

**PROVENANCE GRAPH GENERATION FOR INTRUSION DETECTION**

**PERI ADHITYAN**

**SUUPERVISOR: A/P KE YIPING KELLY**

**COLLEGE OF COMPUTING AND DATA SCIENCE**

Submitted in Partial Fulfillment of the Requirements for the Bachelor of Computing (Computer Science) Nanyang Technological University

**2025**

Table of Contents

[Abstract 3](#_Toc180858169)

[Acknowledgements 4](#_Toc180858170)

[List of Figures 5](#_Toc180858171)

[List of Tables 6](#_Toc180858172)

[Introduction 7](#_Toc180858173)

[Background 7](#_Toc180858174)

[Objectives 8](#_Toc180858175)

[Literature Review 9](#_Toc180858176)

[Data Provenance 9](#_Toc180858177)

[Provenance Capture Systems - CamFlow 9](#_Toc180858178)

[CamFlow Exploration 10](#_Toc180858179)

[Setting Up CamFlow 10](#_Toc180858180)

[CamTool Installation 12](#_Toc180858181)

[Provenance Graph Visualisation 13](#_Toc180858182)

[Flurry Framework 14](#_Toc180858183)

[Scenarios 15](#_Toc180858184)

# Abstract

Cyber intrusion has been a growing issue for anyone with a digital footprint from individuals to companies to countries. This gives rise to the dire need for urgent and robust intrusion detection systems to pre-empt and mitigate cyber security incidents before major damage can be done. Data provenance graphs are being researched and utilized in auditing and intrusion detection for cyber security. Provenance graphs can depict the entirety of system execution and assist in gathering information regarding the origin of data, the current state and the entities that acted upon it. This project aims to setup existing provenance capture systems to generate provenance graphs during benign system execution and simulated attack scenarios. Additionally, the generated graphs will be used to train and test models in order to develop intrusion detection systems that provide semantic rich information for real-world application.

# Acknowledgements

I would like to express my gratitude for my final year project supervisor, Associate Professor Ke Yi Ping, Kelly for her mentorship, guidance and assistance throughout this project.

Additionally, I would like to thank PHD Student Mr. Li Wei for taking the time to answer queries and offer advice with regards to this project.

# List of Figures

# List of Tables

# Introduction

## Background

The world we live in today is one that relies on the virtual cyber space as much as it does on the physical space, some might say even more so. But the exponential growth in cyber activity has led to the direct increase in cyber-attacks as well. An increased digital footprint gives way to an array of attack vectors for threat actors to use as an entry point. Year by year we see an increase in cyber-attacks, with 2023 showing a 72% increase in data breaches surpassing the previous record high in 2021 [[Forbes]](https://www.forbes.com/advisor/education/it-and-tech/cybersecurity-statistics/).

To counteract these breaches and detect intrusion, researchers have turned to the usage of a provenance graph-based system. Provenance graphs are directed acyclic graphs used to determine relations between entities such as sockets, files and users, and actions such as the flow of data between them. These give an insight as to what benign activity might look like as compared to when a threat actor has breached the system for malicious intent. Researchers from Harvard and Cambridge Universities have pointed out the extensive capture of security sensitive kernel operations, the explicit relations it depicts between objects, that intrusions result from unexpected relations and the robustness of graphical representation [[Paper]](https://www.seltzer.com/assets/publications/Provenance-Based-Intrusion-Detection.pdf).

Provenance graphs show great advantages when it comes to intrusion detection. Since provenance graphs show system execution by displaying relations between system objects, simple audit files that ae unstructured and hard to read can be converted to provenance graphs. Secondly, provenance graphs are hard for attackers to replicate or forge as they are rich in semantics. They take into consideration spatial and temporal information which allow security analysts to conduct thorough and effective investigations. Finally, provenance graphs store all the execution history which aid analysts in investigating Advanced Persistent Threats (APTs). APTs are known for their long term embedding in systems and stealth in being undetected. The complete history of system execution provided by provenance graphs can easily aid analysts in the event of APTs [[Paper]](https://www.sciencedirect.com/science/article/abs/pii/S0167404821001061).

However, these discoveries and techniques do not come with their downsides. Recent Provenance Graph Based Intrusion Detection Systems (PIDS) used embedding techniques that incur high computational resource cost. Furthermore, as these systems take inputs from graphs, there are detection delays. Finally, these systems output uninterpretable results that do not give much detail other than the fact that they have been flagged out due to the fact that they deviate from normal system operation. [[Paper]](https://arxiv.org/pdf/2404.14720).

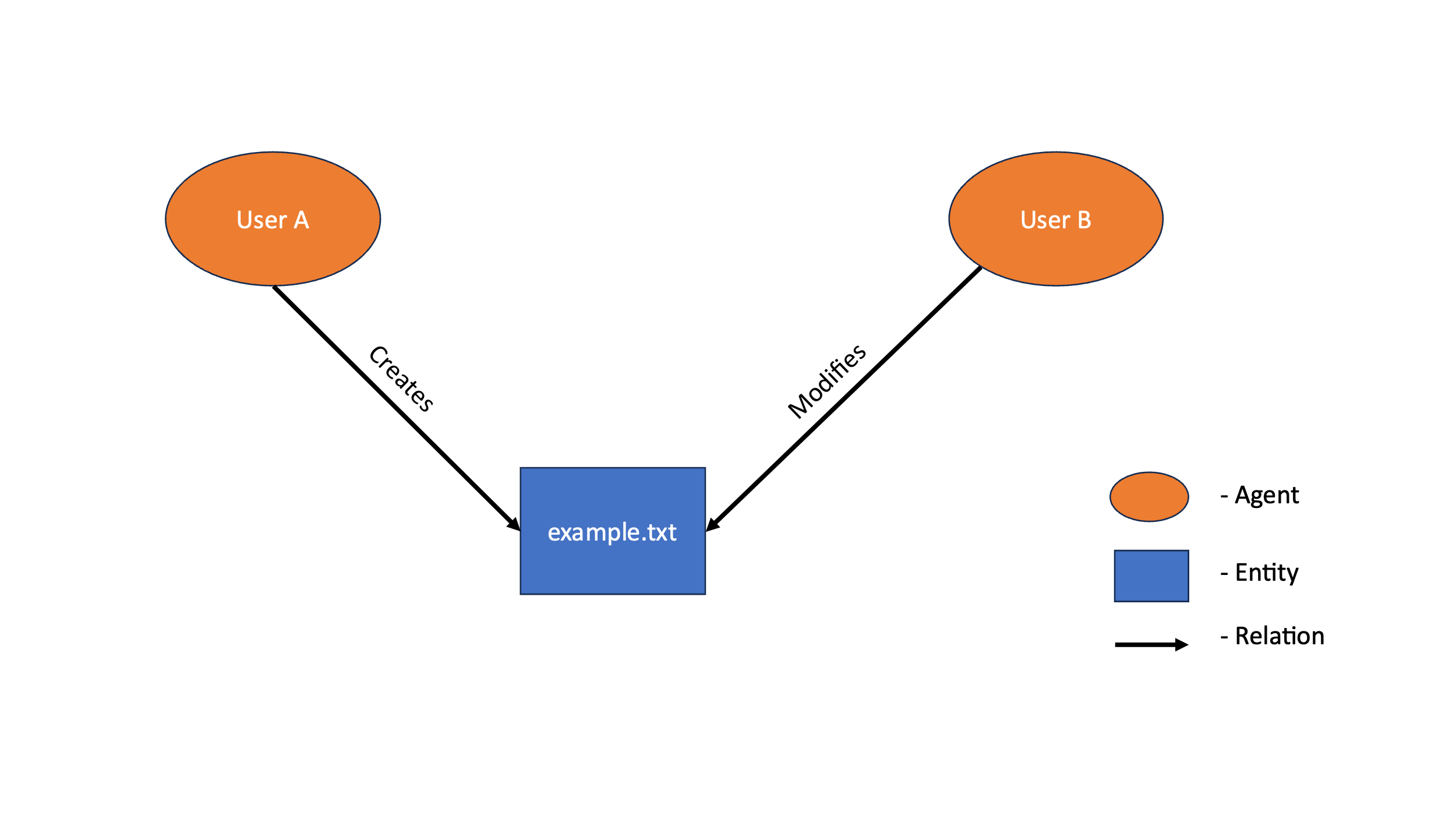
## Objectives

This study aims to setup a provenance capture system to capture whole system provenance. It will leverage CamFLow, an open-source project to bring observed provenance collection to the Linux Operating System that complies with W3C PROV-DM standard. These provenance graphs will be generated by implementing the FLURRY framework, an end-to-end data pipeline which simulates cyberattacks captures provenance data from these attacks into data provenance graphs and incorporates this data with a framework for training deep neural models that supports preconfigured or custom-designed models for analysis in real-world resilient systems[[paper]](https://arxiv.org/pdf/2203.02744) Finally, it will utilize these generated graphs to train and test models and gather statistics for benchmarking and further improvement.

# Literature Review

## Data Provenance

Data provenance in the context of computer systems is the description of relationships between entities, the activities conducted using those entities and the agents associated with them. For example, if *user A* creates a file *example.txt*, followed by *user B* editing the file, *example.txt*, data provenance will give us these relations from the creation, all the way through till the file is destroyed. *Figure 1* depicts how a simple data provenance graph can be generated based on the earlier scenario.



**Figure 1: Simple provenance graph showing relations between agents and entities**

## Provenance Capture Systems

Over the years, researchers have developed various provenance data capturing systems and libraries. This study will utilize CamFlow, an open-source project to bring observed provenance collection to the Linux Operating System. The upper hand CamFlow has compared to other provenance capture systems, is that CamFlow uses a self-contained, easily maintainable implementation relying on a Linux Security Module, NetFilter and other existing kernel facilities. This provides a mechanism to tailor the captured provenance data to the needs of the application, making it easy to integrate provenance across distributed systems [[paper]](https://arxiv.org/pdf/1711.05296).

## CamFlow Features

CamFlow captures a system’s provenance data through *Linux Sucrity Model* and *NetFilter* hooks which is subsequently transferred to user space for storage and analysis. CamFlow tackles the major concern of generating extremely large whole-system provenance by providing mechanisms that capture only necessary subsets of the entire provenance graph required by applications. It further allows the possibility to limit the capture to specific edge or node types and flows from specific sources such as inodes, network interfaces, security contexts and user IDs.

# CamFlow Provenance Capture System

## Setting Up CamFlow

The first task for this project would be to setup CamFlow on a system in order to capture whole system provenance. CamFlow is recommended to be installed on Fedora Linux which is based on the Linux OS kernel architecture. For maximal compatibility, a virtual machine running Fedora Linux was setup with the following configurations:

Host OS: Windows 11

Hypervisor: VirtualBox 7.1

Virtual Machine OS: Fedora 35 64-bit

Memory: 8GB

Storage 100GB

Referencing the CamFlow project website [[site]](https://camflow.org/), the fastest installation method was used, downloading the package manager and installing the camflow package using the following commands



After which to activate the services, the following commands need to be run



Finally, reboot the system. Hitting ‘shift’ when the VirtualBox splash screen appears will bring the system to the boot menu, where the kernel option with the word ‘camflow’ should be chosen. Rebooting the system should set camflow as the default kernel to boot from, however in the event that does not occur the following command can be run within a terminal:



This will list the existing kernels installed, simply find the kernel with ‘camflow’ in it’s title:



**Figure 2: Screenshot of the results after listing existing kernels**

After identifying the correct kernel simply run the following command to ensure that the default kernel is set:



Ensure that the kernel is the correct one according to the system being setup. In the context of this project, installation scripts were created to facilitate multiple VMs being setup with ease. The entirety of the script will be listed in the appendix below



## CamTool Installation

CamTool is an extension of CamFlow that helps with the visualization of simple provenance graphs. The Message Queuing Telemetry Transport (MQTT) protocol is used to publish provenance in real time on CamFlow’s demo website (<https://camflow.org/demo>). Another script was used for the installation of CamTool as shown below. To collect logs in the w3c format the *camflowd.ini* file needs to be edited, followed by restarting the *camflowd.service* service.



## Provenance Graph Visualisation

To visualize provenance using CamFlow and CamTool, a simple scenario of file creation and deletion was used. In a browser, open the demo website and click ‘Start CamFlow MQTT’, ensure the browser is open throughout the demonstration. To capture and track the provenance of a file from creation to deletion the following commands were issued:



On the demo website the following provenance graph can be visualized tracing the example.txt file from creation to deletion.



**Figure 3: Provenance graph generated from example.txt**

# Flurry Framework

## Background

To bridge the gap between generating usable provenance data and utilizing these graphs for unsupervised learning to detect intrusions, researchers have developed Flurry. Flurry is equipped with provenance capture, graph generation tools and can execute automated cyber-attacks and collect provenance graphs that can be used as input for machine learning tools. [[Paper]](https://arxiv.org/pdf/2203.02744) The benefits provided by this framework are that it is an end-to-end pipeline that takes in system execution and models it as a multi-layer provenance graph, displays execution of cyber-attacks, and provides a plug-and-play framework for graph learning models to analyze provenance graphs.

## Flurry Installation

The author of the Flurry Framework, Maya Kapoor, has provided scripts that automated the download and installation of the needed dependencies through her git repository <https://github.com/mayakapoor/flurry> . However, it has since been deprecated and requires manual installation of the following dependencies to run as intended, listed below:

1. XAMPP
2. DVWA
3. MQTT
4. Additional Dependencies

Scripts have been written to ease the installation of Flurry and its dependencies.

### Cloning Flurry Source Code

The first steps are to clone the source code from the git repository followed by resetting it to the last working version of the framework



Followed by cloning *flake* into the created *flurry* folder and configuring the code as shown in the script in the appendix.



### CamFlow Reconfiguration

In order to ensure compatibility between CamFlow and Flurry, the following changes need to be made to configuration files:

*camflowd.ini*

1. output=mqtt
2. format=w3c
3. address=localhost:1883
4. password=camflow
5. qos=0

*camflow.ini*

1. ;all=false
2. Duplicate=true

The automated script displaying the entire configuration files and changes is listed in the appendix for clarity.

### XAMPP

XAMPP is a popular PHP development environment containing, MariaDB, PHP and Perl.

Installing the necessary dependencies:



Download the XAMPP executable from <https://www.apachefriends.org/> and installing it:



Finally complete the installation with the GUI.

