# Flurry: A Fast Framework for Provenance Graph Generation for Representation Learning

Maya Kapoor
mkapoor1@uncc.edu
UNC Charlotte
North Carolina, USA

Joshua Melton
jmelto30@uncc.edu
UNC Charlotte
North Carolina, USA

Michael Ridenhour
mridenh7@uncc.edu
UNC Charlotte
North Carolina, USA

Thomas Moyer
tmoyer2@uncc.edu
UNC Charlotte
North Carolina, USA

Siddharth Krishnan
skrishnan@uncc.edu
UNC Charlotte
North Carolina, USA

## ABSTRACT

Representation learning and deep learning on data provenance graphs has yielded insightful new angles for intrusion detection in cybersecurity systems and is rapidly expanding as a research topic in the scientific community. In order to train the learning models, system execution data must be created and captured which represent realistic cyberattacks of a wide variety. Furthermore, these graphs must contain equally authentic benign workflows relative to the host and its multi-faceted processes. In order to support dynamic generation of provenance graphs to rapidly train new provenance graph learning models, we present Flurry, an end-to-end framework which simulates attack and benign activity and generates provenance graphs in multiple formats exportable to graph learning systems. In this demonstration, we showcase Flurry's ability to simulate both pre-configured and user-defined cyberattacks as well as benign behavior, and convert these captures into provenance graphs. We investigate the spectral properties of these graphs and perform attack classification experiments comparing three graph neural network models. Our results are comparable to those in the original learning models' research papers, showing that the Flurry graphs provide an ideal baseline and an extensible framework for graph representation learning on provenance graphs. In our demo, we show that Flurry will bring brand-new expandability and proficiency to the provenance graph learning community's available data.

## CCS CONCEPTS

• **Security and privacy** → *Intrusion detection systems*; • **Information systems** → *Graph-based database models*; **Extraction, transformation and loading**; • **Computing methodologies** → *Knowledge representation and reasoning*; • **Social and professional topics** → **System management**; • **General and reference** → **Experimentation**; **Verification**.

## KEYWORDS

datasets, cybersecurity, data provenance, graph representation learning

**ACM Reference Format:**
Maya Kapoor, Joshua Melton, Michael Ridenhour, Thomas Moyer, and Siddharth Krishnan. 2022. Flurry: A Fast Framework for Provenance Graph Generation for Representation Learning. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3511808.3557200

## 1 INTRODUCTION

Data provenance, or the legacy of data on a system, has long been a tool for cybersecurity practitioners to forensically investigate anomalous system execution. Directed, acyclic graphs are constructed from provenance data in order to perform causality analysis. Recently, provenance graphs have risen to the forefront of representation learning [3–5, 13]. Symbiotically, provenance graphs provide a multi-layered, heterogeneous problem to graph representation learning researchers, and learning systems provide new insight and a quicker response to resilient systems and security practitioners. Thus, continuing research in provenance graph representation learning is promising for both communities as an innovative and practical way forward.

In order to make this progress, data provenance graphs are needed to train and test new models. It is a systematic process to capture system execution, log data, and interpret it into a node/edge schema. But there are several properties which make data generation a hard and more complex problem in the real world than the current public datasets [3, 4, 6, 10] represent. A threat actor will try several avenues at multiple entrypoints into a single or distributed system in order to gain a foothold. Generally, once an attacker is able to gain a single access, they may establish some backdoor which will allow for further exploitation. Thus, an attacker needs only to succeed one time at only one of the ways they may try to strike, while security professionals must be prepared to guard and adapt at all points simultaneously.

With this common attack pattern, setting up datasets which model only one style of attack is not realistic and can lead to misconclusions and ultimately a false sense of security. Attacks do not happen in isolation and there may be multiple threat actors or attacks occurring simultaneously on a real system. Additionally, there is usually a wealth of benign activity, both related and unrelated to the attack, on the system which is being threatened. Relying on static datasets is also inevitably outdated as new vulnerabilities are discovered daily. Models require a large amount of both benign and sometimes malicious data in order to learn common patterns for classification.

Because of these properties of real-world security threats, a system is direly needed which can simulate multiple attacks mixed with benign traffic, quickly expand to new and emerging threats, and generate a wealth of replicable data. FLURRY provides all these things to the provenance graph research space in a simple package usable by anyone in the community, from security specialists and data scientists.

The contributions we demonstrate that FLURRY makes to the state of the art are as follows:

- An automated framework which can quickly adapt to new forms of attack,
- An easily installable, modular system which can be applied to multiple system scenarios, including scientific workflows, high-powered computing, user systems, and user-less servers,
- Dynamic datasets which are reproducible on demand,
- 95% reduction in graph storage size from CamFlow capture to disk space via summarization and de-duplication techniques,
- Provenance capture at multiple layers of the host system with a mixture of attack types and benign behavior which more realistically represents the complex systems in use than previous datasets [3–6, 10],
- Demonstrated success in training and testing state-of-the-art provenance graph learning systems using FLURRY data,
- And a research tool for the future which facilitates new development for emerging provenance graph learning systems.

## 2 RELATED WORK

*StreamSpot* [6] is an original clustering-based anomaly detection approach on heterogeneous streaming graphs. They solely analyze browsing data they gathered and provided as a public dataset, and downloading software in an effort to identify drive-by download attacks as anomalous behavior. *Unicorn* [3] employs a version of the Weisfeiler-Lehman subtree graph kernel algorithm. They offer two static datasets which are majorly benign data and a few examples of supply chain attacks. In *PROV-GEm* [5], the authors propose a graph embedding framework based on graph convolutional networks coupled with relational self-attention to generate informative representations of provenance graphs. *ThreaTrace* [13] conducts work in node-level provenance graph representation learning. Both these works re-use the Unicorn and Streamspot datasets because of a lack of alternative datasets.

In conjunction with Unicorn, Han et al. highlight the lack of quality, publicly available provenance graph datasets in *Xanthus* [2] It introduces the idea of a distributable virtual machine image with configurable jobs for automated recreation of attack scenarios, but requires the user to orchestrate the attack and configuration to mimic real-life scenarios. This requires extensive systems knowledge and makes the framework difficult to use for other researchers focused on data science and not security.

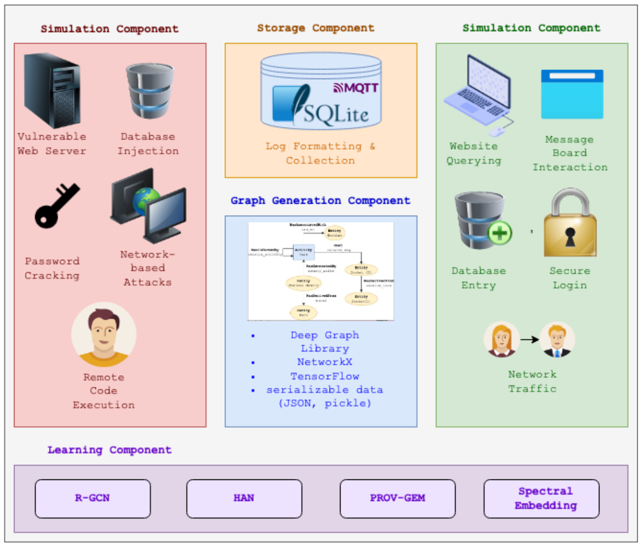

**Figure 1: System overview of components in FLURRY.**

## 3 FLURRY OVERVIEW

FLURRY bridges the gap between security and systems engineering and provenance graph representation learning by automating attacks and system behavior so as to provide the most genuine scenarios for real learning. FLURRY actively brings theoretical graph representation learning to practicality by transforming real provenance data into graphs which are storable, interpretable, and useful for learning systems. In this section, we explain the details for FLURRY's end-to-end pipeline with simplistic steps to go from the click of a button to a provenance graph.

### 3.1 Host Simulation

Under the hood, FLURRY comprises two major applications: one for host simulation and another for provenance capture and graph storage and generation. The simulation component is highly configurable in order to run different kinds of attacks as well as benign behavior. This allows the Flurry simulator to represent many kinds of computing devices, such as a server used in a scientific workflow or a user device, each of which is used for unique purposes and will experience different threats and be suited for different kinds of learning and security responses.

The host simulator is operated through a Python GUI or from the command line. As a baseline, FLURRY provides a set of attack scripts which may be selected to run. Additionally, benign behavior that corresponds to the attack scenarios or expected host behavior is also able to be configured. The researcher may organize one or more of these benign and attack behaviors in any pattern, and may execute it a provided number of times. An example configuration may be running the remote code execution attack, which simulates an attacker attempting to set up a backdoor. Then, the attacker may try some kind of data exfiltration. In order to provide some other benign behavior, the configuration could include some website querying, regular file reads and writes, and local command execution. The execution does not have to be overly complicated - it is also possible to run one simple execution of logging in to a website, for example, or multiple iterations of just that.

Figure 1 shows some of the pre-configured attacks in the red portion of the simulation component, and regular behavior in the green portion. For the web-server based attacks, Flurry deploys an instance of the Damn Vulnerable Web Application (DVWA) [1] and is able to record both client-side, browser-level provenance as well as web server activity:

- **XSS-Stored Attack:** Javascript is injected and stored into the webpage directly, causing a pop-up alert.
- **XSS-Reflected Attack:** Javascript injected into the webpage temporarily through the questionnaire, causing a pop-up alert.
- **XSS-DOM Attack:** Javascript is injected into the webpage through the URL, causing a pop-up alert.
- **Command Line Injection Attack:** A bash command (pwd) is injected and executed.
- **SQL Injection Attack:** A SQL query is injected into the submission and executed.
- **Brute Force Password Attack:** The Hydra password cracking tool is used to repeatedly enter username and password combinations.
- **SYN-Flood Attack:** Using `hping`, several thousand TCP packets are sent as fast as possible with the SYN flag set to simulate a Denial of Service attack.
- **Exfiltration Attack:** Using `hping`, a s̈ecretf̈ile is written to an ICMP packet and exported.
- **Remote Code Execution Attack:** Using `hping`, a secret backdoor is installed on the local machine and the attacker executes a script (the exfiltration attack) through the backdoor with elevated privileges.

FLURRY's host simulator also runs the following benign web and network services:

- Post to a message board,
- Complete a submission box/questionnaire,
- Query for a new webpage,
- Enter user data into a database,
- Log in to the user's account,
- Ping the machine at a normal rate,
- Write a file to `stdout`,
- Execute a script with user privileges locally.

In addition to provided attacks and benign behavior, the user can provide their own Bash or Python script for FLURRY to run. It will similarly capture this behavior and generate a graph from it using the other tools in the graph component. This unique feature allows for FLURRY to simulate experimentally specific or emerging and new cyberattacks.

While a unique and useful feature for training, the host simulation model is not the only thing FLURRY is capable of turning into provenance graphs. The capture, storage, and graph generation components of FLURRY may be installed modularly on a real system so that its authentic behavior may be recorded and transformed into graphs for learning and eventually resilient adaptation. Thus, FLURRY's use cases extend the lifetime of the model's process, from training to real-time deployment.

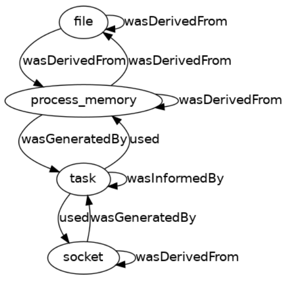

**Figure 2: Type graph of an XSS DOM Attack run on FLURRY.**

## 3.2 Provenance Graph Generation

For kernel-level data provenance, FLURRY uses CamFlow [8], a Linux kernel module which records security accesses and conveys them to a userspace daemon in W3C-PROV [11] format which can be configured to write to an MQTT topic with FLURRY as a subscriber. FLURRY also has an application-layer provenance capture element based on the `libprovenance` [8] userspace library designed by CamFlow creators. For learning purposes, capturing data at multiple layers allows for representation learning on multi-layer graphs.

| **Property** | **Options** |
|---|---|
| *Granularity* | Nodes and edges may be set to coarse (W3C-PROV, ex. `entity`, `wasDerivedFrom`) or fine (system, ex. `file`, `write`) granularity. |
| *Statistics* | A text file is generated which provides the graph ID, list of scripted behaviors run, node/edge counts, and a detailed description and count of the node types, edge types, and unique node-edge-node combinations (ex. `task-used-process_memory`). |
| *One-Hot Encodings* | A JSON-serialized file may be generated which contains a dictionary of indexed node (or edge) types and a one-hot encoded value for each node in the graph describing which type it is. |
| *Serialization* | A JSON file or gpickle may be generated which maps each edge and its source and destination node in a format which is easily transformed into a tensor or numpy arrays for uploading. |
| *Visualization* | A type graph as shown in Figure 2 may be drawn as a clear visualization of the data. |
| *Library Usage* | "CamFlow to Graph", or CF2G, Python library functions included with Flurry can also be used to directly create NetworkX [7] or Deep Graph Library [12] data structures from FLURRY Graphs. |

**Table 1: Configuration options for provenance graphs and output formats.**

From the host simulator, scripted executions are recorded with a corresponding graph ID. Our redundancy reduction and data storage strategy reduced the data size by nearly 95% in our experiments, allowing for system scalability and the ability to detect low-and-slow attack patterns over long periods of time with reduced risk of running out of memory. For real-time capture, FLURRY will also capture whole-system provenance for a configured amount of time to create ID-ed, *time-slice* graphs. In order to generate a graph, the researcher may use the command-line script to select a graph ID. The automated framework will then query the database and construct the graph with the properties selected from Table 1. The framework may also be pre-configured to generate these graphs after the attack is run.

As seen in Figure 2, we can summarize an attack execution into a node-edge-node type unique representation, or *type graph*. This diagram allows a researcher to rapidly visualize the difference between attack and benign execution cycles and see type-based anomalies. While the graph constructed for learning contains every node and edge, comparatively this graph is much more explainable for the human eye, which is further demonstrating how FLURRY is bridging the learning community and techniques with real causality analysis capability.

## 4 DEMONSTRATION

In order to demonstrate the capability of FLURRY to create provenance graphs which may actually be used for training and testing learning models, we conduct experiments using three well-cited heterogeneous graph learning frameworks: PROV-GEm [5], R-GCN [9], and HAN [14]. We generate a total of nine thousand graphs across all the attack and benign behavior described in the overview section of this paper. To conduct our experiment, we randomly sample 20% of the graphs as a test set and train the neural network models on the remaining 80% of the data. For each cyberattack dataset, we conduct five independent trials using randomly sampled test sets, and we report the average F1 micro, precision, and recall for the three GNN (Graph Neural Network) models.

In addition to the classification experiments using GNNs, we investigate the spectral properties of the generated data provenance graphs to elucidate the structural characteristics of provenance graphs generated from different attack scenarios. Any network science-based approach to data provenance, whether it employs on graph hashing, clustering, or graph neural networks, relies on the assumption that anomalous or attack data provenance will differ structurally from benign traffic—as all of these methods utilize structural information to generate representations of graphs. FLURRY's multi-layer graph construction conforms to a rigorous multiplex network definition, allowing us to generate the symmetric, positive semidefinite supra-Laplacian matrix representations of provenance graphs. From this construction, we can perform eigenvalue analysis on provenance graphs using efficient numerical methods. This analysis provides important insights into the overall connectivity and dominant layers (system operations) in benign and attack scenario provenance graphs. These spectral properties also provide important information as to why certain attack types are more easily classified by GNN methods as compared with others. For certain attack types, such as backdoor attacks, we note a greater distinction between the eigenvalue distributions for benign traffic and attack traffic. By contrast, attack types such as brute force are less distinguishable based on their spectral properties, indicating a greater structural similarity between the benign and attack provenance graphs. By employing a rigorous multiplex graph construction technique based in graph spectral theory, data generated by FLURRY are not only ingestible by common heterogeneous GNN frameworks but also open up new avenues for cybersecurity practitioners to investigate data provenance graphs and to conduct more in-depth evaluation of down-stream learning tasks such as anomaly detection and attack classification. A real-time video demonstration of FLURRY is available. [1] Portions of the source code, ICD/user documentation for FLURRY, and nine thousand pre-generated graphs used in the GNN experiments are also publicly available. [2]

## 5 CONCLUSION

FLURRY is an end-to-end system which enhances all aspects of provenance graph representation learning, from simulating multiple angles of new cyber attacks to capturing and recording provenance to transforming it into learnable graphs. These graphs can be used for both training and testing graph embedding models. Additional graphs can be generated quickly or data reproduced for re-training, extending the lifetime of the learning model and any detection system it is a part of. Lastly, FLURRY also offers modular adaptation of its graph generation software in order to support dynamic graph generation in real systems, furthering security practicioners' and learning researchers' capabilities to expand frontiers in their respective research areas. As we demonstrate, FLURRY realizes the union of these two fields and will be a powerful tool in researching and designing resilient anomaly detection systems using provenance graphs.

[1] video demonstration: https://youtu.be/79yBLnL9PSg
[2] repository: https://github.com/mayakapoor/flurry

| | R-GCN | | | HAN | | | PROV-GEm | | |
|---|---|---|---|---|---|---|---|---|---|
| | F1 Micro | Precision | Recall | F1 Micro | Precision | Recall | F1 Micro | Precision | Recall |
| Backdoor | 99.4±0.5 | 99.4±0.5 | 99.4.0±1.2 | 97.2±2.9 | 99.5±0.4 | 95.4±5.3 | 97.5±2.1 | 97.0±3.2 | 98.0±1.7 |
| Brute Force | 67.2±2.6 | 67.0±10.3 | 68.1±2.8 | 62.2±0.6 | 52.6±8.4 | 67.1±6.8 | 60.0±4.9 | 84.6±8.6 | 58.1±6.5 |
| Command Injection | 75.9±2.0 | 76.4±4.9 | 75.8±3.8 | 78.1±2.2 | 75.2±5.2 | 80.1±3.6 | 71.3±0.8 | 80.1±6.2 | 68.5±1.9 |
| Data Exflitration | 94.1±0.6 | 95.1±2.3 | 93.4±2.4 | 90.6±2.4 | 91.8±3.7 | 90.2±4.4 | 88.9±7.3 | 92.0±4.1 | 89.1±12.5 |
| SQL Injection | 73.2±3.5 | 67.1±8.0 | 76.7±2.5 | 70.4±2.1 | 72.4±3.5 | 70.3±3.2 | 62.4±6.2 | 89.2±8.8 | 59.1±4.6 |
| SYN Flood | 98.1±0.9 | 98.6±1.3 | 97.5±1.0 | 97.4±0.7 | 96.7±1.0 | 98.1±1.2 | 94.1±5.1 | 89.7±10.3 | 98.5±0.4 |
| XSS DOM | 86.2±1.9 | 89.1±0.8 | 84.4.0±3.6 | 79.2±3.7 | 69.7±7.2 | 86.7±0.9 | 81.9±3.1 | 80.3±7.6 | 83.5±3.1 |
| XSS Reflected | 83.6±2.1 | 81.3±2.1 | 85.5±4.5 | 81.8±0.8 | 76.9±3.7 | 85.8±3.5 | 82.9±1.5 | 79.2±2.6 | 86.1±4.6 |
| XSS Stored | 85.5±1.2 | 84.9±2.8 | 86.3±1.6 | 83.3±1.7 | 91.5±2.6 | 79.0±2.7 | 78.2±3.3 | 76.0±5.8 | 79.7±3.0 |

**Table 2: Graph classification results using FLURRY data for training and testing.**

## REFERENCES

[1] digininja. 2020. Damn Vulnerable Web Application. https://github.com/digininja/DVWA.

[2] Xueyuan Han, James Mickens, Ashish Gehani, Margo I. Seltzer, and Thomas F. J.-M. Pasquier. 2020. Xanthus: Push-button Orchestration of Host Provenance Data Collection. *CoRR* abs/2005.04717 (2020). arXiv:2005.04717 https://arxiv.org/abs/2005.04717

[3] Xueyuan Han, Thomas F. J.-M. Pasquier, Adam Bates, James Mickens, and Margo I. Seltzer. 2020. UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats. *CoRR* abs/2001.01525 (2020). arXiv:2001.01525 http://arxiv.org/abs/2001.01525

[4] Xueyuan Han, Xiao Yu, Thomas Pasquier, Ding Li, Junghwan Rhee, James Mickens, Margo Seltzer, and Haifeng Chen. 2021. SIGL: Securing Software Installations Through Deep Graph Learning. arXiv:2008.11533 [cs.CR]

[5] Maya Kapoor, Joshua Melton, Michael Ridenhour, Siddharth Krishnan, and Thomas Moyer. 2021. PROV-GEM: Automated Provenance Analysis Framework using Graph Embeddings. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 1720–1727. https://doi.org/10.1109/ICMLA52953.2021.00273

[6] Emaad A. Manzoor, Sadegh Momeni, Venkat N. Venkatakrishnan, and Leman Akoglu. 2016. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. arXiv:1602.04844 [cs.SI]

[7] NetworkX developer team. 2014. NetworkX. https://networkx.github.io/

[8] Thomas F. J.-M. Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David M. Eyers, Margo I. Seltzer, and Jean Bacon. 2017. Practical Whole-System Provenance Capture. *CoRR* abs/1711.05296 (2017). arXiv:1711.05296 http://arxiv.org/abs/1711.05296

[9] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *The Semantic Web*. Springer International Publishing, 593–607.

[10] Jacob Torrey. 2020. Transparent Computing Engagement 5 Data Release. https://github.com/darpa-i2o/Transparent-Computing

[11] W3C-PROV Working Group. 2013. PROV-Overview: An Overview of the PROV Family of Documents. https://www.w3.org/TR/prov-overview/

[12] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J. Smola, and Zheng Zhang. 2019. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *CoRR* abs/1909.01315 (2019). arXiv:1909.01315 http://arxiv.org/abs/1909.01315

[13] Su Wang, Zhiliang Wang, Tao Zhou, Xia Yin, Dongqi Han, Han Zhang, Hongbin Sun, Xingang Shi, and Jiahai Yang. 2021. threaTrace: Detecting and Tracing Host-based Threats in Node Level Through Provenance Graph Learning. *CoRR* abs/2111.04333 (2021). arXiv:2111.04333 https://arxiv.org/abs/2111.04333

[14] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *WWW*. 2022–2032.