darshan and fsatrace.
- To test the program tracing overhead, we use **rr**, **CDE**, **strace**, and **ltrace**. Rr and CDE are full solutions for replaying an execution; not only do they log every I/O call, they also store the contents of these calls which is quite expensive in time, RAM, and storage. On the other hand, strace, and ltrace are just barebones logging/tracing utilities; they just print a line every I/O call. Strace logs syscalls and ltrace logs libc calls. We are interested in whether syscall-level or libc-level will be more efficient. We expect a provenance tool based on ptrace would have runtime characteristics somewhere between the record/replay tools (rr and CDE) and barebones logging tools (strace, ltrace).
- SPADE+auditd does test one possible auditing method, but eBPF another auditing method that no conventional tool uses yet. eBPF is potentialy more efficient because it can safely execute a limited set of programs (not Turing complete) inside the kernel, which gets rid of expensive context switches between user- and kernel-space. To test auditing with eBPF, we use **bpftrace** with a custom script. Bpftrace compiles a simple scripting language into eBPF code and injects it into the kernel. Our script captures file I/O operations beginning with a certain prefix and logs them (comparable to strace and ltrace).

Thus, we arrive at the candidates in Table 4.

[2]The ProvMark artifact [10] does not have the infrastructure to set up OPUS; where ProvMark would have that infrastructure, the code only says here:

> As OPUS is not published anywhere in the internet, it is not able to generate a vargrant file that completely install opus within…You will need to obtain your own copy of OPUS and extracted within the vagrant VM in order to use OPUS with Prov Mark system.

OPUS *is* available on GitHub at the time of writing.

Tool	Collection method	Collection tool	Notes
SPADE+auditd	Auditing	auditd	Most used in prior work
bpftrace	Auditing	eBPF	Potentially more efficient than auditd
SPADE+FUSE	Filesystem ins.	FUSE	
rr	Tracing	ptrace syscalls	Expected to be slower than prov. collection
CDE	Tracing	ptrace syscalls	Expected to be slower than prov. collection
strace	Tracing	ptrace syscalls	Expected to be faster than prov. collection
ltrace	Tracing	ptrace libc calls	Expected to be faster than prov. collection
fsatrace	Lib ins.	libc instrumentation	
Darshan	Lib ins.	libc instrumentation	

Table 4: Candidates for performance analysis, after filtered for suitability and augmented with "mock" collectors

3.3 Performance analysis results

4 Discussion

How do these methods interact with containers?

4.1 Threats to validity

4.1.1 Limitations in candidate selection

We ruled out several categories of collection methods in order to limit our candidates. We believe that our ruling out is justified by suitability needs. However, we risk prematurely ruling out potentially important methods.

However, we believe this risk is mitigated. In prior work, the only way to tell when the kernel hit a certain point was to modify the code to insert a logging statement. However, since this became a common task, moder Linux kernels ship with built-in tracepoints [11] and kprobes [26], which can be instrumented with eBPF. Since eBPF can instrument these points without kernel modification, the risk of ruling out kernel modification is mitigated.

Provenance can be logged and processed offline (heavy space requirements) or processed online (greater performance impact). We wanted to try online processing methods, but we could not find source-code for those.

Another issue is provenance explosion. Some authors focus on creating one "runner" script that launches everything. If it does everything by library calls instead of subprocess calls, then SLRP would see just one process that reads thousands of files and writes thousands of files. The SLRP graph useless, because the granularity is too coarse; all SLRP can conclude is that any of the thousand inputs might affect all of the thousand outputs. This problem is called **provenance explosion**. To mitigate provenance explosion prior work suggests detect intraprogram dataflow (using compile-time instrumentation or binary instrumentation) or even more complex methods like taint tracking. Our first priority was to assess the feasibility of the simplest approach; then if that approach has prohibitive overheads, future work should look at provenance graph pruning.

Future work should compile-from-source options. HPC systems might already have the facilities to compile the entire software stack from source by virtue of using the Spack package manager [16].

5 Conclusion

6 Future work

References

[1] *About DTrace*. URL: http://dtrace.org/blogs/about/ (visited on 08/23/2023).

[2] Nikilesh Balakrishnan et al. "{OPUS}: A Lightweight System for Observational Provenance in User Space". In: 5th USENIX Workshop on the Theory and Practice of Provenance (TaPP 13). 2013. URL: https://www.usenix.org/conference/tapp13/technical-sessions/presentation/balakrishnan (visited on 07/06/2023).

[3] Adam Bates et al. "Trustworthy {Whole-System} Provenance for the Linux Kernel". In: 24th USENIX Security Symposium (USENIX Security 15). 2015, pp. 319–334. ISBN: 978-1-939133-11-3. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/bates (visited on 08/25/2023).

[4] Ghita Berrada et al. "A baseline for unsupervised advanced persistent threat detection in system-level provenance". In: *Future Generation Computer Systems* 108 (July 1, 2020), pp. 401–413. ISSN: 0167-739X. DOI: 10.1016/j.future.2020.02.015. URL: https://www.sciencedirect.com/science/article/pii/S0167739X19320448 (visited on 08/23/2023).

[5] Dirk Beyer, Stefan Löwe, and Philipp Wendler. "Reliable benchmarking: requirements and solutions". In: *Int J Softw Tools Technol Transfer* 21.1 (Feb. 6, 2019). interest: 90, pp. 1–29. ISSN: 1433-2779, 1433-2787. DOI: 10.1007/s10009-017-0469-y. URL: https://link.springer.com/10.1007/s10009-017-0469-y (visited on 06/30/2022).

[6] *BPF Documentation*. The Linux Kernel documentation. URL: https://docs.kernel.org/bpf/index.html (visited on 08/24/2023).

[7] Uri Braun et al. "Issues in Automatic Provenance Collection". In: *Provenance and Annotation of Data*. Ed. by Luc Moreau and Ian Foster. Red. by David Hutchison et al. Vol. 4145. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 171–183. ISBN: 978-3-540-46302-3 978-3-540-46303-0. DOI: 10.1007/11890850_18. URL: http://link.springer.com/10.1007/11890850_18 (visited on 08/23/2023).

[8] Lucian Carata et al. "A primer on provenance". In: *Commun. ACM* 57.5 (May 1, 2014), pp. 52–60. ISSN: 0001-0782. DOI: 10.1145/2596628. URL: https://dl.acm.org/doi/10.1145/2596628 (visited on 08/23/2023).

[9] Sheung Chi Chan et al. "Expressiveness Benchmarking for {System-Level} Provenance". In: 9th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2017). 2017. URL: https://www.usenix.org/conference/tapp17/workshop-program/presentation/chan (visited on 08/23/2023).

[10] Sheung Chi Chan et al. "ProvMark: A Provenance Expressiveness Benchmarking System". In: *Proceedings of the 20th International Middleware Conference*. Middleware '19. New York, NY, USA: Association for Computing Machinery, Dec. 9, 2019, pp. 268–279. ISBN: 978-1-4503-7009-7. DOI: 10.1145/3361525.3361552. URL: https://dl.acm.org/doi/10.1145/3361525.3361552 (visited on 08/21/2023).

[11] Matthieu Desnoyers. *Using the Linux Kernel Tracepoints*. The Linux Kernel documentation. URL: https://www.kernel.org/doc/html/latest/trace/tracepoints.html (visited on 08/24/2023).

[12] *Event Tracing - Win32 apps*. Jan. 7, 2021. URL: https://learn.microsoft.com/en-us/windows/win32/etw/event-tracing-portal (visited on 08/23/2023).

[13] Daren Fadolalkarim, Asmaa Sallam, and Elisa Bertino. "PANDDE: Provenance-based ANomaly Detection of Data Exfiltration". In: *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. CODASPY'16: Sixth ACM Conference on Data and Application Security and Privacy. CODASPY '16. New Orleans Louisiana USA: Association for Computing Machinery, Mar. 9, 2016, pp. 267–276. ISBN: 978-1-4503-3935-3. DOI: 10.1145/2857705.2857710. URL: https://dl.acm.org/doi/10.1145/2857705.2857710 (visited on 08/24/2023).

[14] Juliana Freire et al. "Provenance for Computational Tasks: A Survey". In: *Comput. Sci. Eng.* 10.3 (May 2008). interest: 97, pp. 11–21. ISSN: 1521-9615. DOI: 10.1109/MCSE.2008.79. URL: http://ieeexplore.ieee.org/document/4488060/ (visited on 07/08/2022).

[15] *FUSE*. The Linux Kernel documentation. URL: https://www.kernel.org/doc/html/latest/filesystems/fuse.html (visited on 08/24/2023).

[16] Todd Gamblin et al. "The Spack package manager: bringing order to HPC software chaos". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15. interest: 80. New York, NY, USA: Association for Computing Machinery, Nov. 15, 2015, pp. 1–12. ISBN: 978-1-4503-3723-6. DOI: 10.1145/2807591.2807623. URL: https://doi.org/10.1145/2807591.2807623 (visited on 04/10/2022).

[17] Ashish Gehani and Dawood Tariq. "SPADE: Support for Provenance Auditing in Distributed Environments". In: *Middleware 2012*. Ed. by Priya Narasimhan and Peter Triantafillou. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 101–120. ISBN: 978-3-642-35170-9. DOI: 10.1007/978-3-642-35170-9_6.

[18] Gooch. *Overview of the Linux Virtual File System*. The Linux Kernel documentation. URL: https://docs.kernel.org/filesystems/vfs.html (visited on 08/24/2023).

[19] Wajih Ul Hassan et al. "OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis". In: *Network and Distributed System Security Symposium* (Jan. 2020). URL: https://par.nsf.gov/biblio/10146531-omegalog-high-fidelity-attack-investigation-via-transparent-multi-layer-log-analysis (visited on 08/23/2023).

[20] Wajih Ul Hassan et al. "Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs". In: *Proceedings 2018 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA: Internet Society, 2018. ISBN: 978-1-891562-49-5. DOI: 10.14722/ndss.2018.23141. URL: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_07B-1_Hassan_paper.pdf (visited on 08/23/2023).

[21] Yang Ji, Sangho Lee, and Wenke Lee. "RecProv: Towards Provenance-Aware User Space Record and Replay". In: *Provenance and Annotation of Data and Processes*. Ed. by Marta Mattoso and Boris Glavic. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 3–15. ISBN: 978-3-319-40593-3. DOI: 10.1007/978-3-319-40593-3_1.

[22] Yang Ji et al. "Enabling Refinable {Cross-Host} Attack Investigation with Efficient Data Flow Tagging and Tracking". In: 27th USENIX Security Symposium (USENIX Security 18). 2018, pp. 1705–1722. ISBN: 978-1-939133-04-5. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/jia-yang (visited on 08/23/2023).

[23] Yang Ji et al. "RAIN: Refinable Attack Investigation with On-demand Inter-Process Information Flow Tracking". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. New York, NY, USA: Association for Computing Machinery, Oct. 30, 2017, pp. 377–390. ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3134045. URL: https://dl.acm.org/doi/10.1145/3133956.3134045 (visited on 08/23/2023).

[24] Tomas Kalibera and Richard Jones. *Quantifying Performance Changes with Effect Size Confidence Intervals*. July 21, 2020. DOI: 10.48550/arXiv.2007.10899. arXiv: 2007.10899[cs,stat]. URL: http://arxiv.org/abs/2007.10899 (visited on 08/22/2023).

[25] Vasileios P. Kemerlis et al. "libdft: practical dynamic data flow tracking for commodity systems". In: *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments*. VEE '12. New York, NY, USA: Association for Computing Machinery, Mar. 3, 2012, pp. 121–132. ISBN: 978-1-4503-1176-2. DOI: 10.1145/2151024.2151042. URL: https://dl.acm.org/doi/10.1145/2151024.2151042 (visited on 08/23/2023).

[26] Jim Keniston, Prasanna S Panchamukhi, and Masami Hiramatsu. *Kernel Probes (Kprobes)*. The Linux Kernel documentation. URL: https://www.kernel.org/doc/html/latest/trace/kprobes.html (visited on 08/24/2023).

[27] Andi Kleen and Beeman Strong. "Intel® Processor Trace on Linux". Tracing Summit 2015. Seattle, Washington, USA, Aug. 10, 2015.

[28] Yonghwi Kwon et al. "LDX: Causality Inference by Lightweight Dual Execution". In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '16. New York, NY, USA: Association for Computing Machinery, Mar. 25, 2016, pp. 503–515. ISBN: 978-1-4503-4091-5. DOI: 10.1145/2872362.2872395. URL: https://dl.acm.org/doi/10.1145/2872362.2872395 (visited on 08/23/2023).

[29] Yonghwi Kwon et al. "MCI : Modeling-based Causality Inference in Audit Logging for Attack Investigation". In: *Proceedings 2018 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA: Internet Society, 2018. ISBN: 978-1-891562-49-5. DOI: 10.14722/ndss.2018.23306. URL: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_07B-2_Kwon_paper.pdf (visited on 08/23/2023).

[30] Brian Lee, Abir Awad, and Mirna Awad. "Towards Secure Provenance in the Cloud: A Survey". In: *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC). Dec. 2015, pp. 577–582. DOI: 10.1109/UCC.2015.102.

[31] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. "High Accuracy Attack Provenance via Binary-based Execution Partition". In: *Proceedings of the 2017 Network and Distributed System Security (NDSS) Symposium*. Network and Distributed System Security (NDSS) 2017. 2017.

[32] Zhenyuan Li et al. "Threat detection and investigation with system-level provenance graphs: A survey". In: *Computers & Security* 106 (July 1, 2021), p. 102282. ISSN: 0167-4048. DOI: 10.1016/j.cose.2021.102282. URL: https://www.sciencedirect.com/science/article/pii/S0167404821001061 (visited on 08/23/2023).

[33] Chi-Keung Luk et al. "Pin: building customized program analysis tools with dynamic instrumentation". In: *SIGPLAN Not.* 40.6 (June 12, 2005), pp. 190–200. ISSN: 0362-1340. DOI: 10.1145/1064978.1065034. URL: https://dl.acm.org/doi/10.1145/1064978.1065034 (visited on 08/24/2023).

[34] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. "ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting". In: *Proceedings 2016 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA: Internet Society, 2016. ISBN: 978-1-891562-41-9. DOI: 10.14722/ndss.2016.23350. URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/protracer-towards-practical-provenance-tracing-alternating-logging-tainting.pdf (visited on 08/23/2023).

[35] Shiqing Ma et al. "{MPI}: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning". In: 26th USENIX Security Symposium (USENIX Security 17). 2017, pp. 1111–1128. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/ma (visited on 08/23/2023).

[36] Shiqing Ma et al. "Accurate, Low Cost and Instrumentation-Free Security Audit Logging for Windows". In: *Proceedings of the 31st Annual Computer Security Applications Conference*. ACSAC '15. New York, NY, USA: Association for Computing Machinery, Dec. 7, 2015, pp. 401–410. ISBN: 978-1-4503-3682-6. DOI: 10.1145/2818000.2818039. URL: https://dl.acm.org/doi/10.1145/2818000.2818039 (visited on 08/23/2023).

[37] markruss. *Sysmon - Sysinternals*. Apr. 11, 2023. URL: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon (visited on 08/23/2023).

[38] Kiran-Kumar Muniswamy-Reddy et al. "Layering in provenance systems". In: *Proceedings of the 2009 conference on USENIX Annual technical conference*. USENIX'09. USA: USENIX Association, June 14, 2009, p. 10. DOI: 10.5555/1855807.1855817. (Visited on 07/18/2023).

[39] Kiran-Kumar Muniswamy-Reddy et al. "Provenance-Aware Storage Systems". In: *2006 USENIX Annual Technical Conference*. 2006 USENIX Annual Technical Conference. Accepted: 2015-12-07T19:07:43Z. 2006. URL: https://dash.harvard.edu/handle/1/23853812.

[40] Thomas Pasquier et al. "Practical whole-system provenance capture". In: *Proceedings of the 2017 Symposium on Cloud Computing*. SoCC '17. New York, NY, USA: Association for Computing Machinery, Sept. 24, 2017, pp. 405–418. ISBN: 978-1-4503-5028-0. DOI: 10.1145/3127479.3129249. URL: https://dl.acm.org/doi/10.1145/3127479.3129249 (visited on 08/23/2023).

[41] Devin J. Pohly et al. "Hi-Fi: collecting high-fidelity whole-system provenance". In: *Proceedings of the 28th Annual Computer Security Applications Conference*. ACSAC '12. New York, NY, USA: Association for Computing Machinery, Dec. 3, 2012, pp. 259–268. ISBN: 978-1-4503-1312-4. DOI: 10.1145/2420950.2420989. URL: https://dl.acm.org/doi/10.1145/2420950.2420989 (visited on 08/23/2023).

[42] *ptrace*. Linux manual page. URL: https://man7.org/linux/man-pages/man2/ptrace.2.html (visited on 08/24/2023).

[43] Lukas Rupprecht et al. "Improving reproducibility of data science pipelines through transparent provenance capture". In: *Proc. VLDB Endow.* 13.12 (Aug. 1, 2020), pp. 3354–3368. ISSN: 2150-8097. DOI: 10.14778/3415478.3415556. URL: https://dl.acm.org/doi/10.14778/3415478.3415556 (visited on 08/24/2023).

[44] Can Sar and Pei Cao. "Lineage File System". URL: http://crypto.stanford.edu/~cao/lineage.html (visited on 08/23/2023).

[45] Manolis Stamatogiannakis, Paul Groth, and Herbert Bos. "Decoupling provenance capture and analysis from execution". In: *Proceedings of the 7th USENIX Conference on Theory and Practice of Provenance*. TaPP'15. USA: USENIX Association, July 8, 2015, p. 3. (Visited on 08/24/2023).

[46] Manolis Stamatogiannakis, Paul Groth, and Herbert Bos. "Looking Inside the Black-Box: Capturing Data Provenance Using Dynamic Instrumentation". In: *Provenance and Annotation of Data and Processes*. Ed. by Bertram Ludäscher and Beth Plale. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 155–167. ISBN: 978-3-319-16462-5. DOI: 10.1007/978-3-319-16462-5_12.

[47] Chun Hui Suen et al. "S2Logger: End-to-End Data Tracking Mechanism for Cloud Data Provenance". In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. ISSN: 2324-9013. July 2013, pp. 594–602. DOI: 10.1109/TrustCom.2013.73.

[48] Young-Kyoon Suh et al. "EMP: execution time measurement protocol for compute-bound programs". In: *Software: Practice and Experience* 47.4 (Apr. 2017). interest: 71, pp. 559–597. ISSN: 0038-0644, 1097-024X. DOI: 10.1002/spe.2476. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2476 (visited on 08/22/2023).

[49] Salmin Sultana and Elisa Bertino. "A file provenance system". In: *Proceedings of the third ACM conference on Data and application security and privacy*. CODASPY '13. New York, NY, USA: Association for Computing Machinery, Feb. 18, 2013, pp. 153–156. ISBN: 978-1-4503-1890-7. DOI: 10.1145/2435349.2435368. URL: https://dl.acm.org/doi/10.1145/2435349.2435368 (visited on 08/23/2023).

[50] Dawood Tariq, Ali Masaim, and Ashish Gehani. "Towards Automated Collection of {Application-Level} Data Provenance". In: 4th USENIX Workshop on the Theory and Practice of Provenance (TaPP 12). 2012. URL: https://www.usenix.org/conference/tapp12/workshop-program/presentation/tariq (visited on 08/23/2023).

[51] Amin Vahdat and Thomas Anderson. "Transparent result caching". In: *Proceedings of the annual conference on USENIX Annual Technical Conference*. ATEC '98. USA: USENIX Association, June 15, 1998, p. 3. (Visited on 08/23/2023).

[52] Fei Wang et al. "Lprov: Practical Library-aware Provenance Tracing". In: *Proceedings of the 34th Annual Computer Security Applications Conference*. ACSAC '18. New York, NY, USA: Association for Computing Machinery, Dec. 3, 2018, pp. 605–617. ISBN: 978-1-4503-6569-7. DOI: 10.1145/3274694.3274751. URL: https://dl.acm.org/doi/10.1145/3274694.3274751 (visited on 08/24/2023).

[53] Runqing Yang et al. "UISCOPE: Accurate, Instrumentation-free, and Visible Attack Investigation for GUI Applications". In: *Proceedings 2020 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA: Internet Society, 2020. ISBN: 978-1-891562-61-7. DOI: 10.14722/ndss.2020.24329. URL: https://www.ndss-symposium.org/wp-content/uploads/2020/02/24329.pdf (visited on 08/23/2023).

[54] Heng Yin et al. "Panorama: capturing system-wide information flow for malware detection and analysis". In: *Proceedings of the 14th ACM conference on Computer and communications security*. CCS '07. New York, NY, USA: Association for Computing Machinery, Oct. 28, 2007, pp. 116–127. ISBN: 978-1-59593-703-2. DOI: 10.1145/1315245.1315261. URL: https://dl.acm.org/doi/10.1145/1315245.1315261 (visited on 08/23/2023).

[55] Yong Zhao, Michael Wilde, and Ian Foster. "Applying the Virtual Data Provenance Model". In: *Provenance and Annotation of Data*. Ed. by Luc Moreau and Ian Foster. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 148–161. ISBN: 978-3-540-46303-0. DOI: 10.1007/11890850_16.

[56] Michael Zipperle et al. "Provenance-based Intrusion Detection Systems: A Survey". In: *ACM Comput. Surv.* 55.7 (Dec. 15, 2022), 135:1–135:36. ISSN: 0360-0300. DOI: 10.1145/3539605. URL: https://dl.acm.org/doi/10.1145/3539605 (visited on 08/23/2023).