

Accurate Path Prediction of Provenance Traces

Raza Ahmad
DePaul University
Chicago, Illinois, USA
sra0nasir@gmail.com

Yuta Nakamura
DePaul University
Chicago, Illinois, USA
ynakamu1@depaul.edu

Hee Young Jung
The University of Chicago
Chicago, Illinois, USA
hyeong@uchicago.edu

Tanu Malik
DePaul University
Chicago, Illinois, USA
tanu.malik@depaul.edu

Abstract

Several security and workflow applications require provenance information at the operating system level for diagnostics. The resulting provenance traces are often more informative if they are efficiently mapped to execution paths within the control flow graph. However, current provenance systems do not map traces to control flow graphs for diagnostics purposes due to the computational complexity of mapping traces to graphs. We formulate the path prediction problem for provenance traces and take a machine learning approach to solve the problem. We develop a transformer-based graph convolutional network to predict paths. Our experiments demonstrate that our machine learning model achieves more than twice the accuracy on average compared to simple probabilistic models, with an increased computation time trade-off.

CCS Concepts

• Information systems → Information integration; • Computing methodologies → Structured outputs.

Keywords

data provenance, systems, path prediction, machine learning, graph-sage, transformers

ACM Reference Format:

Raza Ahmad, Hee Young Jung, Yuta Nakamura, and Tanu Malik. 2024. Accurate Path Prediction of Provenance Traces. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24)*, October 21–25, 2024, Boise, ID, USA. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3627673.3679872>

1 Introduction

Data provenance is a record of the origin and evolution of data within a program or application. It is vital for establishing reproducibility, identifying security breaches, and conducting diagnostics. Audited provenance, however, often needs to incorporate additional

sources such as program specifications [22], workflow specifications [3, 18], or knowledge graphs [27] to improve the quality and explanation of lineage results. Consider the provenance graph shown in Figure 1(a) and its audited provenance trace in Figure 1(b). While the trace events can be mapped to individual nodes of the graph, including the program specification as in Figure 1(c) explains the specific path taken by the trace and *why* a write appeared in it.

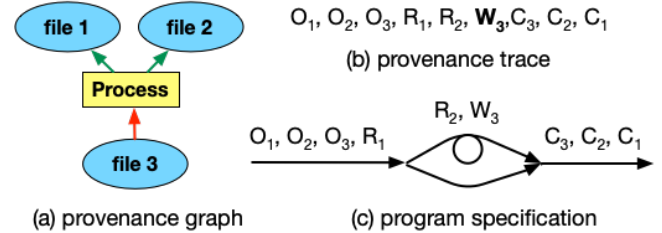


Figure 1: Explanation of provenance traces improves with program specifications.

While the program specification improves causal explanations, current provenance systems [2, 10, 24] often do not include such sources to interpret traces because of the computational complexity required to map a provenance trace onto a directed graph which generates that trace. Typically, provenance traces do not have unique labels, and mapping these labels onto a directed graph with non-unique node labels requires maintaining an exponential state, as illustrated in Figure 2. From a graph theoretical perspective, this is similar to the path explosion problem for which several heuristics are available [4, 15]. For example, in RWset [4], the program stops exploring the path as soon as the suffix matches the suffix of a previously explored path. Other heuristics rely on graph pre-processing to reduce complexity to cubic levels [15].

In this paper, we address the path explosion problem using a machine learning approach. We develop a transformer-based graph convolutional network (GCN) that predicts a path within a specification for a given provenance trace. This model takes both trace sequences and program specification graphs as its input to generate the predicted path. The Transformer [29] encoder utilizes the self-attention mechanism to effectively capture the sequence representation of the provenance trace. The graph convolutional network utilizes the graph structure to generate efficient embeddings to represent graph nodes and edges, which are then used to predict the the most likely path as the model output¹.

¹The code for this model can be found at <https://github.com/depaul-dice/MLProvDiff>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '24, October 21–25, 2024, Boise, ID, USA.

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0436-9/24/10
<https://doi.org/10.1145/3627673.3679872>

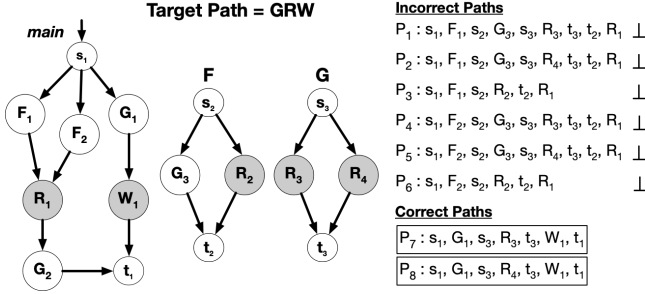


Figure 2: The path explosion occurs due to large state maintenance. We consider a program consisting of three functions *main*, *F*, and *G*, and two system calls *R* and *W*. Each function is a directed control flow graph. Different occurrences of the same function or system call are subscripted. To find the path *G, R, W*, we must traverse 6 paths in the control flow graphs before finding the corresponding paths. If the functions have control flow loops, the number of paths become countably infinite. Shaded nodes represent system calls.

Our transformer-based GCN utilizes the Transformer [29] model to encode traces and the GraphSAGE model [11] to learn low-dimensional vector embeddings for nodes and edges in large graphs. GraphSAGE is chosen over other machine learning models because it learns from the graph structure and node adjacency in a non-sequential manner by sampling nodes from each node's neighborhood and aggregating their information to predict node representations. Our combined model uses a dot product of the two encodings to determine which path has a higher probability of occurrence. Additionally, we adapt two existing models for path prediction: the most likely path based on frequencies and the n -gram language model. Our results show approximately 41% and 340% improvement on average over the most likely path model and the n -gram model, respectively.

To the best of our knowledge, this is the first paper to use provenance traces and program specifications for path prediction. We plan to demonstrate the impact of the path prediction problem on differential analysis in our future work.

2 Problem Statement

The path prediction problem is to find a path in a program specification that aligns with the labels in a provenance trace. Formally, given a labeled trace $T = \{l_1, l_2, \dots, l_m\}$, and a program specification $S = G(V, E)$, the path prediction problem is to find a path $P = (v_1, v_2, \dots, v_n)$ such that

$$\begin{aligned} v_i &\in G \wedge l_j \in T, \\ \text{label}(v_i) &= l_j, \text{ and} \\ \text{length}(T) &= \text{length}(P) \end{aligned}$$

Operating system-level provenance traces can consist of millions of events, each containing a rich set of meta-information. As per Kwon et al. [16], we exclusively consider the events as system and function call names and ignore the arguments such as process names and return values. While these arguments provide valuable data that have enabled other machine learning models [8, 9], in

our case, they often correspond to dynamic information that is not present in static program specifications. Therefore, they do not aid in solving the path prediction problem.

The program specification, in general, is obtained from the source code of a program. We assume the specification comprises a set of directed graphs in which each directed graph is a control flow graph of a distinct function element that appears in the program, with one graph corresponding to the entry function “main” of the program. Thus, each graph ‘belongs’ to a function and its nodes ‘call’ other functions, system calls, and special instructions including itself. Such program specifications can be easily generated from LLVM-IR [17]. Figure 2 shows a program specification with three functions: *main*, *F*, and *G*, along with two system calls *R* and *W*.

3 Path Prediction

3.1 Most Likely Path (MLP) Model

The most likely path with backtracking (MLP) model traverses the graph in depth-first order and selects the node that matches the label in the path. However, when a node has two children with the same label as the next label in the trace, the model makes a decision based on the frequency of node transitions. Figure 3 shows an example.

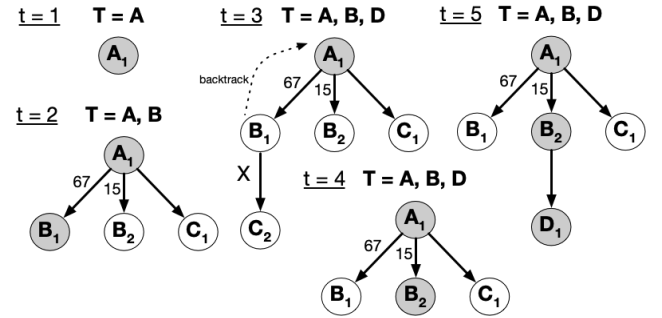


Figure 3: Path prediction with the MLP model shows the shaded nodes representing the predicted path at each time step. Initially, B_1 is temporarily selected as the next node, but then the model backtracks to the parent of B_1 , and chooses B_2 to find the path with the highest probability that maintains consistent label progression with the trace.

3.2 n -gram Model

A simple model akin to the n -gram model used in natural language processing is employed to predict the next node in a path based on a fixed size window of the previous n nodes. In our scenario, we train the n -gram model using known, valid paths obtained from the program specification corresponding to the traces. Prediction for the next node v_i in the path relies on the probability $P(v_i | v_{i-1}, \dots, v_{n-1})$, where n represents path length in the training data.

3.3 Transformer-based Graph Convolutional Network Model

Expanding on the n -gram model, a sequence-to-sequence model (Seq2seq) generates a sequence as output based on another sequence provided as the model input. The Transformer model [29]

is a state-of-the-art architecture in sequence-to-sequence learning which excels in capturing long-range dependencies within the input. Experiments have demonstrated the efficacy of Transformers for processing large sequences. For instance, the token lengths in models like GPT-2 [26] and GPT-4 [1] are 1024 and 8192, respectively. In our dataset, the maximum path length is 996 with an average path length of 177. However, sequences lack inherent knowledge about the structure of graphs. To address this limitation and incorporate graph structure, we augment sequences with a graph convolutional network model constructed from program specification graphs.

Figure 4 shows the transformer-based graph convolutional network. Each node of $G(V, E)$ is represented using a one-hot vector encoding of fixed size d calculated from the label and type of that node. Since a trace corresponds to labels of nodes in the graph, each trace is identified by a matrix of one-hot encoded vectors and is given as input to the Transformer. We only use the encoder layer of the Transformer which creates representations for the list of traces and its output is given as follows:

$$B = \text{Transformer}(T) \quad (1)$$

where B is a matrix of size $M \times d$, M is the size of the largest trace, and d is the number of feature dimensions for each node.

The directed graph of the program specification is modeled via GraphSage which learns vector representations of nodes and their neighborhoods in a graph. The nodes are presented as a matrix of one-hot encoded vectors and the edges are given as two lists of source node and destination node identifiers. Given $G(V, E)$, the GraphSAGE model generates representations as:

$$A = \text{GraphSAGE}(V, E) \quad (2)$$

where A is a matrix of size $N \times d$, in which N is the total number of nodes in G and d is the number of feature dimensions of each node. Thus each row in A represents a node $v \in V$. The node representations of traces and the graph are fed to a combined model which predicts path vectors as its output, as shown in Figure 4. These node representations are updated and improved through backpropagation during the training of the model. The task of predicting the correct path vector is recognized by the model as a multi-class classification problem where the number of classes is equal to the total number of nodes in the graph. The following equations represent the inputs and outputs of the model.

$$P = BA^T \quad (3)$$

where matrices A and B represent nodes in the graph and the traces in our dataset, respectively, and P is the matrix of path predictions of size $M \times N$ such that the row $p_j \in P$ gives us the probabilities for each $v \in V$ of occurring at the j^{th} position in the path. We compute the path predicted by the model for a trace $t \in T$ by finding the node having maximum probability at position j in the path vector:

$$\hat{p} = \text{argmax}(p_j \in P), \forall j \leq M \quad (4)$$

where $\hat{p}$ is an M -dimensional vector. Given a target path p and the path $\hat{p}$ predicted by the model, we define the cross-entropy loss function of the model for one trace in the dataset as:

$$L(p, \hat{p}) = - \sum_i^N p_i \log(\hat{p}_i) \quad (5)$$

Table 1: Graph sizes (nodes and edges) and trace lengths (min, max, and avg) for six Linux utilities.

Program	Nodes	Edges	Min	Max	Avg
cat	71,698	100,796	26	226	85
uniq	150,376	216,510	52	666	138
chown	304,169	495,517	88	975	257
rm	758,421	1,215,916	22	728	101
date	1,362,766	1,982,002	56	996	317
sort	1,444,557	2,319,361	68	822	166
AVG	681,998	1,055,017	52	736	177

where N is the total number of nodes in the graphs and corresponds to the number of classes used in the multi-class cross-entropy loss calculation. The output of the model is the predicted path and is represented by a vector of numeric node identifiers in the graph.

4 Experiments

We conduct separate experiments on datasets generated by six Linux utilities: cat, uniq, chown, rm, date, and sort. We generate their control flow graphs using LLVM-IR [17]. All graphs have the same source and destination node for a given program and contains loops. Table 1 shows the number of nodes and edges for each program. Each node is identified by its unique identifier, label, and type (source/destination/internal). It is represented by a one-hot encoding using type and label. Each edge in the graph is represented in the form of pair of node identifiers. To generate traces, we sample one thousand paths from each graph by doing a random walk of the graph and obtain the corresponding traces. Each path corresponds to a valid execution path of the program, which may include loops, such that it starts from the main function and ends at the return instruction. For each program, paths are of variable lengths, as shown in Table 1. Generating flat traces using the dynamic binary instrumentation framework Intel Pin [19] such that they map to program specifications is beyond the scope of this work [21].

All three models were run on an Ubuntu 20.04 machine having 1TB of RAM and 85GB of GPU Memory. For the n -gram model, we used the value of n as 5, 10, and 15, and report their average accuracy. In Tr-GCN, the transformer network had 12 nodes in the hidden layer and 6 attention heads. The Tr-GCN experiments were performed on a batch size of 2 with a learning rate of 0.001, and were trained over 100 epochs with an 80-20 train-test split and a dropout rate of 10%. Finally, the accuracy metric is computed by first performing element-wise comparison between the actual and predicted path and then counting the nodes present in their correct positions in the predicted path. Given the set of predicted paths $\hat{P}$ and the corresponding set of actual paths P , we define it as follows $\forall p_j \in P$ and $\forall \hat{p}_j \in \hat{P}$:

$$\text{acc} = \sum_j^{|P|} \frac{\text{acc}_j}{|p_j|}, \quad \text{acc}_j = \sum_k \begin{cases} 1 & \hat{p}_{jk} = p_{jk} \\ 0 & \hat{p}_{jk} \neq p_{jk} \end{cases} \quad (6)$$

Table 2 shows the results of accuracy and time for all Linux commands across the three models. The Tr-GCN model outperforms both techniques by a large margin. On average, the Tr-GCN model has an accuracy of 83% on the test data, compared to the average

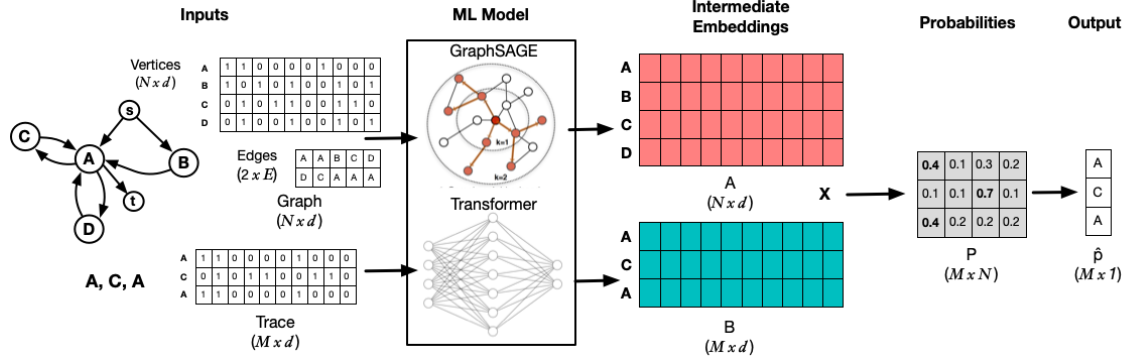


Figure 4: The Tr-GCN model architecture demonstrating an iteration for one provenance trace as input. The model creates intermediate embeddings which are then used to generate probabilities over all nodes for occurring in the path. The nodes with highest probabilities of occurrence at each position are selected to form the model output path.

Table 2: Accuracy and time results for the three models. The time for all three methods is in seconds.

Prog	Accuracy			Time		
	Tr-GCN	n-gram	MLP	Tr-GCN	n-gram	MLP
cat	82%	67%	30%	928	0.25	0.46
uniq	86%	65%	34%	1,837	0.39	0.78
chown	79%	56%	12%	3,751	0.75	2.01
rm	91%	47%	19%	8,215	0.30	2.43
date	74%	49%	8%	15,803	0.99	4.91
sort	88%	68%	11%	15,842	0.48	5.10
AVG	83%	59%	19%	2.15	0.53	2.61

Table 3: Results by training and testing on different Linux utilities demonstrate the generalizability of Tr-GCN model.

Train	Test	Time (h)	Accuracy
cat	uniq	0.36	82%
uniq	cat	0.78	72%
chown	cat,uniq	1.81	77%
cat,uniq	chown	1.37	74%
chown	rm	2.01	90%
rm	chown	3.90	76%

accuracy of 19% and 59% for MLP and n -gram, respectively, which is more than twice their combined average accuracy of 39%. The n -gram model performs significantly better than the MLP model because it predicts the next node in the path based on several previous nodes, while the MLP model predicts the next node based only on the current node. It is important to note that the accuracy of both conventional models decreases as the graph size increases due to their inability to correctly model larger node sequences in those graphs. We also investigate the generalizability of our model by training and testing on different programs. The results shown in Table 3 demonstrate that our model is general enough to accurately map provenance traces to their paths in a graph which corresponds to an entirely unseen set of functions calls in the test program.

5 Related Work

Control flow graphs determine the order in which individual function calls of an imperative program are executed or evaluated. Alternatively, data flow analysis determines the possible set of values

calculated at various points in a computer program. Most provenance systems [5, 12, 23, 25] semantically represent data flows. Provenance systems that operate at the operating system level often need control flow graphs since they capture data flow of black-box processes. The closest work in this regard is [28] but requires bitcode representation of the programs. We instead work with raw provenance traces without necessitating bitcode representation of the programs. Raw system call traces are often represented as vectors of system calls and their attributes in machine learning models that detect outliers and anomalies [8, 9, 13, 30]. Our work differs in that we use trace data of function and system calls to predict the correct path for a given trace. Trace analysis using machine learning has largely employed RNNs and LSTMs which are particularly suited for longer sequences [6, 7, 14, 31]. For instance, Lv et al. [20] perform intrusion detection using Seq2seq modeling to predict which path will be executed in the future by an intruder. We combine learning over both sequences and control flow graphs.

6 Conclusion

We have presented a machine learning architecture for predicting paths corresponding to provenance traces. Our model shows higher accuracy though at a cost of computation time. Addressing the computation requirements via parallelism and showing how this model can empower applications such as differential analysis, reproducibility, and intrusion detection, is part of our ongoing work. In future we aim to (i) improve accuracy by capturing long term dependencies in control flow graphs, and (ii) improve the expressiveness of different models via ablation analysis.

Acknowledgements

This work is supported in part by NSF grant number CNS-1846418 and OAC-2411437. We would like to thank Sayan Ranu and Amitabha Bagchi for suggesting a machine learning approach to the path prediction problem. We would also like to thank anonymous reviewers for their feedback.

References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Floren-
cia Leoni Aleman, Diogo Almeida, Janko Altmenschmidt, Sam Altman, Shyamal
Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*
(2023).
- [2] Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, and Andy Hopper.
2013. OPUS: A lightweight system for observational provenance in user space. In
5th USENIX Workshop on the Theory and Practice of Provenance (TaPP 13). USENIX
Association. <https://doi.org/10.5555/2814579.2814587>
- [3] Zhuowei Bao, Sarah Cohen-Boulakia, Susan B Davidson, Anat Eyal, and Sanjeev
Khanna. 2009. Differencing provenance in scientific workflows. In *ICDE*.
- [4] Peter Boonstoppel, Cristian Cadar, and Dawson Engler. 2008. RWset: Attacking
path explosion in constraint-based test generation. In *Tools and Algorithms for
the Construction and Analysis of Systems: 14th International Conference, TACAS
2008, Held as Part of the Joint European Conferences on Theory and Practice of
Software, ETAPS 2008, Budapest, Hungary, March 29–April 6, 2008. Proceedings 14*.
Springer, 351–366. https://doi.org/10.1007/978-3-540-78800-3_27
- [5] Andrew Davison. 2012. Automated Capture of Experiment Context for Easier
Reproducibility in Computational Research. *Computing in Science & Engineering*
14, 4 (2012), 48–56. <https://doi.org/10.1109/MCSE.2012.41>
- [6] Michael Dymshits, Benjamin Myara, and David Tolpin. 2017. Process monitor-
ing on sequences of system call count vectors. In *2017 International Camahan
Conference on Security Technology (ICCST)*. 1–5. <https://doi.org/10.1109/CCST.2017.8167792>
- [7] Okwudili M. Ezeme, Qusay H. Mahmoud, and Akramul Azim. 2022. A framework
for anomaly detection in time-driven and event-driven processes using kernel
traces. *IEEE Transactions on Knowledge and Data Engineering* 34, 1 (2022), 1–14.
<https://doi.org/10.1109/TKDE.2020.2978469>
- [8] Quentin Fournier, Daniel Aloise, Seyed Vahid Azhari, and François Tetreault.
2021. On improving deep learning trace analysis with system call arguments.
In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories
(MSR)*. 120–130. <https://doi.org/10.1109/MSR52588.2021.00025>
- [9] Quentin Fournier, Daniel Aloise, and Leandro R. Costa. 2023. Language Models
for Novelty Detection in System Call Traces. *arXiv preprint arXiv:2309.02206*
(2023). <https://doi.org/10.48550/arXiv.2309.02206>
- [10] Ashish Gehani and Dawood Tariq. 2012. SPADE: Support for provenance auditing
in distributed environments. In *ACM/IFIP/USENIX International Conference on
Distributed Systems Platforms and Open Distributed Processing*. Springer, 101–120.
https://doi.org/10.1007/978-3-642-35170-9_6
- [11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation
learning on large graphs. *Advances in neural information processing systems* 30
(2017), 1025–1035. <https://doi.org/10.5555/3294771.3294869>
- [12] Mohammad Rezwani Huq, Peter M. G. Apers, and Andreas Wombacher. 2013.
An Inference-Based Framework to Manage Data Provenance in Geoscience Ap-
plications. *IEEE Transactions on Geoscience and Remote Sensing* 51, 11 (2013),
5113–5130. <https://doi.org/10.1109/TGRS.2013.2247769>
- [13] Rupesh Raj Karn, Prabhakar Kudva, Hai Huang, Sahil Suneja, and Ibrahim M.
Elfadel. 2021. Cryptomining detection in container clouds using system calls
and explainable machine learning. *IEEE Transactions on Parallel and Distributed
Systems* 32, 3 (2021), 674–691. <https://doi.org/10.1109/TPDS.2020.3029088>
- [14] Gyuwan Kim, Hayoon Yi, Jangho Lee, Yunheung Paek, and Sungroh Yoon.
2016. LSTM-Based System-Call Language Modeling and Robust Ensemble
Method for Designing Host-Based Intrusion Detection Systems. *ArXiv preprint
abs/1611.01726* (2016). <https://doi.org/10.48550/arXiv.1611.01726>
- [15] Saparya Krishnamoorthy, S Michael Hsiao, and et. al. 2010. Tackling the path
explosion problem in symbolic execution-driven test generation for programs.
In *2010 19th IEEE Asian Test Symposium*. IEEE, 59–64. <https://doi.org/10.1109/ATS.2010.19>
- [16] Yonghui Kwon, Fei Wang, Weihang Wang, Kyu Hyung Lee, Wen-Chuan Lee,
Shiqing Ma, Xiangyu Zhang, Dongyan Xu, Somesh Jha, Gabriela F. Ciocarlie,
Ashish Gehani, and Vinod Yegneswaran. 2018. MCI : Modeling-based causality
inference in audit logging for attack investigation. In *25th Annual Network and
Distributed System Security Symposium, NDSS 2018, San Diego, California, USA,
February 18–21, 2018*. The Internet Society. <https://doi.org/10.14722/ndss.2018.23312>
- [17] Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for
lifelong program analysis and transformation. In *CGO*. 75–88.
- [18] Chunhyeok Lim, Shiyong Lu, and et. al. 2010. Prospective and retrospective prove-
nance collection in scientific workflow environments. In *2010 IEEE International
Conference on Services Computing*. IEEE, 449–456.
- [19] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff
Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin:
Building customized program analysis tools with dynamic instrumentation. In
*Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language
Design and Implementation (PLDI '05)*. Association for Computing Machinery,
190–200. <https://doi.org/10.1145/1065010.1065034>
- [20] Shaohua Lv, Jian Wang, Yinqi Yang, and Jiqiang Liu. 2018. Intrusion prediction
With system-call sequence-to-sequence model. *IEEE Access* 6 (2018), 71413–71421.
<https://doi.org/10.1109/ACCESS.2018.2881561>
- [21] Yuta Nakamura, Iyad Kanj, and Tanu Malik. 2023. Efficient Differencing of
System-level Provenance Graphs. In *Proceedings of the 32nd ACM International
Conference on Information and Knowledge Management (Birmingham, United
Kingdom) (CIKM '23)*. Association for Computing Machinery, New York, NY,
USA, 4220–4223. <https://doi.org/10.1145/3583780.3615171>
- [22] Yuta Nakamura, Tanu Malik, Iyad Kanj, and Ashish Gehani. 2022. Provenance-
based Workflow Diagnostics Using Program Specification. In *2022 IEEE 29th
International Conference on High Performance Computing, Data, and Analytics
(HiPC)*. 292–301. <https://doi.org/10.1109/HiPC56025.2022.00046>
- [23] Nikolaus Nova Parulian, Timothy M. McPhillips, and Bertram Ludascher. 2021. A
Model and System for Querying Provenance from Data Cleaning Workflows. In
Provenance and Annotation of Data and Processes, Boris Glavic, Vanessa Bragan-
holo, and David Koop (Eds.). Springer International Publishing, Cham, 183–197.
https://doi.org/10.1007/978-3-030-80960-7_11
- [24] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Evers,
Margo Seltzer, and Jean Bacon. 2017. Practical whole-system provenance capture.
In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 405–418. <https://doi.org/10.1145/3127479.3129249>
- [25] João Felipe Pimentel, Juliana Freire, Leonardo Murta, and Vanessa Braganholo.
2019. A Survey on Collecting, Managing, and Analyzing Provenance from Scripts.
ACM Comput. Surv. 52, 3, Article 47 (jun 2019), 38 pages. <https://doi.org/10.1145/3311955>
- [26] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya
Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI
blog* (2019).
- [27] Leslie F Sikos and Dean Philp. 2020. Provenance-aware knowledge representation:
A survey of data models and contextualized knowledge graphs. *Data Science and
Engineering* 5 (2020), 293–316.
- [28] Dawood Tariq, Maisem Ali, and Ashish Gehani. 2012. Towards automated col-
lection of application-level data provenance. In *Proceedings of the 4th USENIX
Conference on Theory and Practice of Provenance (Boston, MA) (TaPP'12)*. USENIX
Association, USA, 16. <https://doi.org/10.5555/2342875.2342891>
- [29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones,
Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you
need. *Advances in neural information processing systems* 30 (2017), 5998–6008.
- [30] Sarah Wunderlich, Markus Ring, Dieter Landes, and Andreas Hotho. 2020. Com-
parison of system call representations for intrusion detection. In *International
Joint Conference: 12th International Conference on Computational Intelligence in Se-
curity for Information Systems (CISIS 2019) and 10th International Conference on EU-
ropean Transnational Education (ICEUTE 2019) Seville, Spain, May 13th–15th, 2019
Proceedings 12*. Springer, 14–24. https://doi.org/10.1007/978-3-030-20005-3_2
- [31] Nezhir Yigitbasi, Matthieu Gallet, Derrick Kondo, Alexandru Iosup, and Dick
Epema. 2010. Analysis and modeling of time-correlated failures in large-scale
distributed systems. In *2010 11th IEEE/ACM International Conference on Grid
Computing*. 65–72. <https://doi.org/10.1109/GRID.2010.5697961>