Evaluating prior research on system-level provenance for practical use

Samuel Grayson Reed Milewicz Daniel S. Katz
Darko Marinov

2024-01-30

Background

The Oxford English Dictionary defines *provenance* as "a record of the ultimate origin and passage of an item through its previous owners." In the domain of computational science, we will specialize this definition to mean "the computational input artifacts and computational processes that influenced a certain computational output artifact", as described by Freire [1]. This definition is usually applied recursively, so the computational provenance would describe the output, the inputs, the inputs' inputs, and so forth until the artifacts were generated outside the computational system (they could be generated by an opaque computational system or directly by a user)[1].

Computational provenance has many potential applications, including the following [2]:

1. **Caching**. A black-box system that captures provenance knows what inputs affect the output for each process, so it knows which processes to re-execute without a user-supplied dependency graph programmer, as in traditional Make[2].
2. **Comprehension**. Provenance data helps the user understand, debug, and document workflows.
3. **Management**. Provenance data can help catalog, label, and recall experimental results based on the input parameters.
4. **Reproducibility**. A description of the inputs and processes used to generate a specific output can aid manual and automatic reproduction of that

[1] DSK: perhaps better "the artifacts come from outside the computational system, such as observations"?

[2] I think of caching as knowing what not to re-execute, aka memoization. What you describe here is more checkpointing to me. They are related…]

output[3].

One can capture computational provenance by modifying an application to report provenance data, leveraging a workflow engine or programming language to report provenance data, or leveraging an operating system to emit provenance data to report provenance data [1].

- **Application-level** provenance is the most semantically rich, since it knows the use of each input at the application-level (see fig. 1a), but the least general, since each application would have to be modified individually.
- **Workflow-level** or **language-level** provenance is a middle ground in semantic richness and generality; it only knows the use of inputs in a dataflow sense (see fig. 1b), but all applications using the provenance-modified workflow engine or programming language would emit provenance data without themselves being modified to emit provenance data.
- **System-level** is the most general, since all applications on the system would emit provenance data, but it is the least semantically rich, since observed dependencies may overapproximate the true dependencies (see lst. **??**, fig. 1c).

Motivation

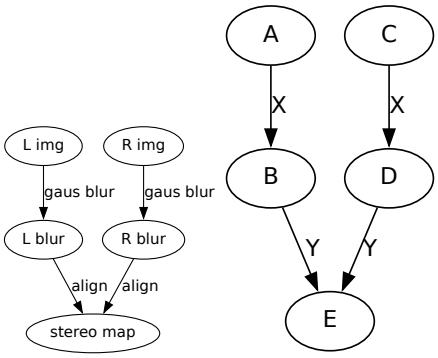
Lack of reproducibility in computational experiments undermines the long-term credibility of science and hinders the day-to-day work of researchers. Empirical studies [4]–[7] show that reproducibility is rarely achieved in practice, probably due to its difficulty under the short time constraints that scientists have for new publications.[4] Rather than exhorting researchers to spend more of their short time-budget on reproducibility, it may be more effective to make reproducibility easier to achieve with the same amount of effort.

Provenance data improves manual reproducibility, because users have a record of the inputs, outputs, and processes used to create a computational artifact. Provenance data also has the potential to enable automatic reproducibility, if the process trace is detailed enough to be "re-executed".

One may imagine an abstract tradeoff curve between "enabling provenance applications such as reproducibility" as the horizontal axis increasing rightwards and "cost to implement" that provenance data on the vertical axis increasing upwards (fig. fig. **??**). A typical status quo, not collecting any provenance data and not using workflows, is at the bottom left: no added cost and does nothing to enable provenance applications. System-level, workflow/language-level, and

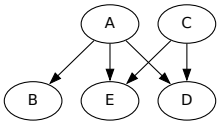
[3]"Reproduction", in the ACM sense, where a **different team** uses the **same artifacts** to generate the output artifact [3].

[4]DSK: also due to a lack of incentives for achieving it, at least in comparison to other things that could be done with the same effort.



(a) Application-level prov (b) Workflow-level prov

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



(c) System-level prov

Figure 1: Several provenance graphs collected at different levels.

application-level are on a curve, increasing cost and enabling more provenance applications.

The initial jump in cost from nothing to system-level may be low because the user need not change *anything* about their application; they merely need to install some provenance tracer onto their system and run their code, without modifying it, in the tracer. Perceived ease of use is a critical factor in the adoption of new technologies (formalized in the Technology Acceptance Model [8]). Although the user may eventually use more semantically rich provenance, low-cost system-level provenance would get provenance's "foot in the door". While this data is less rich than that of the workflow or application level, it may be enough to enable important applications such as caching, reproducibility, and file-level comprehension. Since system-level provenance collection is a possibly valuable tradeoff between cost and enabling provenance applications, system-level provenance will be the subject of this work.

Contributions

This work aims to summarize state of the art, establish goalposts for future research in the area, and identify which provenance tools are practically usable.

This work contributes:

- **A rapid review**: There are a scores of academic publications on system-level provenance (see tbl. 1), so we collate all of prior provenance tools and classify them by *capture method* (e.g., does the provenance collector require you to load a kernel module or run your code in a VM?).
- **A benchmark suite**: Prior work does not use a consistent set of benchmarks; often publications use an overlapping set of benchmarks from prior work . We collate benchmarks used in prior work, add some unrepresented areas, and find a statistically valid subset of the benchmark.
- **A quantitative performance comparison**: Prior publications often only compares the performance their provenance tool to the baseline, no-provenance performance, not to other provenance tools. It is difficult to compare provenance tools, given data of different benchmarks on different machines. This work runs a consistent set of benchmarks on a single machine over all provenance tools.
- **A predictive performance model**: The performance overhead of a single provenance system vary from <1% to 23% [9] based on the application, so a single number for overhead is not sufficient. This work develops a statistical model for predicting the overhead of $X application in $Y provenance system based on $Y provenance system's performance on our benchmark suite and $X application's performance characteristics (e.g., number of I/O syscalls).

Methods

Rapid review

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

Rapid Reviews are a lighter-weight alternative to systematic literature reviews with a focus on timely feedback for decision-making. Schünemann and Moja [10] show that Rapid Reviews can yield substantially similar results to a systematic literature review, albeit with less detail. Although developed in medicine, Cartaxo et al. show that Rapid Reviews are useful for informing software engineering design decisions [11], [12].

We conducted a rapid review with the following parameters:

- **Objective**: Identify system-level provenance collection tools.
- **Search terms**: "system-level" AND "provenance"
- **Search engine**: Google Scholar
- **Number of results**: 50
  - This threshold is the point of diminishing returns, as no new tools came up in the 40th – 50th results.
- **Criteria**: A relevant publication would center on one or more operating system-level tools that capture file provenance. A tool requiring that the user use a specific application or platform would be irrelevant.

We record the following features for each system-level provenance tool:

- **Capture method**: What method does the tool use to capture provenance?
  - **User-level tracing**: A provenance tool may use "debugging" or "tracing" features provided by the kernel to trace the I/O operations of another program, e.g., `ptrace(2)` [13].
  - **Builtin auditing service**: A provenance tool may use auditing service built in to the kernel, e.g., Linux Auditing Framework [14], enhanced Berkeley Packet Filter (eBPF) [15], kprobes [16], and ETW [17] for Windows.
  - **Filesystem instrumentation**: A provenance tool may set up a file system, so it can log I/O operations, e.g., using Filesystem in User SpacE (FUSE) interface [18], or Virtual File System (VFS) interface [19].
  - **Dynamic library instrumentation**: A provenance tool may replace a library used to execute I/O operations (e.g., glibc) with one that logs the calls before executing them.

  - **Binary instrumentation**: A provenance tool may use binary instrumentation (dynamic or static) to identify I/O operations in another program.
  - **Compile-time instrumentation**: A provenance tool may be a compiler pass that modifies the program to emit provenance data, especially intra-program control flow.
  - **Kernel instrumentation**: A provenance tool may be a modified kernel either by directly modifying the kernel's source tree or loading a kernel module.
  - **VM instrumentation**: A provenance tool may execute the program in a virtual machine, where it can observe the program's I/O operations.
- **Is source code available?**: We use the categorical codes given by [6] to describe whether the source code is in the article, found on the web, found by an email from the author, refused from an email by the author, or the authors did not reply.

Selecting benchmarks

Using the tools selected above, we identify all benchmarks that have been used in prior work. To get consistent measurements, we select as many benchmarks and provenance tracers as we reasonably can, and run a complete matrix (every tracer on every benchmark).

We also added new benchmarks:

- **Workflows**: Only one of the commonly used benchmarks from prior work (BLAST) resembles an e-science workflow (multiple intermediate inputs/outputs on the filesystem), so we added non-containerized Snakemake workflows from prior work [5].
- **Data science**: None of the benchmarks resembled a typical data science program, so we added the most popular Notebooks from Kaggle.com, a data science competition website.
- **Compilations**: Prior work uses compilation of Apache or of Linux. We added compilation of several other packages (any package in Spack) to our benchmark. Compiling packages is a good use-case for a provenance tracer, because a user might trial-and-error multiple compile commands and not remember the exact sequence of "correct" commands; the provenance tracker would be able to recall the commands which did not get overwritten, so the user can know what commands "actually worked". [5]
- **Computational simulations**: High-performance computing (HPC) scientific simulations could benefit from provenance tracing. These HPC

[5] DSK: this reminds me of VisTrails from Utah

applications may have access patterns quite different than conventional desktop applications. The xSDK framework [20] collects a

In order to reduce the amount of work needed to reduce the amount of time it takes for the now numerous benchmarks to run for future work, we subset our benchmark based on interpolative matrix decomposition. **Interpolative decomposition** (ID) factors a matrix $A \in \mathbb{R}^{n \times m}$ of rank k into two factors $B \times P$ where the B contains a subset of the columns of A and a subset of the columns of P form the $k \times k$ identity matrix [21][6]. When A is not of rank k, the factorization is guaranteed to be within a certain factor of the best unrestricted $R^{n \times k} \times R^{k \times m}$ factorization.

To apply ID to benchmark subsetting, suppose A_{ij} was the runtime performance of the ith system on the jth benchmark, then the ID essentially finds k "non-redundant" benchmarks and approximates the rest (which I call "redundant") as a linear function of the "non-redundant" benchmarks, while minimizing $\|A - BP\|_2$.

$$\begin{pmatrix} | & & | \\ A_{:,1} & \cdots & A_{:,n} \\ | & & | \end{pmatrix} = A \approx BP = \begin{pmatrix} | & & | \\ A_{:,1} & \cdots & A_{:,k} \\ | & & | \end{pmatrix} \begin{pmatrix} 1 & 0 & \cdots & 0 & | & & | \\ 0 & \ddots & & \vdots & P_{:,j} & \cdots & P_{:,m} \\ \vdots & & \ddots & 0 & | & & | \\ 0 & \cdots & 0 & 1 & | & & | \end{pmatrix}.$$

- If the jth benchmark is non-redundant, then the jth column of A is "copied" to the jth column of B, and the jth row of$P$ is set to the jth row of the identity matrix; multiplying B by P "passes through verbatim" the jth column of B. - Otherwise, the jth benchmark is redundant and the jth column of P is set to the coefficients of a linear regression from the non-redundant benchmarks to the $j$the benchmark; multiplying B by P evaluates a prediction based on the linear regression in the jth column.

Quantitative performance comparison

We use BenchExec [22] to precisely measure the CPU time, wall time, memory utilization, and other attributes of the process (including child processes) in a Linux CGroup without networking, isolated from other processes on the system.

Predictive performnace model

Results

Qualitative feature comparison

Tbl. 1 shows the provenance tools we collected and their qualitative features, while tbl. 2 shows the tools which have been called "system-level provenance tools" but do not fit the definition used in this paper. Of these, tbl. 3 shows the

[6] Also see the function scipy.linalg.interpolative of the Python package scipy.

benchmarks used to evaluate each tool. Running HTTP servers may be a popular benchmark because prior work focuses overwhelmingly on provenance for the sake of security (auditing, intrusion detection, or digital forensics); securing HTTP servers and web applications is a common task.

Table 1: Provenance trackers mentioned in primary and secondary studies in our search results.

Tool	Collection method	Collection tool	Notes
SPADE [23]	Audit, FS, **or** compile-time	Multiple[7]	Can use multiple sources
OPUS [24]	Library instrumentation	libc instrumentaiton	
FiPS [25]	FS. ins.	VFS	
RecProv [26]	Tracing	rr, ptrace	
PANDDE [27]	FS. ins.	Custom VFS	
URSprung [28]	Audit, file-system ins.	auditd, IBM Spectrum Scale	For IBM Spectrum Scale
Event Tracer for Windows [17]	Audit	NT Kernel	For Windows
TREC [29]	Audit	Proc filesystem	For Solaris
DTrace [30]	Audit	Respective kernels	Does processing in kernel
Sysmon [31]	Audit	NT Kernel	Implemented for Windows
Ma et al. [32]	Kernel ins.	ETW	
BEEP [33]	Dyn., static binary ins.	Intel Pin, PEBIL	
libdft [34]	Dyn. binary ins.	Intel Pin	
RAIN [35]	kernel ins., lib. ins., dyn. binary ins.	libc ins., custom kernel module, Intel Pin	Replays syscalls under DIFT
DataTracker [36]	Dinamic binary ins.		
MPI[37]	Compile-time ins.	LLVM pass	Requires manual input
LDX [38]	Compile-time ins.	LLVM pass	

[7]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.

Tool	Collection method	Collection tool	Notes
S2Logger [39]	Kernel ins.	Kernel module or Linux Security Module	
ProTracer [40]	Kernel ins.	Linux tracepoints, custom kernel module	
Hi-Fi [41]	Kernel ins.	Linux Security Module	
Lineage FS [42]	Kernel ins.	Modified kernel	
PASS/Pasta [43]	Kernel ins., filesystem ins.	Modified kernel, VFS	
PASSv2/Lasagna [9]	Kernel ins., filesystem, lib. ins.	Modified kernel, instrumented libc, VFS	
RTAG [44]	Kernel ins.	Modified kernel	
LPM/ProvMon [45]	Kernel ins.	Modified kernel, kernel module, NetFilter	
CamFlow [46]	Kernel ins.	Linux Security Module, NetFilter	
LPROV [47]	Kernel ins., library ins.	Custom kernel module, custom loader, BEEP	
Panorama [48]	VM ins.	QEMU	
PROV-Tracer [49]	VM ins.	QEMU, PANDA	

Table 2: Excluded tools.

Tool	Reason
ES3 [50]	specific to ES3 platform
Chimera [51]	specific to Chimera platform
INSPECTOR [52]	doesn't track files
MCI [44]	offline; depends on online-LDX
OmegaLog [53]	depends on app-level logs
LogGC [54]	contribution is deleting irrelevant events in the logs
UIScope [55]	captures UI interactions; uses ETW to capture I/O operations
Winnower [56]	specific to Docker Swarm

Table 3: Benchmarks used in various provenance publications.

Provenance publication	Benchmarks	Comparisons	Year
TREC [29]	open/close, compile Apache, compile LaTeX doc	Native	1999
PASS [43]	BLAST	Native ext2	2006
Panorama [48]	curl, scp, gzip, bzip2	Native	2007
PASSv2 [9]	BLAST, compile Linux, Postmark, Mercurial, Kepler	Native ext3, NFS	2009
SPADEv2 [23]	BLAST, compile Apache, Apache	Native	2012
Hi-Fi [41]	lmbench, compile Linux, Postmark	Native	2012
libdft [34]	scp, {tar, gzip, bzip2} x {extract, compress}	PIN	2012
LogGC [54]	RUBiS, Firefox, MC, Pidgin, Pine, Proftpd, Sendmail, sshd, vim, w3m, wget, xpdf, yafc, Audacious, bash, Apache, mysqld	[^loggc-bench]	2013
LPM/ProvMon [45]	lmbench, compile Linux, Postmark, BLAST	Native	2015
Ma et al. [32]	TextTransfer, Chromium, DrawTool, NetFTP, AdvancedFTP, Apache, IE, Paint, Notepad, Notepad++, simplehttp, Sublime Text	Native	2015
ProTracer [40]	Apache, miniHTTP, ProFTPD, Vim, Firefox, w3m, wget, mplayer, Pine, xpdf, MC, yafc	Auditd, BEEP, LogGC	2016
LDX [38]	SPEC CPU 2006, Firefox, lynx, nginx, tnftp, sysstat, gif2png, mp3info, prozilla	Native	2016
(LDX continued)	yopsweb, ngircd, gocr, Apache, pbzip2, pigz, axel, x264		
MPI [37]	Apache, bash, Evince, Firefox, Krusader, wget, most, MC, mplayer,	Audit, LPM-HiFi	2017
(MPI continued)	MPV, nano, Pine, ProFTPd, SKOD, TinyHTTPd, Transmission, Vim, w3m, Xpdf, Yafc		
CamFlow [46]	lmbench, postmark, unpack kernel, compile Linux, Apache, Memcache, redis, php, pybench	Native	2017
BEEP [33]	Apache, Vim, Firefox, wget, Cherokee, w3m, ProFTPd, yafc, Transmission, Pine, bash, MC, sshd, sendmail	Native	2017

Provenance publication	Benchmarks	Comparisons	Year
RAIN [35]	SPEC CPU 2006, cp linux, wget, compile libc, Firefox, SPLASH-3	Native	2017
Sciunit [57]	VIC, FIE	Native	2017
LPROV [47]	Apache, simplehttp, proftpd, sshd, firefox, filezilla, lynx, links, w3m, wget, ssh, pine, vim, emacs, xpdf	Native	2018
MCI [58]	Firefox, Apache, Lighttpd, nginx, ProFTPd, CUPS, vim, elinks,	BEEP	2018
(MCI continued)	alpine, zip, transmission, lftp, yafc, wget, ping, procps		
RTAG [44]	SPEC CPU 2006, scp, wget, compile llvm, Apache	RAIN	2018
URSPRING [28]	open/close, fork/exec/exit, pipe/dup/close, socket/connect, CleanML, Vanderbilt, Spark, ImageML	Native, SPADE+auditd	2020

Table 4: Benchmarks implemented by this work

Benchmark group	Uses in prior work
BLAST	4
Python data science	0
HTTP server/traffic (Apache httpd, miniHTTP, simplehttp, lighttpd, Nginx, tinyhttpd, cherokee)	10
HTTP serer/client (curl, wget, prozilla, axel)	9
FTP server/traffic (ProFTPd)	5
FTP client (lftp, yafc, tnftp, skod)	2
SPEC CPU 2006	3
Compile (Apache, LLVM, libc, Linux, LaTeX doc)	8
lmbench[8]	5
Postmark	4
SPLASH-3	1
Browsers (Firefox, Chromium, w3m)	8
VCS (Mercurial, git)	1
Un/archive ({compress, decompress} x tar x {nothing, bzip2, pbzip, gzip, pigz, zip})[9]	5
Shellbench (bash, sh)	3
Workflows (Snakemake-workflow-catalog, Nf-core, CleanML, Spark, VIC, FIE)	2

[8]Noting that lmbench contains open/close, fork/exec, and pipe/close.
[9]We count "unpack linux" as an occurrence of an un/archive benchmark.

Benchmark group	Uses in prior work
Microbenchmarks (open/close, fork/exec, pipe/dup/close, socket/connect)	2
xSDK codes	0
Small, fast binaries (procps, sysstat, gif2png, mp3info, gocr)	2
x264	1

Benchmarks

Quantitative performance comparison

Predictive model

Analysis

Threats to validity

Future work

Conclusion

[1] J. Freire, D. Koop, E. Santos, and C. Silva, "Provenance for Computational Tasks: A Survey," *Comput. Sci. Eng.*, vol. 10, no. 3, pp. 11–21, May 2008, doi: 10.1109/MCSE.2008.79. [Online]. Available: http://ieeexplore.ieee.org/document/4488060/. [Accessed: Jul. 08, 2022]

[2] J. F. Pimentel, J. Freire, L. Murta, and V. Braganholo, "A Survey on Collecting, Managing, and Analyzing Provenance from Scripts," *ACM Comput. Surv.*, vol. 52, no. 3, pp. 47:1–47:38, Jun. 2019, doi: 10.1145/3311955. [Online]. Available: https://dl.acm.org/doi/10.1145/3311955. [Accessed: Jan. 20, 2024]

[3] A. Inc. staff, "Artifact Review and Badging," Aug. 24, 2020. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-and-badging-current. [Accessed: Jan. 19, 2023]

[4] A. Trisovic, M. K. Lau, T. Pasquier, and M. Crosas, "A large-scale study on research code quality and execution," *Sci Data*, vol. 9, no. 1, 1, p. 60, Feb. 2022, doi: 10.1038/s41597-022-01143-6. [Online]. Available: https://www.nature.com/articles/s41597-022-01143-6. [Accessed: Dec. 13, 2022]

[5] S. Grayson, D. Marinov, D. S. Katz, and R. Milewicz, "Automatic Reproduction of Workflows in the Snakemake Workflow Catalog and nf-core Registries," in *Proceedings of the 2023 ACM Conference on Reproducibility and Replicability*, Jun. 2023, pp. 74–84, doi: 10.1145/3589806.3600037 [Online]. Available: https://dl.acm.org/doi/10.1145/3589806.3600037. [Accessed: Jan. 20, 2024]

[6] C. Collberg and T. A. Proebsting, "Repeatability in computer systems research," *Commun. ACM*, vol. 59, no. 3, pp. 62–69, Feb. 2016, doi: 10.1145/2812803. [Online]. Available: https://dl.acm.org/doi/10.1145/2812803. [Accessed: May 27, 2022]

[7] J. Zhao *et al.*, "Why workflows break — understanding and combating decay in Taverna workflows," in *2012 IEEE 8th International Conference on E-Science (e-Science)*, Oct. 2012, p. 9, doi: 10.1109/eScience.2012.6404482 [Online]. Available: https://www.research.manchester.ac.uk/portal/en/publications/why-workflows-break--understanding-and-combating-decay-in-taverna-workflows(cba81ca4-e92c-408e-8442-383d1f15fcdf)/export.html

[8] F. D. Davis, "A technology acceptance model for empirically testing new end-user information systems: Theory and results," Thesis, Massachusetts Institute of Technology, 1985 [Online]. Available: https://dspace.mit.edu/handle/1721.1/15192. [Accessed: Jun. 03, 2022]

[9] K.-K. Muniswamy-Reddy *et al.*, "Layering in provenance systems," in *Proceedings of the 2009 conference on USENIX Annual technical conference*, Jun. 2009, p. 10, doi: 10.5555/1855807.1855817.

[10] H. J. Schünemann and L. Moja, "Reviews: Rapid! Rapid! Rapid! …And systematic," *Systematic Reviews*, vol. 4, no. 1, p. 4, Jan. 2015, doi: 10.1186/2046-4053-4-4. [Online]. Available: https://doi.org/10.1186/2046-4053-4-4. [Accessed: Oct. 27, 2023]

[11] B. Cartaxo, G. Pinto, and S. Soares, "The Role of Rapid Reviews in Supporting Decision-Making in Software Engineering Practice," in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, Jun. 2018, pp. 24–34, doi: 10.1145/3210459.3210462 [Online]. Available: https://dl.acm.org/doi/10.1145/3210459.3210462. [Accessed: Oct. 26, 2023]

[12] B. Cartaxo, G. Pinto, and S. Soares, "Rapid Reviews in Software Engineering," in *Contemporary Empirical Methods in Software Engineering*, M. Felderer and G. H. Travassos, Eds. Cham: Springer International Publishing, 2020, pp. 357–384 [Online]. Available: https://doi.org/10.1007/978-3-030-32489-6_13. [Accessed: Oct. 27, 2023]

[13] "Ptrace." [Online]. Available: https://man7.org/linux/man-pages/man2/ptrace.2.html. [Accessed: Aug. 24, 2023]

[14] A. B. Madabhushana, "Configure Linux system auditing with auditd," Oct. 26, 2021. [Online]. Available: https://www.redhat.com/sysadmin/configure-linux-auditing-auditd. [Accessed: Jan. 21, 2024]

[15] "BPF Documentation." [Online]. Available: https://docs.kernel.org/bpf/index.html. [Accessed: Aug. 24, 2023]

[16] J. Keniston, P. S. Panchamukhi, and M. Hiramatsu, "Kernel Probes (Kprobes)." [Online]. Available: https://www.kernel.org/doc/html/latest/trace/kprobes.html. [Accessed: Aug. 24, 2023]

[17] "Event Tracing - Win32 apps," Jan. 07, 2021. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/etw/event-tracing-portal. [Accessed: Aug. 23, 2023]

[18] "FUSE." [Online]. Available: https://www.kernel.org/doc/html/latest/filesystems/fuse.html. [Accessed: Aug. 24, 2023]

[19] Gooch, "Overview of the Linux Virtual File System." [Online]. Available: https://docs.kernel.org/filesystems/vfs.html. [Accessed: Aug. 24, 2023]

[20] R. Bartlett *et al.*, "xSDK Foundations: Toward an Extreme-scale Scientific Software Development Kit," *Supercomputing Frontiers and Innovations*, vol. 4, no. 1, 1, pp. 69–82, Feb. 2017, doi: 10.14529/jsfi170104. [Online]. Available: https://superfri.org/index.php/superfri/article/view/127. [Accessed: Aug. 31, 2023]

[21] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert, "Randomized algorithms for the low-rank approximation of matrices," *Proceedings of the National Academy of Sciences*, vol. 104, no. 51, pp. 20167–20172, Dec. 2007, doi: 10.1073/pnas.0709640104. [Online]. Available: https://www.pnas.org/doi/full/10.1073/pnas.0709640104. [Accessed: Jan. 25, 2024]

[22] D. Beyer, S. Löwe, and P. Wendler, "Reliable benchmarking: Requirements and solutions," *Int J Softw Tools Technol Transfer*, vol. 21, no. 1, pp. 1–29, Feb. 2019, doi: 10.1007/s10009-017-0469-y. [Online]. Available: https://link.springer.com/10.1007/s10009-017-0469-y. [Accessed: Jun. 30, 2022]

[23] A. Gehani and D. Tariq, "SPADE: Support for Provenance Auditing in Distributed Environments," in *Middleware 2012*, 2012, pp. 101–120, doi: 10.1007/978-3-642-35170-9_6.

[24] N. Balakrishnan, T. Bytheway, R. Sohan, and A. Hopper, "{OPUS}: A Lightweight System for Observational Provenance in User Space," presented at the 5th USENIX Workshop on the Theory and Practice of Provenance (TaPP 13), 2013 [Online]. Available: https://www.usenix.org/conference/tapp13/technical-sessions/presentation/balakrishnan. [Accessed: Jul. 06, 2023]

[25] S. Sultana and E. Bertino, "A file provenance system," in *Proceedings of the third ACM conference on Data and application security and privacy*, Feb. 2013, pp. 153–156, doi: 10.1145/2435349.2435368 [Online]. Available: https://dl.acm.org/doi/10.1145/2435349.2435368. [Accessed: Aug. 23, 2023]

[26] Y. Ji, S. Lee, and W. Lee, "RecProv: Towards Provenance-Aware User Space Record and Replay," in *Provenance and Annotation of Data and Processes*, 2016, pp. 3–15, doi: 10.1007/978-3-319-40593-3_1.

[27] D. Fadolalkarim, A. Sallam, and E. Bertino, "PANDDE: Provenance-based ANomaly Detection of Data Exfiltration," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, Mar. 2016, pp. 267–276, doi: 10.1145/2857705.2857710 [Online]. Available: https://dl.acm.org/doi/10.1145/2857705.2857710. [Accessed: Aug. 24, 2023]

[28] L. Rupprecht, J. C. Davis, C. Arnold, Y. Gur, and D. Bhagwat, "Improving reproducibility of data science pipelines through transparent provenance capture," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3354–3368, Aug. 2020, doi: 10.14778/3415478.3415556. [Online]. Available: https://dl.acm.org/doi/10.14778/3415478.3415556. [Accessed: Aug. 24, 2023]

[29] A. Vahdat and T. Anderson, "Transparent result caching," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, Jun. 1998, p. 3.

[30] "About DTrace." [Online]. Available: http://dtrace.org/blogs/about/. [Accessed: Aug. 23, 2023]

[31] markruss, "Sysmon - Sysinternals," Apr. 11, 2023. [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon. [Accessed: Aug. 23, 2023]

[32] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu, "Accurate, Low Cost and Instrumentation-Free Security Audit Logging for Windows," in *Proceedings of the 31st Annual Computer Security Applications Conference*, Dec. 2015, pp. 401–410, doi: 10.1145/2818000.2818039 [Online]. Available: https://dl.acm.org/doi/10.1145/2818000.2818039. [Accessed: Aug. 23, 2023]

[33] K. H. Lee, X. Zhang, and D. Xu, "High Accuracy Attack Provenance via Binary-based Execution Partition," in *Proceedings of the 2017 Network and Distributed System Security (NDSS) Symposium*, 2017.

[34] V. P. Kemerlis, G. Portokalidis, K. Jee, and A. D. Keromytis, "Libdft: Practical dynamic data flow tracking for commodity systems," in *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments*, Mar. 2012, pp. 121–132, doi: 10.1145/2151024.2151042 [Online]. Available: https://dl.acm.org/doi/10.1145/2151024.2151042. [Accessed: Aug. 23, 2023]

[35] Y. 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*, Oct. 2017, pp. 377–390, doi: 10.1145/3133956.3134045 [Online]. Available: https://dl.acm.org/doi/10.1145/3133956.3134045. [Accessed: Aug. 23, 2023]

[36] M. Stamatogiannakis, P. Groth, and H. Bos, "Looking Inside the Black-Box: Capturing Data Provenance Using Dynamic Instrumentation," in *Provenance and Annotation of Data and Processes*, 2015, pp. 155–167, doi: 10.1007/978-3-319-16462-5_12.

[37] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "{MPI}: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning," presented at the 26th USENIX Security Symposium (USENIX Security 17), 2017, pp. 1111–1128 [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/ma. [Accessed: Aug. 23, 2023]

[38] Y. 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*, Mar. 2016, pp. 503–515, doi: 10.1145/2872362.2872395 [Online]. Available: https://dl.acm.org/doi/10.1145/2872362.2872395. [Accessed: Aug. 23, 2023]

[39] C. H. Suen, R. K. L. Ko, Y. S. Tan, P. Jagadpramana, and B. S. Lee, "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*, Jul. 2013, pp. 594–602, doi: 10.1109/TrustCom.2013.73.

[40] S. Ma, X. Zhang, and D. Xu, "ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting," in *Proceedings 2016 Network and Distributed System Security Symposium*, 2016, doi: 10.14722/ndss.2016.23350 [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/protracer-towards-practical-provenance-tracing-alternating-logging-tainting.pdf. [Accessed: Aug. 23, 2023]

[41] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-Fi: Collecting high-fidelity whole-system provenance," in *Proceedings of the 28th Annual Computer Security Applications Conference*, Dec. 2012, pp. 259–268, doi: 10.1145/2420950.2420989 [Online]. Available: https://dl.acm.org/doi/10.1145/2420950.2420989. [Accessed: Aug. 23, 2023]

[42] C. Sar and P. Cao, "Lineage File System" [Online]. Available: http://crypto.stanford.edu/~cao/lineage.html. [Accessed: Aug. 23, 2023]

[43] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-Aware Storage Systems," in *2006 USENIX Annual Technical Conference*, 2006 [Online]. Available: https://dash.harvard.edu/handle/1/23853812

[44] Y. Ji *et al.*, "Enabling Refinable {Cross-Host} Attack Investigation with Efficient Data Flow Tagging and Tracking," presented at the 27th USENIX Security Symposium (USENIX Security 18), 2018, pp. 1705–1722 [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/jia-yang. [Accessed: Aug. 23, 2023]

[45] A. Bates, D. (Jing). Tian, K. R. B. Butler, and T. Moyer, "Trustworthy {Whole-System} Provenance for the Linux Kernel," presented at the 24th USENIX Security Symposium (USENIX Security 15), 2015, pp. 319–334 [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/bates. [Accessed: Aug. 25, 2023]

[46] T. Pasquier *et al.*, "Practical whole-system provenance capture," in *Proceedings of the 2017 Symposium on Cloud Computing*, Sep. 2017, pp. 405–418, doi: 10.1145/3127479.3129249 [Online]. Available: https://dl.acm.org/doi/10.1145/3127479.3129249. [Accessed: Jul. 07, 2023]

[47] F. Wang, Y. Kwon, S. Ma, X. Zhang, and D. Xu, "Lprov: Practical Library-aware Provenance Tracing," in *Proceedings of the 34th Annual Computer Security Applications Conference*, Dec. 2018, pp. 605–617, doi: 10.1145/3274694.3274751 [Online]. Available: https://dl.acm.org/doi/10.1145/3274694.3274751. [Accessed: Aug. 24, 2023]

[48] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: Capturing system-wide information flow for malware detection and analysis," in *Proceedings of the 14th ACM conference on Computer and communications security*, Oct. 2007, pp. 116–127, doi: 10.1145/1315245.1315261 [Online]. Available: https://dl.acm.org/doi/10.1145/1315245.1315261. [Accessed: Aug. 23, 2023]

[49] M. Stamatogiannakis, P. Groth, and H. Bos, "Decoupling provenance capture and analysis from execution," in *Proceedings of the 7th USENIX Conference on Theory and Practice of Provenance*, Jul. 2015, p. 3.

[50] J. Frew and P. Slaughter, "ES3: A Demonstration of Transparent Provenance for Scientific Computation," in *Provenance and Annotation of Data and Processes*, 2008, pp. 200–207, doi: 10.1007/978-3-540-89965-5_21.

[51] I. Foster, J. Vockler, M. Wilde, and Y. Zhao, "Chimera: A virtual data system for representing, querying, and automating data derivation," in *Proceedings 14th International Conference on Scientific and Statistical Database Management*, Jul. 2002, pp. 37–46, doi: 10.1109/SSDM.2002.1029704 [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1029704. [Accessed: Jan. 21, 2024]

[52] J. Thalheim, P. Bhatotia, and C. Fetzer, "Inspector: A Data Provenance Library for Multithreaded Programs," May 02, 2016. [Online]. Available: http://arxiv.org/abs/1605.00498. [Accessed: Jan. 21, 2024]

[53] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis," *Network and Distributed System Security Symposium*, Jan. 2020 [Online]. Available: https://par.nsf.gov/biblio/10146531-omegalog-high-fidelity-attack-investigation-via-transparent-multi-layer-log-analysis. [Accessed: Aug. 23, 2023]

[54] K. H. Lee, X. Zhang, and D. Xu, "LogGC: Garbage collecting audit log," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, Nov. 2013, pp. 1005–1016, doi: 10.1145/2508859.2516731 [Online]. Available: https://dl.acm.org/doi/10.1145/2508859.2516731. [Accessed: Jan. 21, 2024]

[55] R. Yang, S. Ma, H. Xu, X. Zhang, and Y. Chen, "UISCOPE: Accurate, Instrumentation-free, and Visible Attack Investigation for GUI Applications," in *Proceedings 2020 Network and Distributed System Security Symposium*, 2020, doi: 10.14722/ndss.2020.24329 [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2020/02/24329.pdf. [Accessed: Aug. 23, 2023]

[56] W. U. Hassan, M. Lemay, N. Aguse, A. Bates, and T. Moyer, "Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs," in *Proceedings 2018 Network and Distributed System Security Symposium*, 2018, doi: 10.14722/ndss.2018.23141 [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_07B-1_Hassan_paper.pdf. [Accessed: Aug. 23, 2023]

[57] D. H. Ton That, G. Fils, Z. Yuan, and T. Malik, "Sciunits: Reusable Research Objects," in *2017 IEEE 13th International Conference on e-Science (e-Science)*, Oct. 2017, pp. 374–383, doi: 10.1109/eScience.2017.51.

[58] Y. Kwon *et al.*, "MCI : Modeling-based Causality Inference in Audit Logging for Attack Investigation," in *Proceedings 2018 Network and Distributed System Security Symposium*, 2018, doi: 10.14722/ndss.2018.23306 [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_07B-2_Kwon_paper.pdf. [Accessed: Aug. 23, 2023]