CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Leveraging Root Cause Analysis for Mimicry Attack Detection

Detection of Mimicry Attacks Using Root Cause Analysis in Provenance Graph Host-based Intrusion Detection Systems

Master's thesis in Computer Science and Engineering

SEBASTIAN KVALDÉN
ERIK HENRIKSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025

Master's thesis 2025

Leveraging Root Cause Analysis for Mimicry Attack Detection

Detection of Mimicry Attacks Using Root Cause Analysis in Provenance Graph Host-based Intrusion Detection Systems

SEBASTIAN KVALDÉN
ERIK HENRIKSSON

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Detection of Mimicry Attacks Using Root Cause Analysis in Provenance Graph Host-based Intrusion Detection Systems
SEBASTIAN KVALDÉN
ERIK HENRIKSSON

Supervisor: Romaric Duvignau, Computer Science and Engineering
Advisor: Martin Forssen, Recorded Future
Examiner: Ahmed Ali-Eldin Hassan, Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Detection of Mimicry Attacks Using Root Cause Analysis in Provenance Graph Host-based Intrusion Detection Systems
SEBASTIAN KVALDÉN
ERIK HENRIKSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Cyber attacks are a constant threat to all computer systems, and there are many different approaches to detecting attacks in real-time. One such approach is a Host-Based Intrusion Detection System (HIDS), which detects threats on a single host machine. A recent addition to HIDS is Provenance-Graphs [1], [2], which creates a representation of the flow of all system calls. Many of these systems condense the provenance data for increased space efficiency. This, however, creates a vulnerability for a type of evasion attack called Mimicry Attack. Mimicry Attacks work by copying benign sequences of system calls and injecting them into the attack to mask their malicious intent.

This thesis aims to combat this vulnerability by incorporating coarse-grained Root Cause Analysis (RCA) into Provence-Graph-based HIDSs (Prov-HIDS). Earlier attempts at fine-grained RCA, where the analysis is done on the process level, have been successful in detecting Mimicry Attacks. These solutions have, however, had significant overhead, consuming most of the host's resources [3]. The thesis aims to combat this issue by utilising the summarisation approach of coarse-grained detection models on the system's root nodes. This solution increases the context for the classification models during threat detection. The detection accuracy increased from 13.4% for the Unicorn Prov-HIDS to 99.5% with the RCA implementation, when evaluated against the StreamSpot dataset. With only a 4.97% increase in execution time and a 2.17% peak memory consumption increase. However, the RCA implementation also caused issues with misclassification of benign data. The findings also raise concerns about the reliability of existing datasets. To address these challenges, new dataset criteria are proposed, and the development of a new, high-quality dataset is essential to advance research in this field.

Keywords: Mimicry Attack, HIDS, Prov-HIDS, Root Cause Analysis, APT, Provenance Graph

Acknowledgements

We would like to express our gratitude to everyone who supported us throughout this thesis. We thank our supervisors, Romaric Duvignau and Martin Forssén, as well as our examiner, Ahmed Ali-Eldin Hassan, for valuable feedback during this thesis. Thank you to our opponents, Emil Lindblad and Amanda Dyrell Papacosta. Special thanks to Erik Stenvall, My Qvick-Vester, Mikael Gordani, and all the other Futurists at Recorded Future for allowing us to be a part of their team and organisation, and collaborating with us in this thesis.

We would also like to acknowledge all the people we had the opportunity to meet during our studies here at Chalmers. Thank you for intriguing discussions and good memories. We wish you all the best and hope to see you again!

Finally, a sincere thank you to our friends and families for their unwavering trust and support.

Thank You!

Sebastian Kvaldén & Erik Henriksson

Gothenburg, 2025-07-13

Contents

List of Figures

List of Tables

Abbreviations

APT Advanced Persistent Threat.

cbc-edr Carbon Black Cloud - Endpoint Detection and Response.

CVE A record of Common Vulnerabilities and Exposures.

CVSS Common Vulnerability Scoring System.

C&C Command and Control Centre.

DBSCAN Clustering Model; Density-based spatial clustering of applications with noise.

DAG Directed Acyclic Graph.

DARPA TC The Defense Advanced Research Projects Agency Transparent Computing.

FP False-positive.

HDBSCAN Clustering Model; Hierarchical density-based spatial clustering of applications with noise.

HIDS Host-based Intrusion Detection System.

IDS Intrusion Detection System.

MDS Multi Dimensional Scaling

MST Minimum Spanning Tree.

NIDS Network-based Intrusion Detection System.

No-op No Operation.

PC Program counter.

PSW Parallel Sliding Window.

Prov-HIDS Provenance Graph Host-based Intrusion Detection System.

RCA Root Cause Analysis.

SSH Secure Shell.

TA Technical Areas.

TC Transparent Computing.

VM Virtual Machine.

1

Introduction

In the information era, where more and more of the everyday societal and business functions rely on digital information flows, the risk of cyber-attacks is ever-increasing. This risk and an increase in the complexity of information systems broaden the attack surface that an adversary might exploit. To combat this, organisations have adopted a defence-in-depth strategy. Defence-in-depth is a series of defensive measures implemented at all levels of the information system architecture to detect and limit the effect of malicious adversaries. One such defensive measure is the adoption of Host-Based Intrusion Detection Systems (HIDS). A HIDS detects malicious activities across all applications running on a single host.

The HIDS functions as a warning system, alerting administrators to unusual activity. There are two primary detection methods: signature-based and anomaly-based detection. The signature-based approach has seen limited success, particularly against zero-day exploits. A zero-day attack is an attack never seen before, and it is thus infeasible to create proactive rules against undiscovered threats. The alternative approach, anomaly-based detection, avoids this problem by not defining detection rules and instead tries to identify outlier behaviour in the monitored system. In theory, this allows for anomaly-based systems to be effective against various known and unknown threats.

1.1 Context

This thesis is motivated by ongoing research into the effectiveness of HIDSs, particularly their ability to detect sophisticated attacks. General anomaly-based HIDSs have shown promise in detecting novel intrusions, but some advanced attacks might still go undetected. One such attack is the Mimicry Attack. It is an attack where the malicious program disguises itself as a benign application to avoid detection [4].

In order to combat mimicry attacks, among other attacks, previous research developed a new type of HIDS called Provenance Graph-based HIDS (Prov-HIDS). This is an anomaly-based detection system that identifies threats by analysing program behaviour through provenance graphs. A provenance graph, in this context, is a representation of the underlying system, where nodes represent system entities and edges represent the interactions between these entities.

Prov-HIDSs were previously thought to be secure against mimicry attacks. However,

A. Goyal *et al.* identified a critical vulnerability in this approach, demonstrating three methods for modifying attack scripts by injecting benign behaviour to mimic the HIDS detection model and evade detection [5]. This flaw stems from a design choice where the HIDS moves away from the provenance graph domain into a compressed format where information and context are lost. A. Goyal *et al.* demonstrated how this weakness could be exploited in their proposed attack strategy.

A tempting and simplistic approach to address this issue is to abandon the compression step and classify anomalies directly on the provenance graph. Abandoning the compression step might work in theory. However, a single host may generate multiple gigabytes of data every operational day. Such an amount of data makes this approach unfeasible in reality [3]. While some Prov-HIDS detection algorithms have been prone to mimicry attacks, the core implementations are often sound [1], [2], [6]. In this thesis, we will, therefore, extend the detection domain of some existing coarse-grained HIDS models by incorporating Root-Cause Analysis (RCA) to address this problem.

1.2 Problem

As previously mentioned, one of the main challenges in adopting Prov-HIDSs is the large memory usage to construct a complete provenance graph over a system's execution patterns. This graph must include system calls, file access, and network connections to provide a comprehensive picture of process activities. To combat this rise in memory usage, modern HIDSs employ various compression methods or restrict monitoring to only certain parts of the provenance graph to create a condensed version of the original graph. For example, one method is to condense a graph by looking at the top K most uncommon graph paths [6], another is to encode the graphs into histograms [1]. There are several other data compression techniques, and all condensing techniques introduce a trade-off between space efficiency and accuracy. It is also here in the compression step that the problem arises, where mimicry attacks make use of the reduction in granularity of the Prov-HIDS detection model to disguise themselves as benign activity.

There have been earlier attempts to keep the compression step and simultaneously improve the detection rate. Especially for of Prov-IDSs, which is similar to Prov-HIDS but can monitor multiple hosts at once. One such attempt is by incorporating RCA into the IDS. It is also noteworthy to mention that, to our knowledge, this has only been done on a fine-grained system, specifically the Prov-IDS named R-CAID [3]. The classification for R-CAID was performed at the node/process level rather than at the graph level. This Prov-IDS was mostly successful in detecting malicious processes with a low false-positive rate. However, the fine-grained nature of the model presents a large increase in space and computational overhead. This increase makes it generally unsuitable for full-scale adoption.

The authors of R-CAID have, because of this overhead, also proposed a direction for future improvements to incorporate RCA in coarse-grained models to improve the context for decision-making in models that do not currently use RCA and evaluate

its usefulness [3]. This thesis thus tries to fill this gap, and the research questions that arise here are the following:

- Is it possible to achieve accurate detection against mimicry attacks in coarse-grained Prov-HIDS with RCA?
- Is it possible to efficiently incorporate RCA with regards to time and space complexity?

1.3 Goals and Challenges

This thesis, as previously mentioned, will investigate the research gap of whether coarse-grained Prov-HIDS detection, combined with RCA, can enhance the accuracy when detecting Mimicry Attacks. A possible improvement should not come at the cost of large increases in time and space complexity, since that could render the approach infeasible in real-world applications.

To achieve this improvement, some key challenges have been addressed in the thesis, including identifying how RCA can be incorporated into probabilistic detection models. A second challenge is identifying how it can work with existing features of some publicly available Prov-HIDS to increase its detection accuracy. Another challenge is analysing how this addition affects the time and space complexity of the complete detection system.

Clearly stated, the challenges are the following:

1. Design a system for incorporating RCA into a currently available coarse-grained Prov-HIDS algorithms.
2. Keep time and space complexity similar to current approaches.

The goals we aim to achieve are:

1. Evaluate RCA's effectiveness when combined with current coarse-grained Prov-HIDS approaches.
2. Raise the detection probability for mimicry attacks in Prov-HIDS systems.
3. Propose a generalised integration of RCA into coarse-grained Prov-HIDS systems.

See Section 4 for more details on how this should be achieved.

1.4 Limitations

There are a few limitations to the scope of this thesis. The first limitation is that the implementation of RCA will only be applied to one existing HIDS model and may not be directly applicable to others despite their similar traits and the goal to create a universal system. There are too many available Prov-HIDS to ensure compatibility

across all platforms. The findings in this project could thus be considered a proof of concept.

A second limitation is the degree of available data. The new findings intend to be tested against the current "gold standard" datasets for evaluating intrusion detection systems (StreamSpot [2], [7], [8], Theia from DARPA Engagement 3 [9]–[11], and ATLASv2 [12]–[14]). However, these results might not be generalised to all real-world scenarios. Nevertheless, they should be good enough for evaluation and make the findings directly comparable to those of other papers that utilise the same datasets.

The final limitation is based on the fact that the adversarial techniques deployed to avoid detection are constantly changing. Evaluating the findings against all currently available and future attack techniques is infeasible. The focus in this project will thus be strictly against limiting the use of the mimicry techniques presented by A. Goyal *et al.* in their previous report [5], together with the regular attacks already included in the datasets.

1.5 Outline

The thesis is structured around eight chapters, including this one. Chapter 2 introduces key concepts, theories and systems related to the developments created during this thesis. Furthermore, Chapter 3 explains the current problem, available data and mimicry attack techniques in detail. This is followed by the thesis methodology and evaluation metrics in Chapter 4, and the system implementation in Chapter 5. Test results and benchmarks are located in Chapter 6. Finally, a discussion on remaining problems and future works can be found in Chapter 7, and Chapter 8 contains the concluding remarks of the thesis.

2

Background

This chapter will give background information on key terms, theories, and systems crucial for this thesis.

2.1 Provenance Graphs

A provenance graph $G = \{V, E\}$ is a representation of a system's execution where each vertex (V) represents a system entity, such as a `process`, `file` or `socket`. Every edge (E) in the graph represents an interaction between the two connected entities, such as `read()`, `write()`, `execute()`, and more. Edges are represented as a tuple such that an edge $e = \{s, d, x_0, ..., x_i\}$, where source ($s$) and destination ($d$) is combined with other various data points (denoted x in our definition). There is currently no industry or academic standard on the contents of the edge tuples other than that they contain a source and destination. The contents of the variables x_i may take any form or be omitted completely depending on the needs of the underlying detection system (or other application) that uses the provenance graph. Some common adaptation, particularly for IDSs, is the addition of timestamps to enable sequential ordering of events or system call parameters. For example, a provenance graph that includes a timestamp (t) and system call parameters (p) in the edge tuple would be represented as $e = \{s, d, t, p\}$. The graph's vertices also often have a similar representation where the tuple contains information about that specific system entity, such as process ID, file name or location.

Unicorn, a Prov-HIDS, rely on audit systems such as CamFlow [15] to collect provenance graph data. These applications can limit the included data in the edge and vertex tuples. However, it appears to contain sufficient data for most HIDS discussed in this thesis [1], [2], [6]. That said, not all collected data is always necessary for effective detection. Some data might cause over-fitting, unnecessary noise, or simply waste memory. The redundant data could, for example, be removing host-specific folders associated with `read()` and `write()` operations, as they will likely provide very little information on the system state while still increasing the amount of stored data. A solution would be data simplifications, i.e., removing the redundant data. An example of this data simplification could be removing user-specific identifiers in folders or destination IP addresses (from the adversary to the victim). User-specific identifiers such as file path, e.g., `/home/alice/.ssh/id_rsa`, are too specific for the model, but `/home/*/.ssh/id_rsa` is not and will better

match the training data. It does not matter whose `id_rsa` file is accessed by an adversary, since none should be accessed. One Prov-HIDS that utilises this example is ProvDetector [6].

Provenance Graphs as Directed Acyclic Graphs

Provenance graphs are often described as Directed Acyclic Graph (DAG) [16]–[18]. This model is, however, not entirely accurate in computer systems, as pointed out by A. Goyal *et al.* [3], who highlighted that cycles might occur when the model meets reality. It is, for example, entirely possible that two processes continuously call each other, which would create an infinite loop in the model and break the principles of a DAG.

The key to understanding how this model still holds, even with the possibility of process call loops, is the realisation that provenance graphs used for modelling system behaviour have an additional data point: time. This means that even if the same processes call each other recursively, the time parameters will differ. These differences allow them to be split into separate theoretical entities. To clarify this concept, one may project the model into a 3-dimensional space, where time is the third dimension. The 3-dimensional model is then projected back into a 2-dimensional space.

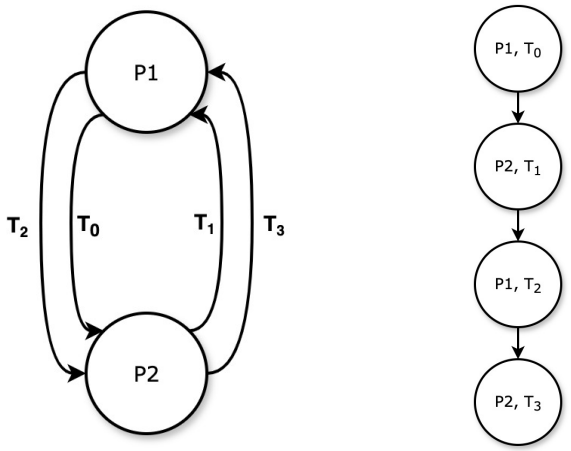
Different Prov-HIDSs handle this conversion a little differently. One example is R-CAID, which mentions that it is possible to version each graph at each instance when information flows into a node [3]. This versioning ultimately creates a duplicate of the entity node at every instant it is called. However, such implementations are very memory-consuming. Most Prov-HIDS instead adopt a model that allows loops that associate each edge with a timestamp. Using timestamps enables the system to remodel the graph into a true DAG by linking each vertex with its incoming timestamp. A process p that gets called twice can be modelled as two different entities by separating them via their timestamps, $p_{t=0} \neq p_{t=1}$. See Figure 2.1 for how a loop between processes can be modelled as a DAG with the addition of timestamps.

2.2 Threat Description

This section will describe the relevant threats for this thesis and the auxiliary term No Operation (no-op), which plays a key role in the attack's success. The two types of attacks are the Advanced Persistent Threat (APT) attack and Mimicry Attacks.

2.2.1 Advanced Persistent Threats

APTs are sophisticated, slow-moving attacks that often use spear phishing (a phishing attack that targets one or a few specific individuals). The attack is slow-moving and thus performed over several days, months, or even years. The extended time frame and the stealthy nature for the attack will thus make it harder for IDSs to detect it.



(a) Regular (b) DAG

Figure 2.1: Process Call Loop Reordered as DAG.

An APT attack typically progresses through several distinct phases [19], [20]:

1. **Reconnaissance:** The attacker begins by gathering information about the target. This phase involves identifying vulnerable systems, employees with access privileges, and other potential points of entry. The goal is to build a detailed profile that will inform later stages of the attack.

2. **Initial Breach:** Using the insights gained from reconnaissance, the attacker gains unauthorised access to the system, often through methods like spear phishing or exploiting known vulnerabilities. This phase may span a long period of time to avoid detection and maintain stealth.

3. **Establishing Control:** After successfully breaching the system, the attacker installs malware or establishes backdoors to maintain persistent access. Communication with a Command and Control (C&C) server allows the attacker to issue malicious instructions and manipulate compromised systems remotely.

4. **Lateral Movement:** With initial access established, the attacker seeks to expand their foothold by moving laterally across the network. They may escalate privileges, compromise additional devices, and infiltrate deeper into the organisation, often blending in with normal user activity to remain undetected.

5. **Execution of Objectives:** Finally, the attacker carries out their primary goal. This could involve exfiltrating sensitive data, disrupting operations, or conducting surveillance. Data is often sent back to the attacker via the established C&C channel.

A HIDS can only detect the attack in steps 2 and 3. That is why an APT is dangerous, and state-of-the-art IDS systems need countermeasures against them [19], [21]. One system specifically designed to counter an APT is Unicorn [1] (see Section 2.7).

2.2.2 Mimicry Attacks

An expansion of the APT attack is the mimicry attack. These are stealthy evasion attacks that aim to mimic benign behaviour and thus go undetected by IDSs and anti-malware systems. The goal of the expansion is to increase an APT's success rate against state-of-the-art countermeasures such as the Prov-HIDS discussed in this thesis. Wagner and Soto introduced the mimicry attack in 2002 [4].

They explained how a trivial mimicry attack could be constructed by adding typical system instructions in a non-malicious sequence as no-ops to a regular attack script. These instructions, e.g., `read()`, will be performed, but their result will be ignored. The IDS cannot determine that this instruction was a no-op. The attacker can then mimic the regular behaviour by injecting benign operations as no-ops in a sequence known by the IDS, ignoring their results, and replacing some non-malicious instructions with malicious ones of the same kind. This attack will create a seemingly benign instruction sequence while completing its malicious intent [4].

Several different types of detection systems have been created to detect mimicry attacks. Particularly, Prov-HIDSs like Unicorn [1], StreamSpot [2], and ProvDetector [6]. They were initially believed to detect mimicry attacks effectively. This belief was, however, disproved by A. Goyal *et al.* in 2023 [5] when they showed that these systems were more robust but still susceptible to mimicry attacks. They had a 100% success rate of evading detection by leveraging a more sophisticated mimicry attack that exploited the compression or summarisation of the provenance graphs. They use APTs, i.e., attacking over a long time, hoping that the IDS will flush the old malicious instructions out by injecting the system with non-malicious substructures from the training set, e.g., the DARPA dataset [10]. When evading StreamSpot, 80K edges were injected, including 28K malicious edges into the original benign graph of 4.8M edges. This results in an increase of only 1.7% edges.

No Operation

An important concept for the mimicry attack is the no-operation (no-op) instruction. No-op is a benign machine code instruction which has no operational impact except skipping the memory address and increasing the program counter (PC) [22]. The no-op instruction is inherently benign and not harmful, however, it is often used in malicious applications.

One of the oldest cyber-attacks is a buffer overflow attack that utilises no-op machine code instructions via a no-op sledge and overwrites the return address and other crucial information [23]. Similarly, high-level system calls such as `read()` or `getpid()` can be used as no-ops if they are called and discarded without processing or using the result from the operation [4]. This is utilised in mimicry attacks, where the no-op instructions are included as discarded system calls. The attack does not overwrite information but uses these instructions to hide the malicious instructions in a seemingly benign instruction sequence [4], [22].

2.3 Host Intrusion Detection System

Intrusion Detection Systems (IDS) are monitoring systems that alert upon detecting malicious activity or potential threats. There are two types of IDSs: Host-based (HIDS) and Network-based IDSs (NIDS). NIDS focuses on monitoring an entire network, for example, a factory, while HIDS monitors a single device, such as a workstation or server. HIDS can further be categorised into two different categories: Signature- and Anomaly-based systems. Signature-based systems are simple and rely on user-predefined rules such as blacklists and whitelists. However, creating a signature-based system that effectively covers sophisticated attacks, including zero-day exploits, is infeasible. An anomaly-based system is more robust to these types of attacks since it monitors and learns the benign behaviour of the processes. Learning benign behaviour allows it to detect malicious behaviour even in zero-day attacks [4], [24].

Prov-HIDS

A newer subtype of Anomaly-based HIDSs is Prov-HIDSs. They utilise provenance graphs (Section 2.1) and often along with other techniques such as machine learning and vector distance to examine the graphs. Prov-HIDSs can effectively detect Advanced Persistent Threats (APT) (see Section 2.2.1).

Prov-HIDSs do, however, use a considerable amount of data. The data can come from a stream or a static database of system activity. State-of-the-art Prov-HIDS like Unicorn [1] and StreamSpot[1] [2] use continuous unbounded data streams collected from the device's user and process behaviour. Other systems can incorporate a static dataset collected from previous behaviour [3].

Prov-HIDSs can also be categorised by the level of operational granularity, with coarse-grained and fine-grained detection models. A coarse-grained model detects and evaluates threats on a graph or sub-graph level, i.e., it accounts for multiple edges and nodes in the complete graph structure. A fine-grained detection model, however, decides on information available at a single node and its surrounding edges. Both coarse- and fine-grained models show similar detection accuracy when compared against the same datasets. This thesis establishes a clear distinction between coarse-grained and fine-grained detection models due to their operational differences. Furthermore, the newly developed mimicry techniques specifically target coarse-grained Prov-HIDS; they are the primary focus of this work. [5].

2.4 Root-Cause Analysis

A crucial concept for this thesis is Root-cause analysis (RCA). RCA is an analysis technique applied to various topics and domains. The book *Root cause analysis* by B. Andersson and T. Fagerhaug presented the definitions of RCA as "Root cause analysis is a structured investigation that aims to identify the true cause of a problem

[1]The StreamSpot project created both a Prov-HIDS and a dataset for testing

and the necessary steps to eliminate it." [25]. However, in the intrusion detection domain, the definition requires some modification since an IDS aims to detect the presence of current threats instead of directly eliminating them. The following is this thesis's definition of RCA:

> *"A technique that allows an event to be directly linked to its true origin regardless of how distant it may be."*

To exemplify the usage of RCA, consider a provenance graph for a web browser under attack by an adversary where the adversary managed to download a malicious script and run it, which opens a Secure Shell (SSH). The chain of events would be: $(Browser \Rightarrow Download\ file \Rightarrow\ Run\ File \Rightarrow SSH)$. The root cause is the web browser. Implementing RCA in the IDS makes it clear that the attack originates from the web browser, regardless of how many intermediate processes the attacker creates before arriving at the SSH process.

2.5 Data Sketches

Data sketching is a tool used in large-scale data streaming. Storing information retained from data streaming can quickly become computationally and memory expensive [26]. There are two known ways of handling the exponentially increasing streaming data. The first is to improve the hardware, but this is a costly and possibly temporary solution. Another way to combat this problem is to summarise the data in data sketches.

In Prov-HIDSs, this summarisation is often achieved through sketching the key features of the provenance graph, which are intended to be used for detection. Such features could be neighbourhood embeddings, top K-paths, or similar features. Summarising the data will save computational power and memory capacity, but it will also cause the data to lose some context, accuracy, and granularity [26]–[28].

HistoSketch

HistoSketch is the data sketching method used in this thesis. It is an effective method and was developed in 2017 [27]. The idea of HistoSketch is to retrieve summarised data as histograms, which are then sampled via consistently weighted sampling to create a fixed-sized sketch from these samples [29].

Furthermore, HistoSketch utilises concept-drift to forget stale data over time. This is achieved by gradually decaying the histogram values at set intervals, which ensures that non-updated histogram counts will shift towards zero as time progresses. HistoSketch continually updates the sketch as new information is added to the histogram. To achieve this, it utilises a normalised min-max similarity, which is an accurate count summarisation model; it counts the number of times an element has been seen and normalises the counts to make histograms comparable over time [30]. It is these normalised counts that are used in the weighted sampling to build the final sketch. HistoSketch has an accuracy loss of 3.5% while it has a computational speed up of 7500x compared to similar sketching systems [27].

2.6 HDBSCAN

After the analysis is done in a Prov-HIDS, the data must be modelled in order to find anomalies. One such model is the HDBSCAN [31] model, which is a hierarchical extension of the clustering model called density-based spatial clustering of applications with noise (DBSCAN) [32]. Both of the models cluster the data based on the density (distance) between data points. Compared to its predecessor, the HDBSCAN model allows for clusters of different densities, which allows for a greater degree of flexibility in the underlying data. For example, if the underlying training data contains more data points related to a subset of potential clusters, data points from the underrepresented subsets are at risk of being incorrectly labelled as noise. This is because their variance is likely higher due to the mathematical relationship between variance and the number of available data points [33]. This risk can be lowered by allowing for different densities between clusters. Lowering the risk of mislabeling enables these sparsely populated areas of the data to form clusters with lower density requirements.

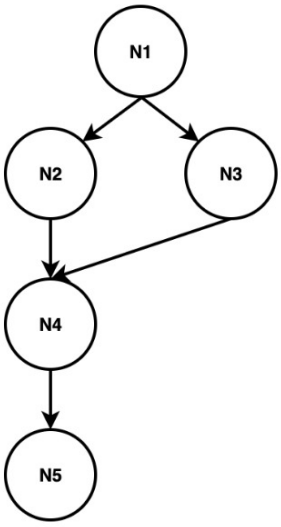
A broad description of the algorithm is that it begins by computing a *mutual reachability distance*, i.e., a metric that captures the distance between two points while also accounting for the local density around each point. This distance is then used to construct a weighted graph, where each edge's weight corresponds to the mutual reachability distance between the connected vertices. From this graph, a minimum spanning tree (MST) is constructed and then converted into a condensed tree. The tree is then recursively split into smaller clusters by splitting off the edges that exceed the distance threshold (λ). It starts splitting the longest edges, as long as the new clusters exceed the minimum cluster size hyperparameter.

The final clusters are then chosen as the clusters that persist over the largest range of distance thresholds without splitting into smaller clusters. The final clusters are often described as the most stable clusters.

2.7 Unicorn

Unicorn [1] is a state-of-the-art coarse-grained Prov-HIDS scheme that collects data via unbounded real-time data streams; it is also the base Prov-HIDS used for integrating the RCA system developed in this thesis. Unicorn was specifically designed to detect APTs. It leverages graph sketching, which is a data sketching technique that condenses graphs built over an extended period to retain contextual information. This technique increases the chance of finding APTs with limited space complexity. It achieves this by collecting provenance graph data and systematically condensing it into histograms. Unicorn then transforms the histograms into constant-sized graph sketches and iteratively sketches them into an adaptive cluster model by leveraging the system's streaming feature.

Unicorn uses a graph processing framework called GraphChi [34] to process the large graphs and feed the data into the classification model. GraphChi allows Unicorn to



Node	Label
N1	1
N2	2,1
N3	3,1
N4	4,3,2
N5	5,4

Figure 2.2: Provenance Graph with associated labels. Labels are constructed by aggregating the unique node labels of its K-nearest predecessors. $K = 2$.

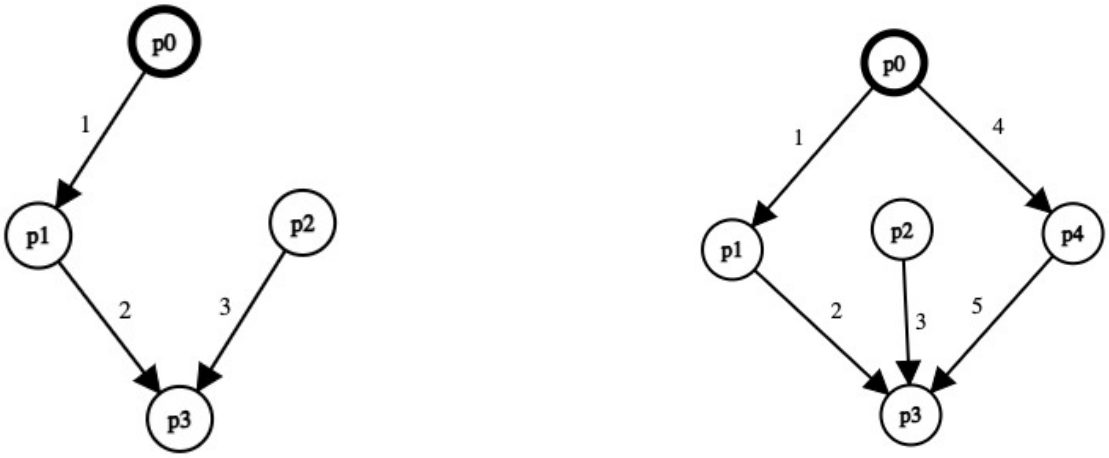
utilise Parallel Sliding Windows (PSW) to split the graphs into parallel computed sub-graphs, reduce the number of per-node computations, and decrease the required memory space.

The following five subsections outline a detailed description of the algorithmic process in the Unicorn Prov-HIDS. The information is a combination of details found in the original paper and code analysis of their implementation. We encourage the reader to examine the original paper and the git repository for exact details [1], [35].

2.7.1 Build the Provenance Graph (Step 1)

As previously mentioned, the system receives the data via a stream where each object represents a directed edge. The edge is added to the complete provenance graph, and adjacent vertices are updated. When vertices are updated, they will re-calculate their associated `label` as an aggregated chain of the unique labels from the K-nearest incoming edges and vertices, where K includes the current vertex. The labels are then sorted by their incoming timestamp to follow a chronological order. See Figure 2.2 for an example of node labelling where the label is aggregated from the $K = 2$ nearest predecessors, similar to how Unicorn handles its labels.

The streamed graph is assumed to be a DAG, although this is not true in practice, as discussed in Section 2.1. Seeing the streamed graph as a DAG is crucial to avoid infinite update loops. The ordered streamed DAG structure ensures that all incoming edges required to generate the label for the next periodic sketch are available in the stream prior to its creation. For illustrative purposes, consider the graphs in Figure 2.3 where an ordered streamed graph is presented at two different time intervals with $K = 2$, where each vertex represents a process. At time interval $t = 3$, the label for process p_3 will be an aggregation of labels from p_1, p_2, p_3, whereas its label at time instance $t = 5$, the aggregation will instead consist of labels from p_1, p_2, p_3, p_4. In the



(a) t=3 (b) t=5

Figure 2.3: DAG structure at set time instances, $K = 2$.

latter, it does not matter that the edge $(p_4 \rightarrow p_3)$ is not present at the earlier time interval $(t = 3)$ since it is not a cause to p_3 at that specific time. However, since p_4 will cause p_3 to execute again at $t = 5$ it will trigger an update in p_3 when it is added to adjust its label in upcoming sketches. This allows the system to systematically update the provenance graph and relevant labels within the K-hop distance without requiring knowledge of the complete graph from the start.

2.7.2 Convert to Histogram (Step 2)

After the labels have been computed, the histogram is updated accordingly, where each index in the histogram corresponds to an aggregated label from the provenance graph. Similar to how only the surrounding affected labels need to be updated for each new incoming edge in the stream, only the impacted histogram labels require updating at any given moment. In practice, this update occurs simultaneously with the label updates, allowing each vertex to adjust its histogram value as soon as all its incoming edges are recalculated.

As the labels are updated in the histogram, the old labels are not directly removed. Instead, Unicorn utilises a discount factor (denoted λ) to re-calculate the values for the stale labels in the histogram [1]. This enables the system to retain information from previous states within a reasonable time frame, determined by the λ value, while allowing earlier states to gradually decay over time. Study Figure 2.4 as an example where the histogram counts the number of incoming edges to each node. The values in `Histogram 1` are decayed by a factor of 0.5 before the new edges are counted in `Histogram 2`. The decay function in Unicorn is a bit more sophisticated, where they decay the histogram values by a factor of $e^{-\lambda \Delta t}$, where Δt is the time delta since the last decay. However, the main idea is the same.

Unicorn features another key mechanic when updating the histogram. This is that the label has a predefined maximum length set by the `Chunk Size` variable. This

means that if the vertex has a larger number of adjacent incoming edges within the chosen K-hop distance, its label will be split into multiple histogram entries. A single highly connected vertex can thus affect multiple histogram values at once.

2.7.3 Create Sketches (Step 3)

The histograms are periodically transformed into sketches to reduce memory usage and increase detection speed. This is achieved by employing a HistoSketch (see Section 2.5), which transforms histograms to similarity-preserved fixed-sized sketches [27]. The HistoSketch system utilises a normalised min-max algorithm to preserve the Jaccardian Similarity [36] between sketches rather than counting the absolute column values in the histogram. In other words, histograms with similar distributions over their columns will have more overlapping values in their sketch compared to dissimilar histograms. These sketches are stored on disk and are later used to compare sketches seen during training and those created when the system is in use.

2.7.4 Clustering (Step 4)

The sketches from Step 3 are then clustered with a clustering algorithm similar to K-Means [37] described as K-Medoids [38]. Two main distinctions between these clustering methods make K-Medoids more suitable in this system. The first one is that the centre of each cluster is not based on the cluster's true mean value, as in K-means. Instead, the centre of each cluster is the position of its mathematical centre's nearest label. This enables each cluster to be distinctly linked to a unique label, enhancing the clarity of its content. The second distinction is that K-Medoids

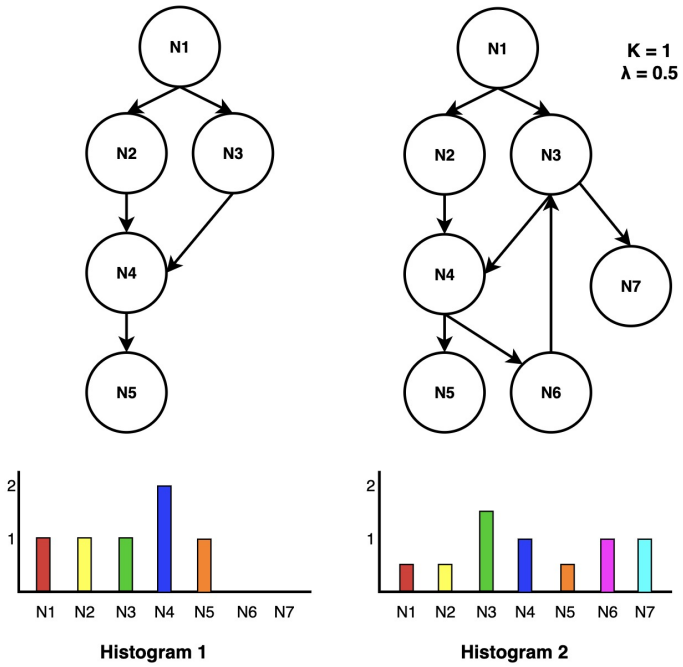


Figure 2.4: Histogram decay over time, $K = 1, \lambda = 0.5$.

enable other similarity metrics that are not supported by K-Means. For example, Unicorn uses the Hamming Distance as its similarity metric when clustering with the K-Medoids algorithm [39].

The clustering is then used during both the training and testing phases, and since Unicorn is streaming-based; it allows the model to track the transitions of the underlying system between the identified clusters and construct an evolutionary model during training [1]. The clustering will thus provide the model with two key data points: the sketch's cluster position and the cluster transitions. The cluster transitions are also presented as an evolution in the original paper, see Figure 2.5 for a visualisation of cluster evolutions. Logically, this should be thought of as the system execution starts in a cluster (c_1). When the system's behaviour evolves, it will transition from the current cluster to the cluster associated with the next evolution (c_2). This allows the system to track whether the underlying system follows a previously seen execution path and if the system deviates from the current cluster.

2.7.5 Anomaly Detection (Step 5)

Unicorn utilises the data points generated during clustering (Step 4) in its training phase to analyse each newly created sketch during runtime. It evaluates whether the sketch aligns with an existing cluster by measuring its proximity to previously seen clusters using the Hamming Distance. Unicorn then checks for any unknown state transitions between the new sketch and its previous evolution. Based on this analysis, Unicorn determines whether the sketch is benign or malicious.

To determine if a sketch belongs to a cluster, the model applies a threshold based on its deviation from the cluster's centre, using either the mean or maximum deviation between sketches within the cluster. The similarity between the tested sketch and the cluster is not allowed to exceed the mean or maximum distance plus a specified number of standard deviations. This approach is not explicitly mentioned in the original paper but can be inferred from the algorithmic implementation published in their project repository [35]. We formally define the detection algorithm as follows.

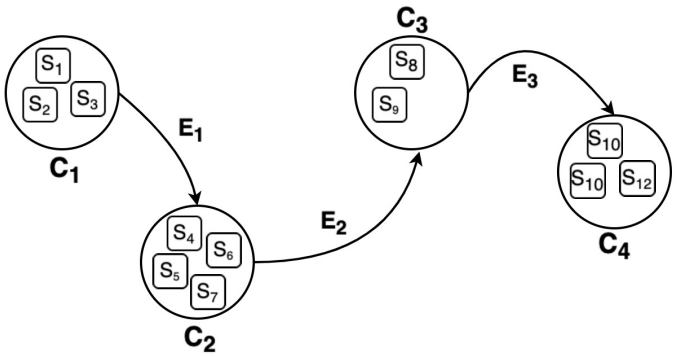


Figure 2.5: Clusters and Evolutions, C Denotes Clusters and E the Evolutions that Govern how the Model Transitions between clusters.

Let s_x denote a newly created sketch seen during runtime, and c_e represent the cluster c in evolution e. C is the set of all clusters seen during training associated with their specific evolution, such that all $c_e \in C$. If equation 2.1 does not hold for $c_e \vee c_{e+1}$, it follows that s_x is marked as malicious. In the equations, D denotes the distance between sketch and cluster, T is the threshold value, c_μ and c_{max} is the mean and maximum sketch deviation within a cluster, and σ is the standard deviation.

$$D(s_x, c_e) \leq T \tag{2.1}$$

$$T = \begin{cases} c_\mu + n \times \sigma & \text{if metric = "mean"} \\ c_{max} + n \times \sigma & \text{if metric = "max"} \end{cases} \tag{2.2}$$

In other words, the algorithm tries to match the current sketch (s_x) with the current cluster (c_e). If it deviates more than the allowed threshold (T), it compares the sketch to the cluster in the next evolution (c_{e+1}).

2.7.6 Structure and Parameters

The system is structured into three parts, all of which can run separately for testing purposes. The parts are the `Parser`, `Analyzer` and `Modeller`. The Parser transforms the data from the system reader's format into the Unicorn-specific format. The Parser has built-in functionality to read from system-integrated data collectors such as CamFlow [15] and directly from datasets such as the StreamSpot dataset [7]. The parsed data can then be streamed to the Analyzer.

The Analyzer then builds the graph (Step 1), histogram (Step 2), and sketches (Step 3). The sketches seen during training will be saved to files to act as a basis for detection when the system is in use.

Finally, the Modeller will then use these saved sketches to create clusters and evolutions to compare against the sketches seen during deployment. While Unicorn works on streamed graphs, it performs interval batching to increase efficiency. The batching size can be set at runtime and is often set to match the batch sizes received from the data collector. For example, CamFlow utilises a batch size of 6000. If Unicorn is set to read data from CamFlow, its internal batching size is often set to match that value.

A summarised list of the build and runtime parameters, together with their purpose and recommended settings, can be seen in Table 2.1.

Parameter	Value	Explanation
K	3	Neighbourhood size
Sketch Size	2000	Floating point values per sketch
Sketch Intervals	3000	Edges streamed between sketch generation
Chunk Size	50	Max labels per histogram column
Decay Factor (λ)	0.02	Histogram column decay factor
Batch Size	6000	Edges processed per batch

Table 2.1: Recommended Parameters for Unicorn.

2.8 R-CAID

R-CAID (2024) [3] is a Prov-IDS[2], and the first system to combine Prov-IDS with Root Cause Analysis (see Section 2.4). R-CAID uses fine-grained provenance graphs and RCA, i.e., it connects the root cause on the node or process level instead of per graph. Simultaneously, it uses the RCA connection during the classification at each node to decide if it is benign or malicious. This technique gives the system better precision compared to a coarse-grained system. However, it comes at the expense of significant memory usage. The node classification method also implies that the R-CAID system is not designed to find complete system threats but rather unusual process nodes. This difference makes the results of the R-CAID system difficult to compare to coarse-grained Prov-HIDS models, as is stated in the paper [3].

The RCA part of the system will increase the efficiency of the system admin or analyst. It does this by automatically exhibiting the root cause of the attack, i.e., a forked process of Firefox, instead of forcing the admin to backtrack the system calls manually to find the root cause.

R-CAID does not support data collection by continuous unbounded data streams but instead utilises static graphs. The authors of R-CAID mention that the system would benefit from a transition to use data streams instead for continuous analysis, as Unicorn does [1], [3]. The R-CAID system is also claimed to be resistant to mimicry attacks (see Section 2.2.2). There is currently no research that has disproven this claim.

[2]R-CAID is not directly specified as a HIDS

2.9 Datasets

Three datasets were used in this thesis project: the StreamSpot dataset [2], [7], [8], the DARPA Transparent Computing Engagement 3 dataset [9]–[11], and the ATLASv2 dataset [12]–[14]. The following section describes them.

2.9.1 StreamSpot

The StreamSpot dataset [2], [7], [8] was created by E. Manzoor *et al.* in 2016, where they proposed the Prov-HIDS StreamSpot [2]. It has later been used in several follow-up papers [1], [3], [5], [6].

The dataset consists of system calls from five benign web browsing scenarios and one attack. All data in the set is entirely task-specific, i.e., the logs only contain behaviour from the specific application and no system noise. The benign scenarios are: browsing CNN, watching YouTube, downloading files, checking Gmail, and playing a video game. The attack scenarios, however, are drive-by-download attacks, with added mimicry behaviour, that gain access to root privileges. Each of the benign scenarios represents 100 performed tasks. The data is split into three sub-datasets. Table 2.2 shows the sub-datasets, their scenarios, number of graphs, vertices and edges, and the number of attack edges.

2.9.2 DARPA Transparent Computing Engagement 3

Transparent Computing (TC) is an initiative from The Defense Advanced Research Projects Agency (DARPA), a United States Department of Defense subsidiary. The goal of the initiative was to make black box operating systems transparent and to contribute to research within the area [9]–[11].

The initiative was performed in five different engagements; the third engagement (used in this thesis) consists of the following five Technical Areas (TAs).

1. **TA1: Tagging and Tracking.** This step is where the different processes and events are linked. Five teams prepared five different datasets: Cadets, Clearscope, Five Directions, Theia, and Trace. Theia is one of the main dataset

Dataset	**Scenarios**	**|Graphs|**	$\|\bar{\mathbf{V}}\|$	$\|\bar{\mathbf{E}}\|$	**|Test E|**
D1	YouTube, Download Files & CNN	300	8705	239,648	21,857,899
D2	Gmail, Video Game & CNN	300	8151	148414	12,854,229
D3	YouTube, Download Files, Gmail, Video Game & Microsoft Office	500	8315	173857	24,826,556

Table 2.2: The contents of the sub-datasets; the size of the training scenarios and the number of test edges, i.e., the attack plus 25% of the benign graphs.

used by X. Han *et al.* [1], A. Goyal *et al.*[3], and E. Manzoor *et al.* [8], in their research.

2. **TA2: Detection and Policy Enforcement.** In this TA, the project teams constructed systems that linked the events created in TA1. They then used the graphs to determine which activity was part of an APT. The project teams are undisclosed.

3. **TA3: Architecture.** The project teams for this TA are undisclosed, but their objective was to link TA1 and 2 on the same system.

4. **TA4: Scenario Development.** Once again, the project team for the TA was anonymous, but their work consisted of creating scenarios for all types of APT activities.

5. **TA5.1: Adversarial Challenge Team.** This task was the actual attack. The undisclosed team created systems to perform and open up for evaluation of the TC.

The Theia sub-dataset created under the TA1 objective will be used for testing during this thesis, as it provides a good foundation to compare against due to its earlier use in similar research. Theia had four published attacks. In the first attack, the adversary wrote to the disk via a Firefox backdoor. The second also wrote to the disk but via a browser extension. The third attack used phishing emails with a malicious link, and the last did the same but with an executable attachment. A more thorough description of the dataset has been released on GitHub and Bitbucket [10], [13]. The data, operational log and the ground truth can be found on Google Drive provided by Five Directions, Inc. [9].

2.9.3 ATLASv2

The dataset ATLAS was created by A. Alsaheel *et al.* [40], [41] in an attempt to create a publicly available dataset that contains audit logs from benign and attack behaviour. The dataset was created in a controlled lab environment with selected activities performed during working hours.

The ATLASv2 dataset was a later extension of ATLAS, developed one year later (2023) [12]–[14]. This new dataset used the same attacks but aimed to improve the quality of the benign activity by removing the emulated benign activity with real activity. The new benign activity was carried out by recording the normal work activity on two workstations used by two researchers during a period of four days, including the stations' idle state during off-working hours. A fifth day was used to run ten attacks sequentially.

The ten attacks are split between two different machines. The first has four single-host attacks, that is, attacks that only target that specific machine. It also has six multi-host attacks, which start at host one and then spread to host two. The latter machine is only targeted via those six attacks. The performed attacks have a Common Vulnerabilities and Exposures (CVE) ID [42] and a rating in the Common Vulnerability Scoring System (CVSS) [43] scale; these can be seen in Table 2.3.

Attack ID	CVE Code	Exploited Application	CVSS (0-10)
S1	2015-5122	Adobe Flash	9.8
S2	2015-3105	Adobe Flash	10
S3	2017-11882	Microsoft Office	7.8
S4	2017-0199	Microsoft Office	7.8
M1	2015-5122	Adobe Flash	9.8
M2	2015-5119	Adobe Flash	9.8
M3	2015-3105	Adobe Flash	10
M4	2018-8174	Microsoft Office	7.5
M5	2017-0199	Microsoft Office	7.8
M6	2017-11882	Microsoft Office	7.8

Table 2.3: File in each data subset.

Another addition by the researchers for ATLASv2 added, was additional logging programs, such as Carbon Black Cloud - Endpoint Detection and Response; these logs were used for this thesis.

2.10 Related Work

The following section provides a deeper knowledge of the previous work that laid the foundation for this thesis.

2.10.1 Mimicry Attacks

The first researchers to scratch the idea of Mimicry Attacks were T. H. Ptacek and T. Newsham [44]. Three years later, D. Wagner and D. Dean coined the term mimicry attacks [45], and D. Wagner and P. Soto solidified it with their previously mentioned paper [4]. In the context of this thesis, the mimicry attacks are also represented as APTs. This definition comes from the baseline system Unicorn [1], which is explicitly specified as a "Runtime Provenance-Based Detector for Advanced Persistent Threats". Another system that also labels their mimicry attacks as APTs is the R-CAID system [3].

2.10.2 Prov-HIDS

There are several previous IDSs and Prov-HIDS that have been of importance to this thesis [1]–[3], [46], [47]. Many of them utilise Provenance Graphs and are host-based. The HIDS Unicorn [1] is the base system the new one will extend, and R-CAID [3] is the system on which the RCA will be based. These systems differ in their approach, the R-CAID system looks at a fine-grained level (node/processes) instead of the coarse-grained (graph) level that Unicorn uses. However, Unicorn does not incorporate RCA, which R-CAID does. Unifying coarse-grained systems with RCA is a new approach, and that is a research gap our thesis fills.

2.10.3 Benchmarks

The baseline for benchmarks used in this thesis is reproductions of tests from previous papers in the same area [1]–[3], [5]. This includes the datasets StreamSpot [2], [7], [8], Darpa Theia E3, [9]–[11], and the newer dataset called ATLASv2 [12]–[14], which has only been used in one previous paper [3].

3

Problem Description

In this chapter, we start by giving insights on the current vulnerabilities (Section 3.1). We then discuss the approach and challenges in reproducing the mimicry attacks (Section 3.2. Finally, we conclude by describing the problems in the datasets and their structure.

3.1 Current Vulnerability

The exploits developed by A. Goyal *et al.* consist of three mimicry attack techniques to circumvent detection against a variety of Prov-HIDS [5].

The first technique masks the malicious operations by including a large amount of system call chains that have been identified as frequently occurring chains of benign system calls. Since the structures are frequently occurring, they are likely to be present in the training data for the Prov-HIDS, which makes the complete chain seem like normal operations. This allows the attacker to mimic normal behaviour by incorporating a large number of benign system calls in the attack, skewing the data model created over the attack to be more similar to the ones seen during training. To achieve this, the original operations needed for the attack will be executed with their normal intent and the system calls that are only incorporated to match the targeted call structure will be executed as No-ops.

The second exploit not only tries to add known benign substructures to the attack graph, but it also tries to match the correct distribution of system calls that are expected to be observed by the Prov-HIDS. This is done by adding benign sub-structures of system calls to the attack, where the structures must at least match the length and distribution of the sub-structures observed by the Prov-HIDS (K-length in Unicorn). In reality, this means that if an adversary tries to exploit a vulnerability in Program A, the adversary would analyse how that specific program operates, what system calls it makes, and how they are distributed. When this is known, the adversary would subsequently add system calls to its attack to match the benign system call distribution. This leads to the provenance graph created by the attack to have a very similar distribution of nodes and edges as observed during regular operations. This exploit directly targets detection systems similar to Unicorn that utilise normalisation (as described in Section 2.7.3) in the detection model. It targets these systems since the attack must deviate enough from the expected behaviour after normalisation to be flagged as malicious.

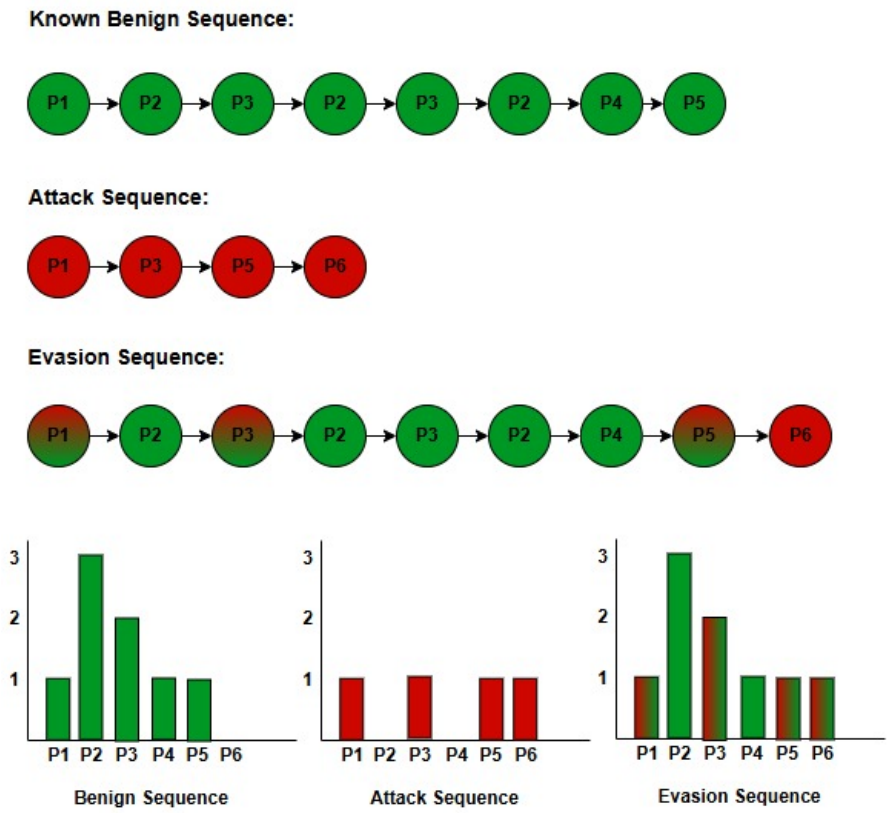


Figure 3.1: Example of How an Adversary Can Modify an Attack to Mimic the Normal Behaviour of the System.

The effect on how a Prov-HIDS sees an attack that has been modified with these techniques can differ greatly from how the attack would normally look. For instance, take the simple example in Figure 3.1, where the benign system call operations are known. The adversary then exploits the known operations to incorporate non-malicious behaviour into its attack as no-ops to nudge the histogram created over the attack closer to the histogram of the known benign behaviour. This is done in an attempt to make the deployed Prov-HIDS misclassify the attack as a benign operation. The theoretical idea for these two exploits mentioned above is the same as the example in Figure 3.1. However, when used in reality, it is applied to a far greater scale since both benign and malicious behaviour are more complex than a few system calls. Subsequently, these additions to the attack scripts can consist of thousands of operations.

The last exploit found by A. Goyal *et al.* was against Prov-HIDS, which only considers unusual behaviour, such as ProvDetector [6]. This means that the system will only keep track of behaviours that occur rarely. This is motivated by the assumption that malicious behaviour deviates significantly from benign behaviour and will thus appear as a rare occasion in the provenance graph. It is thus enough to compare unusual behaviour against previously seen unusual behaviours to see if it has previously been marked as benign. The exploit utilises this by adding these rare benign behaviours to "flush out" the malicious ones during classification. However, this exploit is not viable

against Prov-HIDS that encodes the entire system structure, such as Unicorn [1]. To our knowledge, complete system encoding is more common in Prov-HIDS. Our focus will thus be on the first two exploits since they are viable against Prov-HIDS with full graph embeddings, which is the same type of system that will be built upon for testing in this project. For further details on all of the above exploit techniques, we recommend the reader to read their original report [5].

3.2 Mimicry Attack Reproduction

A. Goyal *et al.* presented a modified attack against the Unicorn system that combines the first two exploits discussed in the previous section into a single attack. Their findings were validated, in this thesis, against this combined attack. This validation was done by training the system on the benign data from the StreamSpot and Theia datasets and perform tests against the regular attacks and the modified version that avoids detection, creating a mimicry attack. The mimicry attacks for the StreamSpot dataset are publicly available and can be found in this repository [48].

There are no published mimicry attacks for the Theia dataset. However, the existing attacks in the dataset can be modified into mimicry attacks by applying the second technique discussed in Section 3.1 and originally presented by A. Goyal *et al.* [5]. This transformation requires identifying the log-line where the attack begins, which is documented alongside the dataset, and knowledge of the benign substructures that are connected to the exploited processes that appear before the attack starts. This knowledge can be inferred by studying the log files. Using this information, the entire system-call attack chain can be altered to incorporate these benign substructures, making the attack resemble normal behaviour. For example, if the attack originates in the browser, the log-line would correspond to the line where the connection to the malicious host is made. The benign substructures are the system calls that have been observed to be performed earlier by the browser processes during normal execution. These system calls can then be appended in-between the malicious instructions to make it look as if the overtaken process continues its execution as normal.

The re-created attacks will act as a baseline that the system extension developed in this project can be evaluated against. The tests in the thesis were completed against the Unicorn system with the recommended parameters displayed in Table 4.2, except for the batch size. These are the same parameters used in the tests performed in the paper where the evasion techniques were first presented [5]. The reproduction of the test shows similar results to the original findings. Tests done on the StreamSpot data, the benign and attack sets, shows a detection rate of around 98-99%, as expected. All regular attacks are detected when tested on the smaller Theia dataset, but all mimicry attacks evade detection. Both datasets display a recall and precision of zero against the mimicry attacks. This indicates that all mimicry attacks are misclassified.

An observant reader may notice that the standard deviation used during classification differs between the datasets, as shown in Tables 3.1 and 3.2. While this may look unusual at first, it is easily explained by the fact that different systems display varying degrees of executional similarity. Systems with greater variability in their

operations often require a higher standard deviation threshold to accommodate for the diversity within clusters computed during training. The standard deviation used in this thesis was selected based on the values that most accurately reproduce the results presented in the earlier study by A. Goyal *et al.* [5]. This ensures that the baseline is as accurate as possible when assessing the impact of RCA.

3.3 Datasets

The following section will describe the structure and problems of the three data sets used in this thesis: the StreamSpot dataset [2], [7], [8], the DARPA Transparent Computing Engagement 3 dataset [9]–[11], and the ATLASv2 dataset [12]–[14].

3.3.1 Dataset Problems

All of the datasets had some problems. The StreamSpot dataset is the oldest dataset and the only one that provides native mimicry attacks, also offering a clear separation between test and training data. What StreamSpot lacks, however, is logs of real system behaviour. The dataset only contains task-specific logs, and it lacks real system noise. This means that the data is simulated and might not replicate a true real-world scenario [2], [7], [8],.

The second dataset, Theia E3, is emulated. This dataset, however, was created under more realistic conditions. The researchers logged system noise, but it was chosen system noise and was created under supervision. This data set also lacked native mimicry attacks, meaning that they had to be created in this project [9]–[11].

StreamSpot	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Benign	0.987	-	-	-
Attack	0.995	0.995	1	0.995
Mimicry	0.134	0.0	0.0	0.0

Table 3.1: StreamSpot Baseline, $T = c_\mu + 3\sigma$.

Theia E3	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Benign	-	-	-	-
Attack	1.0	1.0	1.0	1.0
Mimicry	0.0	0.0	0.0	0.0

Table 3.2: Theia E3 Baseline, $T = c_\mu + 3.4\sigma$.

ATLASv2	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Benign	1	-	-	-
Attack	0.888	1	0.875	0.881
Mimicry	0.0	0.0	0.0	0.0

Table 3.3: ATLASv2 Baseline, $T = c_\mu + 2.6\sigma$.

Dataset	Training Benign	Test Benign	Attack	Mimicry
StreamSpot	74	24	99	92
Theia E3	2	0	4	3
ATLASv2 Original	2	0	10	0
ALTASv2 Modified	12	4	10	4

Table 3.4: File in each data subset.

Lastly, the final data set used in the thesis, ATLASv2, is derived from the earlier version called ATLAS. Both of these datasets came with shortcomings. The attacks used in both datasets are very similar to each other and derive from the same or closely related exploits. Therefore, this dataset can be too niche for a general training dataset [12], and the emulated benign behaviour in ATLAS can make the benign behaviour niche as well.

There is also another problem that exists for both Theia E3 and ATLASv2. The problem is that even though the first four days were believed to be completely benign, there is no way to know if the workstations were under external attack unbeknownst to the researchers [10], [12]–[14].

3.3.2 Dataset Structure

The datasets are saved as batches in different files. Some files contain only benign behaviour (as far as the creators know), while other files contain a mix of malicious and non-malicious behaviour. The benign behaviour is further split into a training and a test set, where the test set contains benign behaviour that the system has not seen during training. Furthermore, the malicious behaviour is split into two subsets: attacks and mimicry attacks. These subsets will be referred to as `Training Benign`, `Test Benign`, `Attack` and `Mimicry`.

The file split between each subset can be found in Table 3.4. Both the Theia E3 and ATLASv2 datasets contain far fewer files compared to the StreamSpot dataset. This will lead to less granularity for the results of these datasets, which must be considered when comparing the findings of this thesis. Furthermore, it is important to note that the Theia E3 and ATLASv2 datasets have no files in the `Test Benign` subset. This will translate to a precision score of 1 across all tests where any malicious files are found, since there is no possibility for the file to be labelled as a false positive. The benign date time stamp-ordered ATLASv2 data was, thus, split into 8 different files to allow both training and test data, while the limited size of the Theia E3 dataset did not allow for such a split.

The datasets need to be converted to a Unicorn-compatible format. The conversion is done by a set of parsers, where the StreamSpot and Theia E3 parsers already exist as a feature of the Unicorn Prov-HIDS system. The ATLASv2 parser, however, is built specifically for this thesis and presented in more detail in Section 5.5. The StreamSpot and Theia E3 parsers are located in the Unicorn-Parsers repository [35], and the ATLASv2 parser is in this project's parsers repository [49].

4

Method

This Chapter details the procedures used to address the previously stated challenges and how to achieve the goals stated in Section 1.3. The high-level characteristics of the RCA system developed in this thesis are presented in Section 4.3, followed by the evaluation metrics in Section 4.4 and the experimental setup in Section 4.5.

4.1 Motivation

When current coarse-grained Prov-HIDS's analyse the system operations, it often puts little emphasis on what processes made the operations occur, i.e., its roots. This design choice becomes problematic when encountering a mimicry attack since normal operational structures are expected during such an attack, which allows it to be indistinguishable from normal behaviour. However, these operations may spawn from unusual roots, which can be detected with root-cause analysis.

If the incorporation of RCA should have any potential to improve the model's detection accuracy, it would require that roots associated with malicious operations are distinguishable from those linked to benign behaviour. If the roots associated with malicious behaviours (R_M) share a completely overlapping set of roots with benign behaviour (R_B), such that $R_M \subset R_B$, then analysing roots in such a system would provide no meaningful indication of whether the observed behaviour is normal or anomalous.

To assess whether RCA can enhance the detection algorithm's ability to differentiate behaviours, the sets of roots associated with the different behaviours in the datasets were analysed. This was done by constructing sets of all vertices that only have outgoing edges or that have at least one outgoing edge that was created before its earliest incoming edge. The corresponding root counts are presented in Table 4.1.

As shown in Table 4.1, the malicious sets contain roots that do not exist in the benign root set. This indicates that RCA could help to detect attacks if these unique malicious roots can be correctly identified in the detection model. There is a concern regarding the absence of roots in the StreamSpot dataset. This is because the StreamSpot dataset contains data from a single type of task, a specific user activity in a web browser. As a result, the dataset lacks operations typically performed by the operating system and background activities. In contrast, the Theia E3 and ATLASv2 datasets may be considered more representative for testing a system-wide

Dataset	Benign Roots (R_B)	Malicious Roots (R_M)	($R_B \cup R_M$)
StreamSpot	1	2	1
Theia E3	2224	1475	738
ATLASv2	45	41	36

Table 4.1: Roots Across Benign and Malicious Sets in StreamSpot, Theia E3, and ATLASv2 Datasets.

Prov-HIDS, as they contain a broader range of system operations. However, the data in Table 4.1 raises a concern that the roots between subsets from the same dataset deviate too much to give any meaningful indication for a coarse-grained classification model. For example, the `Benign` and `Malicious` subsets from Theia E3 share few common roots. Classification problems related to this will have to be evaluated and will be discussed in Chapter 7.

It is difficult to say if unique roots are a characteristic of all types of attacks. However, the presence of such roots, as found in the malicious set, provides sufficient justification to incorporate and evaluate RCA in Prov-HIDS in an attempt to increase the detection rate of mimicry attacks. There is a possibility that if these roots can be associated with their following actions, they can be tracked throughout the provenance graph. From this association, it might be possible to find patterns of normal and malicious root behaviours that can be used to flag anomalous activity where other detection models fall short.

4.2 Root Behaviour

With the existence of unique roots for the benign and malicious datasets, as established in the previous section, we assume that two main root behaviours occur during a mimicry attack when observed on the provenance-graph level.

The first behaviour is that new roots, which are threat-specific, appear in the provenance graph during a mimicry attack. The second behaviour is that already established processes change their execution patterns, which makes them and their roots connect to new areas of the graph.

This expected behaviour, in combination with the knowledge that mimicry attacks produce a large amount of known benign graph structures (Section 3.1) should result in the root activity to change on the host-level. It is this change in root behaviour that the RCA system developed in this thesis uses as the basis for detection.

4.3 System Design

To evaluate the effectiveness of RCA in modern Prov-HIDS, a complete RCA detection system will be developed in this thesis. The system is designed to seamlessly integrate with existing Prov-HIDS frameworks, enhancing their capabilities by adding RCA as a complementary feature to the original model.

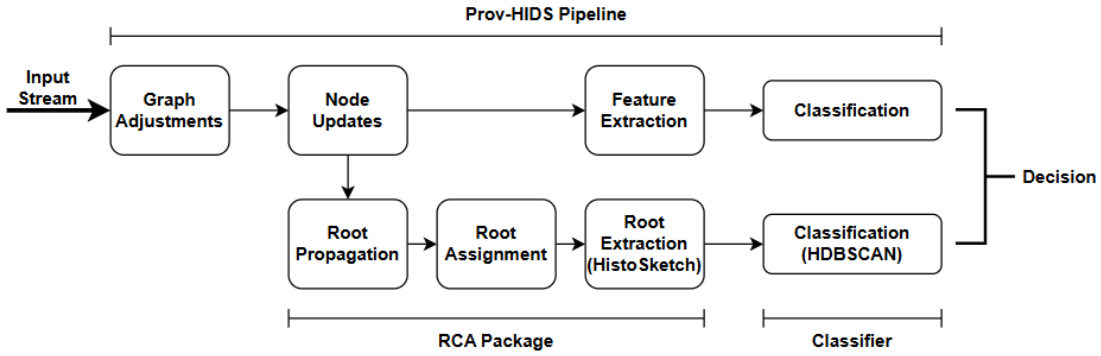


Figure 4.1: Prov-Hids and RCA Pipeline

The design of the detection system contains four main components: Root assignment, Root propagation, Root summarisation, and classification. These four components create a pipeline for root classification which can be integrated into the original Prov-HIDS and act as a complement to the already existing detection system, as seen in Figure 4.1.

The subsystem for handling root assignment and propagation is developed specifically for this thesis. Its primary purpose is to integrate seamlessly with the existing Prov-HIDS architecture by attaching to the functions responsible for graph and node updates. This limits the need for duplicated node iterations and enables greater flexibility in how the RCA system should be executed in the complete Prov-HIDS context.

For summarisation and classification, which is actively researched, existing solutions will be applied. The following sections will highlight design decisions made for all components of the complete RCA system.

4.3.1 Root Assignment

To determine whether a node is a root, a similar approach as in R-CAID has been adopted [3]. However, one major difference is that the root assignee developed in this project considers all nodes as possible roots, whereas the approach in R-CAID only considers process nodes. The decision to consider other nodes, such as files as possible roots is to incorporate a wider range of APT attacks. For instance, threats that originate from PDF files or similar files that dynamically interpret the contents and can execute built-in functions should be regarded as potential roots. [50]. In this system, a node is classified as a root if it adheres to one or more of the following principles:

1. The node has only outgoing edges.
2. The node's earliest outgoing edge has a timestamp that is earlier than its earliest incoming edge.

In a streaming context, where edges are streamed directly from the system recorder,

all edges will be seen in the order they are executed in. This means that if the system contains roots with both incoming and outgoing edges, the first edge that will be processed must be an outgoing one. Otherwise, the node will not be a root node. The latter classification can thus be considered unnecessary in a streaming system that handles edges as they are created. However, the definition is kept to allow for classification in systems that process edges in batches or parallel batches. In such systems, new edges may be grouped and only processed at set intervals where changes in node processing order are expected for optimisation purposes.

4.3.2 Root Propagation

For root propagation, individual nodes will read the roots from their incoming edges, with the assumption that they are updated. The node will then store the information of relevant roots, where a root is considered relevant if it was created before the currently updating node. Lastly, the node will propagate this information to its outgoing edges by enabling the information of individual roots to edges that were created after the root. This prevents new roots from being assigned to older edges, ensuring that systems that process the graph in batches do not propagate roots to already visited edges. For example, consider a system where node R_1 serves as a root for process P_1. This process generates a set of outgoing edges to other processes it triggers, propagating R_1 along all those edges. If a new root, R_2, is created that also calls process P_1 but does not trigger any of the processes connected by those outgoing edges, then R_2 is completely separate from P_1's existing connected processes. Therefore, R_2 should not be propagated along those outgoing edges.

4.3.3 Root Summarisation

The similarity preserving histogram and sketch scheme HistoSketch is used to store a summarisation of the complete root activity of the host [27]. The system is built around a histogram, which is continuously updated each time a node is processed in the stream. The histogram stores the IDs of the seen roots. When a node updates, each root associated with the node is incremented by one in the histogram.

At set intervals of 3000 updated nodes, the histogram is transformed into a fixed-sized label sketch with size 200. This is achieved by applying the algorithm proposed in the HistoSketch paper that adds a subset of the labels to the sketch based on the consistently weighted sampling scheme [29]. Simultaneously, the histogram values are decayed by a factor of 0.02. This intermediate decaying of the histogram values enables the system to weigh current activities as more important while not forgetting past events directly.

The sketch creation algorithm recreates the sample weight distribution found in the histogram as closely as possible while downsizing to fit the sketch size. To exemplify, if the sketch size is set to 30 and the root R_1 makes up 12% of the total count in the histogram. Then R_1 will occupy approximately 3-4 spaces in the sketch since each space represents approximately 3.33% of the histogram. This gives the range of 3-4 spaces a representation between 0.0999 and 0.1332, since $3 * 0.0333 \approx 0.0999$

and $4 * 0.0333 \approx 0.1332$. By utilising consistently weighted sampling, the system will ensure that the labels added to the sketch will occupy the same indices if a similar distribution of the label is seen in the histogram at another time. This ensures that the sketches are comparable to each other.

Both the decay factor of 0.02 and the sketch size of 200 have been found to effectively maintain the relevance of the histogram data and produce comparable sketches [27]. Higher sketch dimensionality displays a diminishing return in classification accuracy and could cause overfitting. Data with high dimensionality tends to display a behaviour where the relative difference between objects, sketches in this case, decreases as dimensionality increases, which causes sketches to appear more similar [51]. Since each label in the sketch represents a dimension during classification, limiting the sketch size to 200 decreases the possible dimensionality. This simplifies the classification of the sketches while providing enough granularity to represent most currently active and relevant root nodes in the system.

Similarly, a sketch interval of 3000 has been found to effectively maintain the relevance of the histogram data and produce comparable sketches [1]. A too-short interval results in overly similar sketches, while a larger interval causes sketches to become too distant, making them difficult to compare. Similarly, a larger decay factor increases time locality but decreases system context, and a smaller decay factor leads to an over-representation of context. Both scenarios have been observed to reduce the accuracy of classification models [27].

4.3.4 Root Classification

The assumption made to motivate the usage of a clustering model is that similar root activity will be observed over time. Meaning that if root R_x is usually observed simultaneously as root R_y, it is expected that a similar pattern will be seen in the future. Thus, making the root sketches appear in clusters, where each cluster represents a specific phase of the system's execution pattern.

Furthermore, it is expected that the root activity of a mimicry attack differs substantially from the normal behaviour since the malicious roots must mimic a large part of the normal behaviour. It is thus expected that roots related to a mimicry attack will create a noticeable chain of outlier sketches that share greater similarity with each other compared to the benign clusters.

The chosen clustering model for sketch comparison is the HDBSCAN model. This is a density-based clustering algorithm that merges clusters hierarchically [31]. The usage of a density-based clustering model limits the need to decide upon a fixed number of clusters, as is often required by other clustering models. This approach simplifies clustering and creates a better representation of the underlying system, as each cluster theoretically corresponds to a distinct phase of the system's execution. Small systems may only need a few clusters to capture all their states effectively, whereas larger and more complex systems may require a greater number of clusters to account for their variability. By not limiting the model with a fixed number of clusters, both small and large system states can be represented.

By choosing HDBSCAN over the regular DBSCAN, the need to set an exact cluster density can be avoided [32]. This further improves the clustering capabilities by allowing clusters of different densities in the same model. The motivation for this is that some clusters will be tightly coupled, where almost the exact same execution will occur every time it is entered, for example, when a system is idling. At the same time, other clusters may be more varied in their behaviour. The HDBSCAN model provides better control by preserving high-density clusters while still allowing low-density clusters to form rather than being classified as excessive noise.

While the model avoids the problem of having to define both a fixed set of clusters and the density of the clusters, two hyperparameters must be set before deployment. These are the `min_cluster_size` and `min_samples`. These parameters decide how small a cluster is allowed to be and what points in a cluster are considered as *core points*, which is later used during clustering to decide a point's distance to a cluster. Hyperparameter tuning was performed to find optimal values for these parameters. More details and results from the tuning are found in Section 6.1.

Since the sketches consist of root labels without numerical relationships to each other, common distance metrics such as the Euclidean Distance will not be appropriate to use. Therefore, the Hamming Distance will be utilised as the distance metric. This metric measures the distance between two sketches by calculating the required number of label substitutions needed for the compared sketches to match completely.

4.4 Evaluation Metrics

Three areas of metrics will be used to evaluate whether the goals and research question have been answered. The metrics are the following:

- **Model Performance:** Accuracy, precision, recall, and F1-score.
- **Model Quality:** Noise and cluster structure.
- **Computational Metric:** Space efficiency and execution time.

Accuracy, precision, recall, and the F1-score indicate how well the detection algorithm performs. Both in identifying anomalies and in assessing how often it incorrectly labels the data it encounters. The noise and cluster structure indicate how well the cluster model matches the data, and enable discussion on the chosen type of model and the achieved results. It may also be analysed during hyperparameter tuning.

The runtime performance, measured in execution time and memory usage, is compared between the RCA-enhanced system and the original Unicorn system. This combination of metrics enables a holistic analysis approach where all aspects of the chosen system implementation can be analysed and discussed thoroughly.

Parameter	Value
K	3
Sketch Size	2000
Sketch Interval	3000
Chunk Size	50
Decay Factor (λ)	0.02
Batch Size	500

Table 4.2: Unicorn Parameters Set During Testing.

4.5 Experimental Setup

Several tests were performed for both the Unicorn and RCA implementations. The tests to measure model performance and quality were achieved by running all datasets on the original Unicorn implementation, and the RCA-enhanced version. These results were then compared to evaluate the systems differences. To ensure consistency in processing, both systems received the data in the same order. Similarly, the split between the bending training and test data remained the same to ensure comparability of the systems.

A hyperparameter tuning was performed to decide upon the settings for the RCA system and the clustering model. This was achieved by running a grid search of possible combinations of parameter values and analysing their results in terms of clustering capabilities and the noise level in the model. The tuning was performed against the benign data in the Theia E3 dataset. No malicious data was used for comparison during tuning to limit the bias of the final model.

The execution time and space efficiency tests were performed on the ATLASv2 dataset, specifically all six m1-6 attacks for the first host machine (H1). These attacks were concatenated into one sequential attack. The total real-world execution time for this attack is 5 hours and 28 minutes. The concatenated data was analysed inside a memory profiler [52] with all child processes included. The memory profiler was executed ten times, and the average value for each measured point was calculated in order to reduce variability and obtain more reliable results.

All tests in this thesis ran on a virtual machine (VM) inside a personal computer, a MacBook Pro 2019. The computer had 64GB of RAM, an 8-core 2.3 GHz Intel Core i9 CPU with a 16MB shared L3 cache, and an integrated Intel UHD Graphics 630 GPU. The VM was given 2 cores, 16GB of RAM and 200GB of storage.

The Unicorn specific parameters used during tests can be seen in Table 4.2. All parameters except for batch size are the recommended ones. The batch size is set to match the value in the previous paper, where the evasion technique was found to make our results comparable to their findings [5]. These settings were constant for all tests performed on both the original Unicorn implementation and the RCA-enhanced version. The Unicorn classifiers' standard deviation value, as discussed in Section 2.7.5, varies between the datasets. The chosen values are set to best match the results presented in earlier reports.

5

Implementation

The fundamental idea behind the implementation is to provide a simple interface for handling all calculations related to finding and tracking roots, along with the necessary data structures. This interface should work as a complement that can easily be added to node and edge objects in already existing detection algorithms for Prov-HIDS. The four components, as described in Section 4.3, are split into two sub-systems. These are the *RCA Package* and the *Classifier*. The implementation of the RCA Package handles the root assignment, propagation, and summarisation components, as these need to be integrated directly into the existing Prov-HIDS graph engine. The Classifier contains the HDBSCAN classification model.

To test the system and its detection capability, it will be integrated into Unicorn's detection algorithm [1]. Unicorn was chosen due to its desirable traits, including being streaming-based and requiring limited CPU and memory resources. The Unicorn system is discussed further in Section 2.7, while the original paper provides a more comprehensive discussion of its implementation and design choices [1]. The Unicorn system is open-source, enabling us to study it in depth and adjust it as needed.

The additions made to the Unicorn algorithm will be detailed in Section 5.4. The complete source code, with all the changes made to the Unicorn system and the RCA package, can be found in the analyser repository [53]. More details on the complete implementation are found in Chapter 5. The effectiveness of this implementation and the optimal parameter settings will be assessed through the tests conducted in this thesis.

The classification model is located in a separate repository [54] and is discussed further in Section 5.3.

5.1 RCA Package Implementation

The developed C++ package is designed to handle roots in a streaming-based system and has three main functionalities. These are to find the roots, propagate them throughout the provenance graph, and finally store the root representation in sketches. The sketches can subsequently be used to build a detection model and track anomalies in the data. Although the package is designed for graph-based systems, it does not include designated node or edge objects. This design choice was intentional to

increase the flexibility of the root handling package, allowing it to integrate with a variety of systems that likely already have node and edge objects in place. Instead, the included data objects are intended to be integrated into these already existing nodes and edges to complement the existing detection model.

Additionally, the package is designed to support multiple root assignments per node. To accommodate this, all root interface functions that compare or propagate roots between nodes are built to handle arrays of roots rather than single root objects. For simplicity, it is recommended that roots be stored in arrays at the node and edge objects to accompany the expected in-/output of RCA-package functions. The size of the arrays is set during class initialisation by the `ROOTS` (R) parameter.

5.1.1 Root Data Objects

The package includes two main data objects, as seen in Figure 5.1, that store required information about the node and system roots. The key node information, stored in `NodeInfo` is the `node_id` and the `node_ts`, which stores the ID associated with the current node and the timestamp for when the node was first observed in the system. The `NodeInfo` object also contains a helper variable, `checkedIfRoot`, which tracks whether the current node has been tested to determine if it is a root node.

The `Root` object is used to contain and track the necessary information about each root in the system. This data can be assigned as a single object to a node or stored in lists, depending on whether the system allows for single or multiple roots connected to each node. The stored data is the unique root ID, the order in which the roots were assigned, and the timestamp for when the root was created. All numerical values are initialised to zero to indicate that they have not been set.

While these two objects contain very similar data points, they can be easily separated by the notion that `NodeInfo` strictly contains data related to the current node. This information is intended to be used for comparison in root assignment and propagation. In contrast, the `Root` object could instead contain information related to distant nodes that are considered roots to the current one.

```
struct NodeInfo {
    uint32_t node_id = 0;
    bool checkedIfRoot = false;
    unsinged long node_ts = 0;
};

struct Root {
    uint32_t root = 0;
    uint32_t order = 0;
    unsinged long tme = 0;
};
```

Figure 5.1: RCA Structures.

5.1.2 Assigning Roots and Root Propagation

To control if the current node is a root, the `isRoot` function is called. The function controls if any of the above-stated principles hold, and can be modelled by the pseudo-code in Figure 5.2. If the function returns `True`, the node is considered a root and can be appended to the current node's root list by calling the `updateRootOrderAndAddToRoots`. These two functions are intended to be called once during the first node visit and are thus combined into a single function in the package interface named `checkAndAssignRoot`.

If a node is found to be a root, it should be stored in the root array on the node object. All roots assigned to a node can then be propagated to nearby nodes by assigning the roots to its outgoing edges via the `updateOutedgeFromNode` function. The `updateOutedgeFromNode` function assigns the current node's roots only to outgoing edges created after the root.

```
Input: unsigned long in_edges_ts[], unsigned long out_edges_ts[]
Output: Boolean

smallestOut = MAX_VALUE
smallestIn = MAX_VALUE
in_size = size(in_edges_ts)
out_size = size(out_edges_ts)

// Find smallest in-edge timestamp
if  in_size > 0 {
    smallestIn = in_edges_ts[0]
    for i = 0 to in_size do
        if in_edges_ts[i] < smallestIn {
            smallestIn = in_edges_ts[i]
        }
    od
}

// Find smallest out-edge timestamp
if out_size > 0 {
    smallestOut = out_edges_ts[0]
    for i = 0 to out_size do
        if out_edges_ts[i] < smallestOut {
            smallestOut = out_edges_ts[i]
        }
    od
}

return (smallestOut <= smallestIn || in_size == 0 then)
```

Figure 5.2: Root Discovery.

Newly discovered roots will be propagated to the node's outgoing edges via the `updateOutedgeFromNode` function. Receiving nodes must collect these new roots and assign them to themselves. This is achieved with the `updateRootsFromInEdges` function. This function reads the roots from all incoming edges and stores the n most recent roots, where n is equal to the `ROOTS` parameter. The pseudo code for both update functions is available in Appendix A.1 and A.2.

5.1.3 Root Histogram and Sketches

The sketching system, as discussed in Section 4.3.3, is built around a histogram, which is continuously updated by calling the `updateHistogram` function each time a node is processed. When the `updateHistogram` function is called, each root associated with the currently processed node is incremented in the histogram.

The `updateHistogram` function is simultaneously responsible for decaying the decaying of histogram values. This is achieved by continuously tracking the number of function calls and decaying the values via a sub-function if the threshold is exceeded. After the decay routine is completed, a root sketch is created by calling the `createSketch` function. Here, the consistently weighted sampling scheme is applied, and the final root sketch is created as output.

To prevent race conditions on the globally updated histogram and threshold variables, both of them are protected by locking mechanisms. This can potentially limit the throughput in graph processing systems that can parallelise their work. However, it is necessary to accurately track the desired sketch and decay intervals.

5.2 Flow Chart

The flow chart in Figure 5.3 represents the intended function call chain for the public functions in the RCA package. While integrating an overarching helper method into the RCA package to handle the entire function chain could be beneficial, this has not been implemented to maintain flexibility for integration with other systems. This also enables intermediate object savings for systems that treat nodes and edges as shared resources that require intermediate locking of these objects. It would thus be possible to lock the in and out-edges separately, approximately halving the locked time per resource.

5.3 Sketch Clustering and Classification

To classify anomalies with the HDBSCAN model, as discussed in Section 4.3.4, newly observed sketches are compared to the existing clusters created during training and labelled as either part of a cluster or as noise. If labelled as noise, it is considered to be unusual behaviour. However, it is impractical to label all seen noise points as malicious activity since a small amount of outliers are likely to be present in the model regardless of malicious activity or not. Instead, a chosen threshold for the number of sequentially seen noise points is introduced, and the system is said to be

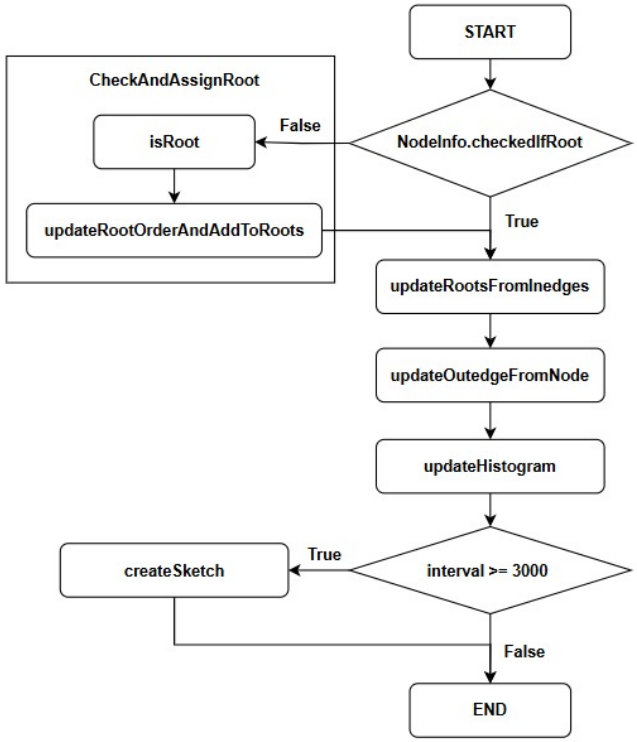


Figure 5.3: Intended Flow of RCA-Package Functions.

in a malicious state only if the classifier labels a consecutive set of new sketches that exceeds the threshold as noise. By utilising a threshold, we aim to limit the number of false positive alarms. This allows the system to transition between the different execution states related to each cluster without being labelled as malicious.

Furthermore, since it is expected that the sketches observed during a mimicry attack will be more similar to each other than the normal behaviour. These sketches can thus be interpreted as forming a new malicious cluster. By utilising the notion of malicious cluster formations, the threshold value is set to the same value as the minimum cluster size.

5.4 Incorporating RCA in Unicorn

The original Unicorn Analyzer [35] is structured around the `update` function found in the GraphChi engine [34]. This function processes each node that has at least one new edge associated with it in the stream. The processing is performed in parallel batches that do not produce overlapping neighbourhoods. The `update` function performs steps 1 to 3, which builds the provenance graph and performs all necessary calculations to produce sketches over the node's K-Neighbourhood, as discussed in Section 2.7.

To incorporate RCA into the already existing `update` function, a helper function was added that controls all interaction between the original Unicorn system and the

system for root handling. The helper function ensures that the system flow presented in Section 5.2 follows the intended flow to create accurate root sketches. The function is called each time a node is processed by the `update` function. When the node first gets processed, the helper function assigns all the values required by the `NodeInfo` struct, checks if the node is a root, assigns roots from its incoming edges, and then propagates roots to its outgoing edges. The `root_id` is assigned from a new field in the data input; this field is a hash of the `src_type`. The has is calculated before the source type is converted to a non-static integer in the parser. During processing, the function simply reads roots from the incoming edges and propagates them to the outgoing edges. The structure of the complete helper function is presented in Appendix A.3.

The sketches created by both Unicorn and the root packages are written to separate files and later used for classification. This is described in steps 4 and 5 for Unicorn in Section 2.7 and Section 4.3.4 for the root classifier.

5.5 ATLASv2 Parser

The ATLASv2 dataset [12]–[14] is the newest dataset of all used in the thesis. ATLASv2 was collected five years after Unicorn was built. It was therefore needed to create a parser from the ATLASv2 log format to a format that Unicorn can utilise. The dataset contained both network and system logs. The format chosen for conversion was system call logs through Carbon Black Cloud - Endpoint Detection and Response (cbc-edr).

The logs in the cbc-edr data was formatted as **jsonl** objects with 66 fields for each log line. The first field specifies the type of log, e.g., file modification or net connection operation. The log type determined which fields were populated in the object. These initial fields were mapped into eight different fields in the output format. The fields taken from the ATLASv2 data can be seen in Table 5.1.

The ATLASv2 logs are converted to the Unicorn data format which has two possible structures, `base` and `stream`. The base files of the Unicorn log have six columns, while the stream files have eight. However, a new column; `src_hash` has been added for both the base and stream rows, as seen in Figure 5.4. This new column is added to keep the root `src` value constant over several different files. The old Unicorn implementation would reset this value, which in turn would reset the root ID for every sketch. The root ID needs to be the same across all files in the training and test data in order for the root sketches to match.

All logs are sorted by date and time in ascending order. The logs line number is stored in the `count` column. All paths seen in `src` and `dst` are saved in a list, and converted to a sequential numerical value based on their appearances in the file. The two extra columns in the `stream` format indicate if the destination or source path has been seen before in the dataset, where a 0 is a previously seen path and a 1 is a new path.

ATLASv2 Fields	Unicorn Field	Description
process_path parent_path	src_id	The execution path for the current process or its parent, e.g., C:\program files\programX.exe. The path is translated into an integer in order to save memory space.
filemod_name regmod_name modload_name childproc_name crossproc_name process_path remote_ip	dst_id	System path of the entity modified by the process. The path is translated into an integer in order to save memory space.
N/A	src_type	Set to 'a' since the src always is a process.
N/A	dst_type	Set to 'a', 'c' or 'e' depending on whether the dst_id is a file, process or remote_ip entity.
action	edge_type	Action(s) associated with the operation.
N/A	new_src	Indicates if the src hasn't been seen before, 1 is a new source and 0 is an old source.
N/A	new_dst	Indicates if the dst hasn't been seen before, 1 is a new destination and 0 is an old destination.
device_timestamp	count	The device's timestamp for each operation converted into an incremental counter in the order of the timestamps. The conversion from timestamp to integer is done in order to save memory space.
src_hash	(new) src_hash	The hash value of the pre-parsed src_id. It is a six digit hexadecimal variable.

Table 5.1: Used fields in the ATLASv2 Dataset parser and their corresponding Unicorn fields.

```
Base log
src dst src_type:dst_type:edge_type:count:src_hash
23 24 a:a:pd:34:abcdef

Stream log
src dst src_type:dst_type:edge_type:new_sec:new_dst:count:src_hash
8 1462 a:c:gv:0:1:1961:abcdef
```

Figure 5.4: Unicorn rows and columns.

6

Results

The results detailed in this chapter are from tests conducted on the Unicorn Prov-HIDS with the added RCA implementation laid out in this thesis. Section 6.1 details the chosen model parameters used during testing and how they were found. Furthermore, Section 6.2 presents the performance of the model, Section 6.3 analyses the results, and Section 6.4 details the system's computational efficiency.

6.1 Hyperparameter Tuning

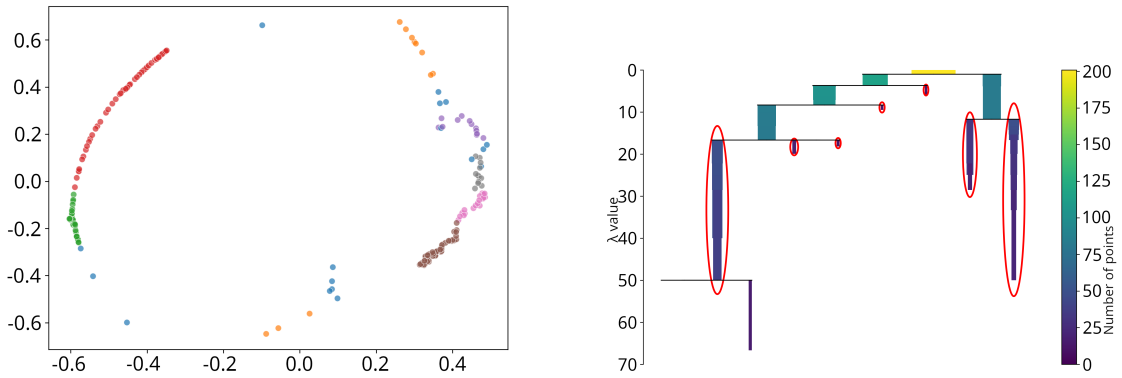
The classification model's hyperparameters and the RCA package R-value must be tuned for the system to function properly. The multi-dimensional scaling (MDS) embedding and minimum spanning tree (MST) representation of the best-performing HDBSCAN settings for each of the R-values can be found in Figures 6.1-6.5. Their corresponding noise levels and cluster counts are located in Table 6.1. Furthermore, an extended range of parameter combinations, although not all, that were tested can be found in Appendix B, while the tunings that performed worse have been left out completely.

The goal of the tuning is to find the settings that create the clearest possible clusters with minimised noise. Given that the dataset used during tuning consists solely of data from normal operations, the ideal noise level is theoretically zero. However, achieving this is improbable since there will always be outliers in the modelled data. Nonetheless, the noise should be kept as low as possible.

By analysing the MDS representations, it was concluded that the clusters for the parameters in Figure 6.1-6.4 performed similarly in terms of clustering capabilities, and most of them resulted in a few elongated clusters. The settings in Figure 6.5 produced a more compact cluster structure with clearer boundaries between its clusters. The MSTs', where cluster formations are marked in red, show that the settings represented in Figure 6.5 produce stable clusters, as most of the clusters

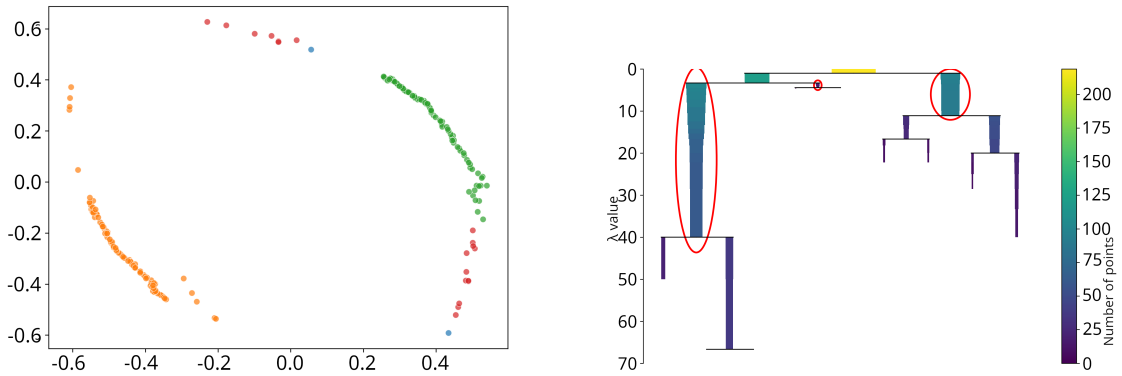
R	**1**	**3**	**5**	**10**	**20**
Noise (%)	0.0845	0.009	0.065	0.070	0.066
Clusters	7	3	9	9	15

Table 6.1: Percentage of Data Points Labelled as Noise.



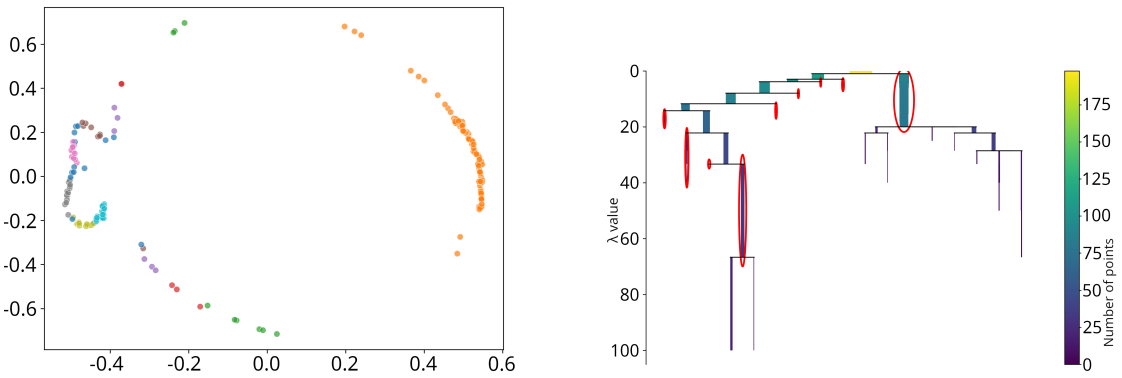
(a) MDS (b) Min-Spanning Tree

Figure 6.1: Results for R = 1, Min-Samples = 3, Min-Cluster Size = 10.



(a) MDS (b) Min-Spanning Tree

Figure 6.2: Results for R = 3, Min-Samples = 5, Min-Cluster Size = 5.

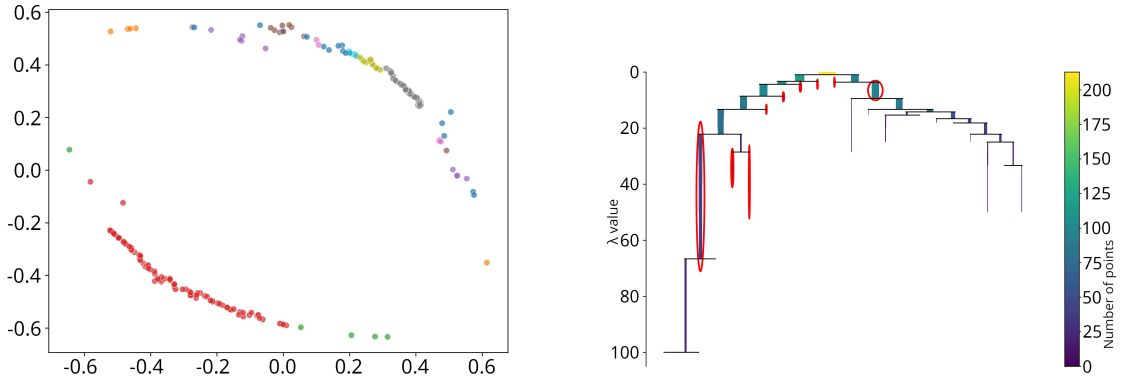


(a) MDS (b) Min-Spanning Tree

Figure 6.3: Results for R = 5, Min-Samples = 3, Min-Cluster Size = 5.

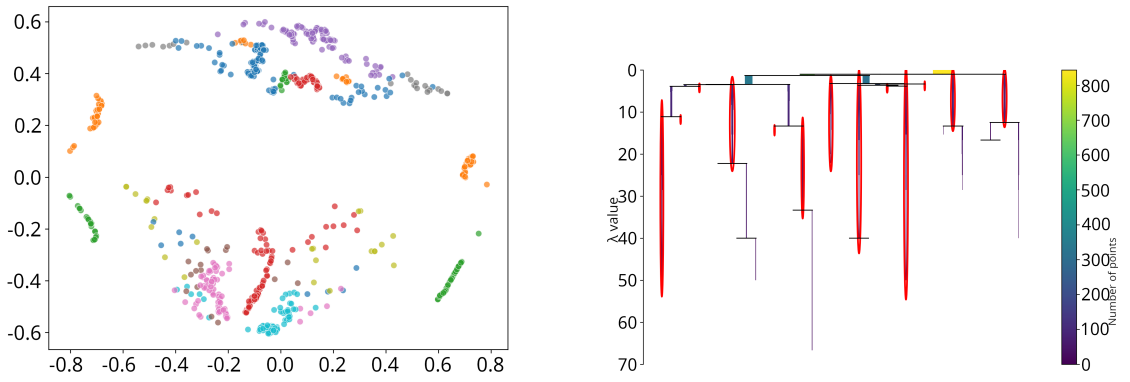
span a large range of the density thresholds value (λ), which means that the cluster stays in place as this value changes.

For further analysis, the noise levels seen in Table 6.1 can be considered. The best performing parameters for $R \in \{1, 5, 10, 20\}$ show a similar amount of noise, while $R = 3$ displays a lower noise level, as seen in Table 6.1. Furthermore, $R = 3$ has



(a) MDS (b) Min-Spanning Tree

Figure 6.4: Results for R = 10, Min-Samples = 3, Min-Cluster Size = 5.



(a) MDS (b) Min-Spanning Tree

Figure 6.5: Results for R = 20, Min-Samples = 5, Min-Cluster Size = 15.

a much lower cluster count compared to the other settings, while $R = 20$ has an increased cluster count.

The combination of good cluster shapes, reasonable cluster count and an acceptable noise level leads to the conclusion that of all the tested parameters, the values $R = 20$, Min-Samples = 5 and Min-Cluster Size = 15 perform the best. These are thus the settings that will be used for the upcoming testing of the model.

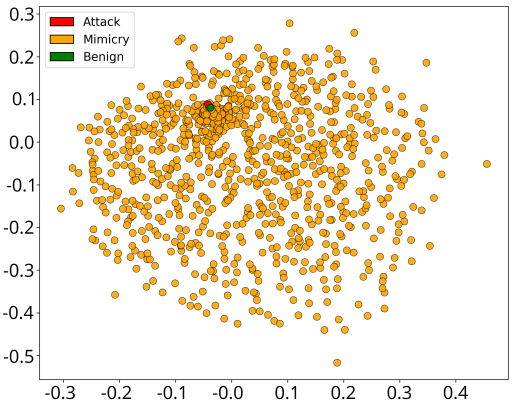
6.2 Model Performance

The RCA package is integrated into Unicorn with the parameters discovered during hyperparameter tuning. The combined model is then evaluated against all three datasets. The results show increased accuracy against mimicry attacks for the StreamSpot and ATLASv2 datasets compared to the original system, as shown in Table 6.2. However, no improvements were observed against the Theia E3 dataset.

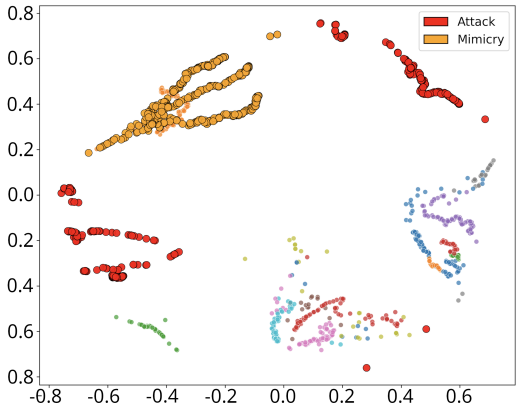
Figure 6.6 shows the projection of the test set's data points onto the clusters generated during training. All datasets display a difference in root activity between the regular attacks and the benign training data. For the mimicry attacks data, this pattern is

Dataset	Subset	Original				RCA-Enhanced			
		Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
StreamSpot	Benign	0.987	-	-	-	0.987	-	-	-
	Attack	0.995	0.995	1.0	0.995	0.995	0.995	1.0	0.995
	Mimicry	0.134	0.0	0.0	0.0	0.995	0.994	1.0	0.994
Theia	Benign	-	-	-	-	-	-	-	-
	Attack	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Mimicry	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ATLASv2	Benign	1.0	-	-	-	0.0	-	-	-
	Attack	0.888	1	0.875	0.881	0.888	0.888	1.0	0.940
	Mimicry	0.0	0.0	0.0	0.0	0.750	0.750	1.0	0.857

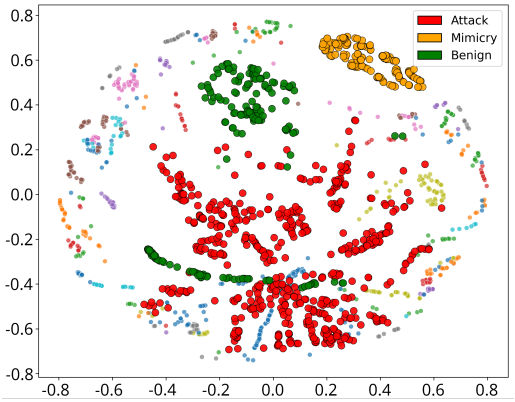
Table 6.2: Comparison of Original and RCA Measurements Across Datasets.



(a) StreamSpot



(b) Theia E3



(c) ATLASv2

Figure 6.6: Learned Clusters and Projected Test Samples.

only observed for the StreamSpot and ATLASv2 data, while the mimicry attacks against Theia E3 overlap to a large extent with an existing cluster.

The noise level in the training data can be observed in Table 6.3. The noise level for Theia E3 is the same as seen during hyperparameter tuning. For StreamSpot, the noise level is zero, which can be explained by a non-existent variance in the roots seen during training, see Table 4.1. The ATLASv2 and Theia E3 training data

Dataset	StreamSpot	Theia E3	ATLASv2
Noise (%)	0.0	0.066	0.099

Table 6.3: Noise in Training Data (%).

display similar noise levels, however, the clusters formed from the ATLASv2 data are not as cohesive as those in Theia E3. This could be explained by the more realistic benign system behaviour, as pointed out in Section 2.9.3.

6.3 Result Analysis

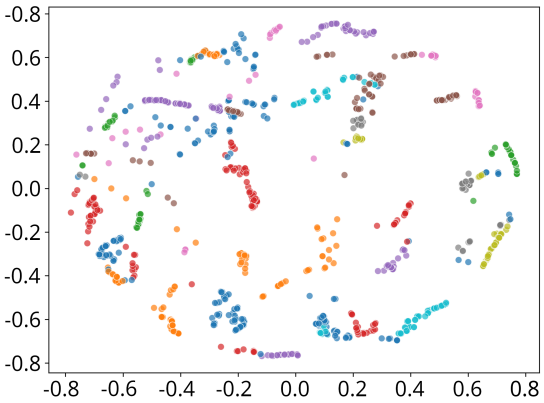
The RCA-enhanced version performs similarly in detecting normal and mimicry attacks against the StreamSpot dataset. This increase in accuracy against mimicry attacks is likely due to the low number of benign roots in the training data, as discussed in Section 5, which trivialises the task of finding outlier roots.

In contrast, the RCA-enhanced classifier does not achieve any improvement when evaluated on the Theia E3 dataset. This is because the root sketches related to mimicry attacks are placed close to an existing cluster and thus classified as benign. The cause of this is likely due to how the edge direction for edges connecting to file nodes is handled in the dataset, as will be discussed further in Section 7.1.

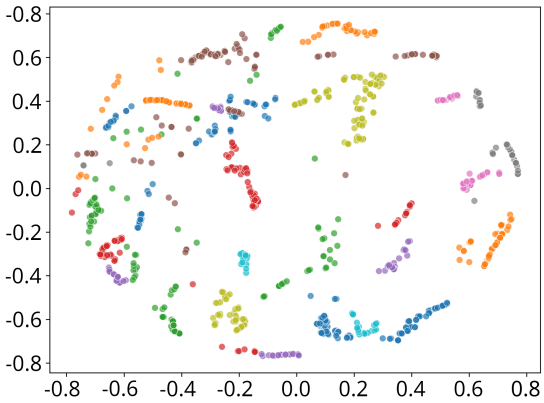
The tests against the ATLASv2 data show an increase in accuracy for mimicry attacks. However, it displays a complete loss of accuracy against the benign test set. Simultaneously, the precision has been reduced while the recall has increased. The model has thus changed from an excess of false negatives to an excess of false positives when compared to the original Unicorn implementation.

This reduction of accuracy for the benign test set has two likely causes. The first cause is that the clustering for the training data does not create clearly defined clusters. This leads to incoherent clusters and reduced accuracy. The clustering becomes slightly better if the hyperparameters are tuned specifically for the ATLASv2 training data. This can be observed in Figure 6.7 that displays the clustering for ATLASv2, where the hyperparameter tuning has been redone for that specific dataset. However, this tuning does not increase the accuracy when tested on the benign test set. This hints at cluster coherence not being a problem since the increased coherence at optimal parameters does not affect accuracy.

The second possible cause, which we deem more likely, is that there is too much variation in the underlying system behaviour and its connected roots. This can be observed in Figure 6.6c, where the data points from the benign test set (green) differ significantly from those observed during training, and are not placed as part of an already existing cluster. The placement of the benign test data results in them being marked as malicious when evaluated by the clustering model. This issue might be limited with an extended training period, as it is currently trained on the provenance data from four days of consecutive usage. An extended training period would likely yield better clusters. However, this would require a completely new dataset to be



Original Parameters, Min-Sample = 5, Min-Cluster Size = 15



Optimal Parameters, Min-Sample = 3, Min-Cluster Size = 20

Figure 6.7: Cluster Comparison for Original and Optimal ATLASv2 Configurations, R = 20

tested properly.

A surprising discovery in these results is that the normal attacks deviate substantially from the benign behaviour. Even in a compromised state, most of the ongoing activity should correspond to benign operations, with the attack only consuming a small portion of the system's resources. Thus, it is unintuitive that these regular attacks and the ongoing background activity still deviate from the expected behaviour. A possible cause for this anomaly could be that the benign behaviour has not been continued during the attack sequence to the same extent as would be expected during normal use. This could skew the root activity and put more emphasis on the abnormal root action. However, this requires further analysis of the dataset to conclude if this is the true cause.

6.4 Execution Time and Space Efficiency

The results shown in Figure 6.8 are the average measurements from all test runs for both the original Unicorn and RCA-enhanced implementations. Both implementations had the default parameters, as seen in Table 4.2.

Figure 6.8 shows that the RCA implementation on average completed at 190 seconds with a peak of 516MB used in the memory. Unicorn, however, finished on average after 181 seconds with a maximum peak of 505MB used in memory. These results show that the RCA implementation has a 4.97% increased execution time and uses 2.17% more memory than Unicorn.

The analysed data is the unmodified attack data from host machine one in the ATLASv2 dataset. It is logs from 10 attacks during a 5.5 hours long real-time log period. The profiler that ran the analysis utilised 100% of the CPU. There was no external input on the machine running the profiled analysis and no other non-idle activity competed for CPU power. In the Unicorn paper, X. Han *et al.* claim

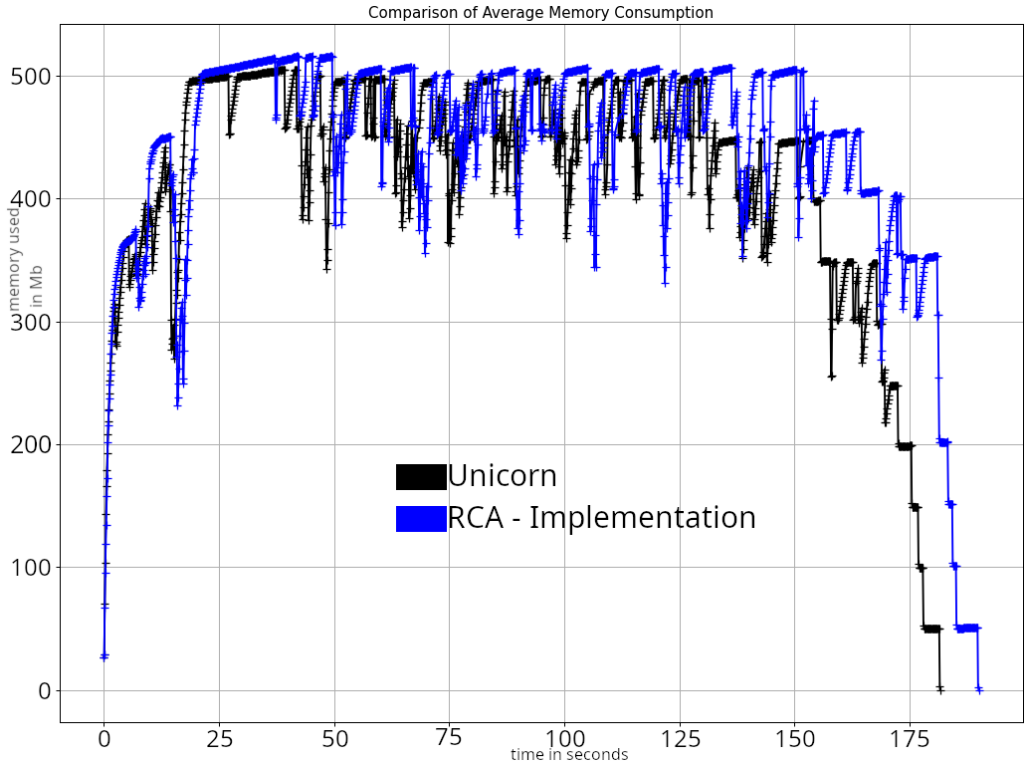


Figure 6.8: Comparison of the average space and time consumption. Unicorn is in black, and the RCA implementation is in blue.

that Unicorn on average utilises 12.3% of the CPU during a real-world long-running experiment [1]. This implies that the RCA implementation uses no more than 5.0% of the Unicorn CPU utilisation and thus is within acceptable boundaries of utilisation.

6.5 Assessing Outcomes of Thesis Goals

The main goals of this thesis, as proposed in Section 1.3, have been achieved to some extent. Below follows an elaboration of how well the goals have been achieved, remaining problems and possible improvements.

6.5.1 Assess RCA's Effectiveness Against Mimicry Attacks in Coarse-Grained Prov-HIDS

The clear separation of the data sketches seen during training and those found in the attack sets indicates that RCA could be an effective way of finding ongoing intrusions for both regular and mimicry attacks.

Our opinion is that future research would require new datasets. The current publicly available datasets are currently not up to the standards needed to asses this properly. The main concern is the low level of overlap between the benign test and training root activity. A proper assessment would require a dataset with three major improvements: increased size, original mimicry attack data and strictly controlled benign behaviour.

These dataset traits will be discussed further in Section 7.2.4. A complete assessment of RCA would only be possible if such a dataset were available. Until then, we would not go further than assessing its effectiveness as possible, but not yet achieved.

6.5.2 Raise the Detection Probability of Mimicry Attacks in Prov-HIDS

The results shown in Table 6.2 show that the RCA enhancement increases the detection of mimicry attacks. However, it also shows a potential decrease in the detection of benign behaviour; marking them as false positives. This flaw was not remedied by further hyperparameter tuning. This could indicate that another classification model or granularity could be better suited to achieve a sustained accuracy across benign and malicious inputs.

6.5.3 Propose a generalised integration of RCA Into Coarse-Grained Prov-HIDS Systems

The RCA package created during this thesis showed potential for being viable for root operations when integrated into other systems. The proposed package solution is fully integrable into a coarse-grained Prov-HIDS, even if the extension to Unicorn was less successful than anticipated. There would only be a slight overhead for such an improvement, with a possible increase in the detection of mimicry attacks.

7

Discussion

This chapter presents practical observations and technical limitations encountered during the development and experimentation phases of the project. It highlights issues related to dataset suitability, system design constraints, and challenges with integrating or extending existing tools. These insights aim to raise potential concerns about the system and its supporting resources, while also guiding future work and improvements.

7.1 Provenance Graph Edge Directions

This project discovered a problem that has not received much attention in earlier Prov-HIDS projects. This is the importance of edge direction, more specifically, the directions of edges for file operations. This difference is displayed in the changed classification behaviour for Theia E3 and ATLASv2 dataset, where only the mimicry attacks from the former dataset avoids detection.

`READ` instructions in the provenance graph for the Theia E3 dataset are directed from the file node to the process node, as seen in Figure 7.1. This enables files to act as roots to process nodes that read the file's content without modifying it first. The result of this is that a mimicry attack that mimics the behaviour of applications with many read operations automatically creates large portions of normal root behaviour when it mimics the expected `READ` instructions. Thus, it can also mimic portions of the normal root activity.

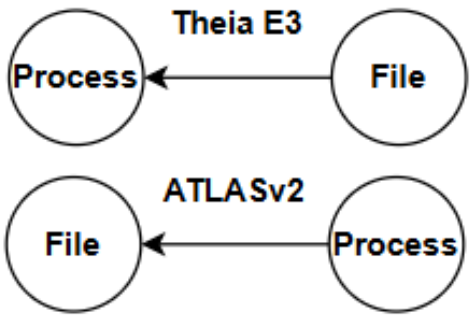


Figure 7.1: Edge Directions for Theia E3 and ATLASv2

In contrast, the ATLASv2 dataset models all file interactions with edges directed to the file. This results in file nodes being exempted as possible roots. When a mimicry attack mimics the file operations in such models, its file activity will not add additional benign roots. Instead, its already connected (malicious) roots will show increased activity and make its root behaviour diverge further from what is expected.

To summarise, the provenance graph edges must be directed towards the file node for all operations to avoid the problem where file interactions can mimic benign roots, Alternatively, file nodes should be exempted as possible roots altogether. This is something that must be taken into account when designing the root handler system. With the generic RCA-package approach proposed in this thesis, this could be achieved by allowing support for node types and introducing a system configuration to control what types are allowed to be assigned as roots.

7.2 Dataset Relevance

Three datasets were used in this thesis: StreamSpot, Theia Engagement 3, and ATLASv2, all have an irregular level of quality, and there are many desired improvements for future work. However, the upside is that they have already been considered relevant for the research topic, and that there is support for the StreamSpot and Theia E3 data in the current Unciorn system. ATLASv2, however, is considerably newer, it had no native provenance graph representation available, and no direct integration with Unicorn. This integration was created in this thesis, as seen in Section 5.5.

7.2.1 StreamSpot Dataset Relevance

StreamSpot is the oldest dataset used and its data is completely task-specific. It is, however, an established dataset and has publicly available mimicry attacks. Taking this information into account, we believe that StreamSpot is the least relevant dataset to this project. The dataset contains outdated and low-quality data. Its relevance lies in its widespread use and the availability of predefined mimicry attacks. Old data in itself does not make a dataset less desirable to utilise. However, old data combined with too specific data will likely cause the trained model to miss several more modern attacks in a complete system with a lot of noise.

7.2.2 Theia E3 Dataset Relevance

Theia E3 is a relevant dataset. The advantages are that the dataset aims to provide logs for the whole system, the publisher (DARPA) is trustworthy, and the attacks are APTs. However, the disadvantages are that there was no thorough check whether the data is accurate, its increasing age, and there is no publicly available mimicry attack for the dataset. The authors acknowledge a lack of accuracy by citing issues during the logging phase and explicitly stating that they make no guarantees about the correctness of the data [10], [13], [55].

7.2.3 ATLASv2 Dataset Relevance

ATLASv2 is the newest dataset out of the three. It contains a larger set of benign data and noise compared to the other datasets. However, the attacks are very similar to each other and there are no available mimicry attacks. Nevertheless, it remains relevant to this thesis and similar projects due to its increased size and wider range of included behaviour. The dataset is well documented, making it straightforward to identify attacks and adapt them into mimicry attacks with minimal risk of introducing errors. Furthermore, the log data should be converted correctly into a provenance graph, according to all available specifications and documentation. It is, however, impossible to verify that no errors have occurred without extensive research into this specific topic which was out of the scope for this thesis.

7.2.4 Desired dataset

Our opinion is that the publicly available datasets are currently not up to the standards needed to asses this type of system properly. A proper assessment would require a dataset with four major improvements.

Firstly, the size of the dataset would need to increase. Currently, the ATLASv2 dataset provides the longest period of benign operations. However, it is still restricted to four days of regular usage. The benign period would need to be extended substantially to allow for training and evaluation of a proper coarse-grained model, since it is dependent on the complete system behaviour at all stages of execution.

Secondly, there is a need for more provenance data for mimicry attacks. To our knowledge, this is only available for the StreamSpot dataset at the current time. Currently, each research project has to transform regular attacks into mimicry attacks when testing with other datasets. The creation of a new dataset specifically focused on mimicry attacks would limit this problem, which could simplify testing and make results across research projects simpler to assess.

The dataset also needs a variety of different types of attacks. ATLASv2 has 10 attacks of similar nature, StreamSpot contains one attack, and Theia E3 does not specify how diverse their attacks are. This lack of diversity could cause the system to show bias for one type of attack, and thus, the system might not be as effective as believed against other types of attacks.

Lastly, the benign data generation needs to continue with similar usage when the attack is performed, as is done during the benign data collection period. This would ensure that the difference in perceived root behaviour found by the model is due to attack roots and not a change in simultaneous benign executions.

7.3 Histogram Size Problem

When analysing the components of the RCA package, a potential problem was noticed in the HistoSketch system. The system has no built-in function to handle outdated labels in the histogram. This means that when label counts decay towards

zero, they will always remain present in the histogram. Even when the label reaches a state of extremely low possibility of being present in the actual sketch, it is still present in the histogram.

In a worst-case scenario for a continuously running system, the number of observed labels (root IDs) can grow indefinitely. The histogram could then contain a few active labels that significantly contribute to the resulting sketch while accumulating an infinite number of stale labels with count values near zero. This could cause the histogram to explode in size. As a result, it could substantially increase the computational time since each label's count is adjusted at set intervals. Increases in the number of histogram labels would also require more memory to store all these stale labels. To remedy this, the HistoSketch system would need a function that removes labels that have a low probability of being present in the final sketch. Preferably with a probability threshold that can be chosen for each system, or as a function of sketch size and tolerable memory usage.

7.4 Proposal of Model Improvements

Most of the Prov-HIDSs that have been studied in this thesis only analyse the complete system state. As a result, the classification system does not make a distinction between the different applications running on the system. If a host is running two applications, A and B, the classifier may not interpret unusual behaviour of Application A as abnormal if it is similar to the usual behaviour of Application B. This can then be exploited by an attacker who knows the complete system state by extending its available system calls to mimic the state of the complete system.

A better approach could be to build smaller application-specific models that combine root cause and subsequent actions directly instead of analysing them separately. This avoids the problem of looking at root activity as a single metric across the host to instead focus on each specific application. Furthermore, the range of available system calls for the mimicry exploit is narrowed to those of the exploited system. This would make it harder to build an accurate mimicry chain and simultaneously achieve the goal of the attack.

This would be a shift away from the coarse-grained classification approach that has been tested in this thesis, while simultaneously not moving fully into the fine-grained classification domain. Such an approach could preserve some of the computational speed of coarse-grained models while achieving a more in-depth analysis of the system's actions.

7.5 Future Works

The thesis proposes a few research areas that future researchers can focus on. First, there is a need for improved public datasets to fully establish the impact of mimicry attacks and enable complete evaluation of effective countermeasures. The properties needed for such a dataset are clearly defined in Section 7.2. A new dataset would

not only enable further evaluation of the effectiveness of root cause analysis, it would also allow for research into other possible countermeasures. In our opinion, this is a necessity to push this field of research further with high confidence in future results.

Simultaneously, there is a need for stricter provenance graph guidelines to avoid problems such as the file edge direction, as discussed in 7.1. Guidelines such as a standardisation of vertex and edge labelling, and edge direction of all available system calls. This could greatly improve the comparability of systems that utilise provenance graphs. This could be implemented at either the logging system level or within the graph parser.

Furthermore, continuously deployed detection systems must adhere to strict limits of resource usage. Future research into system optimisations is thus always of relevance. For the RCA system, possible improvements could be found in the sketching system, as laid out in Section 7.3.

8

Conclusion

This thesis is about leveraging root cause analysis (RCA) in coarse-grained Provenance graph-based host intrusion detection systems (Prov-HIDS). The project extended on a Prov-HIDS called Unicorn by incorporating a newly developed root handling package for storage and propagation of root activity. The new system utilises the HistoSketch system to create comparable snapshots over the host's root activity [27]. Leveraging these new sketches, the system could provide more context for finding attacks and recognising behavioural changes.

The thesis also focused on finding mimicry attacks, a type of evasion attack that has been particularly hard to identify. The new system showed potential in finding mimicry attacks. It was, however, at the cost of misclassifying benign behaviour. The RCA enhancement required only a slight increase in overhead; 4.97% increased execution time and 2.17% memory usage. The potential of improved detection rate with only a slight increase in overhead could give it an edge over other coarse-grained Prov-HIDS's. Although the problem with misclassification of benign behaviour needs to be solved before such a system is viable.

A stand-alone RCA package was implemented. It has a focus on flexibility to not limit the package to a single system. The package had three main functionalities: finding the roots, propagating them throughout the provenance graph, and storing the root representation in sketches. It handles several roots per node, which allows the system to trace backwards several generations in the provenance graph.

The RCA package considers all types of nodes as possible roots compared to other RCA approaches, thus allowing the system to catch a wider range of APTs. However, the results show that file nodes should not be considered as roots, as discussed in 7.1. In the developed RCA-package, a node is determined to be a root if it only has outgoing edges or if it has at least one outgoing edge before its first incoming edge. The HDBSCAN model was used to classify the root sketches. HDBSCAN is an extension of DBSCAN, but the former does not rely on a fixed number of clusters. This allows the system to handle a variety of clusters without manual changes.

Three unique datasets were used to test the system: StreamSpot, DARPA Transparent Computing Theia Engagement 3, and ATLASv2. There were, however, concerns about the quality of the datasets and further analysis would be necessary. Nevertheless, the results presented in this thesis highlight that root cause analysis can possibly enhance the effectiveness of coarse-grained Prov-HIDS in detecting mimicry

attacks, while remaining resource-efficient. However, further research is needed to cement this hypothesis and verify that previously unseen benign behaviour can be classified correctly.

Bibliography

[1] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Proceedings 2020 Network and Distributed System Security Symposium*, ser. NDSS 2020, Internet Society, 2020. DOI: 10.14722/ndss.2020.24046. [Online]. Available: http://dx.doi.org/10.14722/ndss.2020.24046.

[2] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1035–1044, ISBN: 9781450342322. DOI: 10.1145/2939672.2939783. [Online]. Available: https://doi.org/10.1145/2939672.2939783.

[3] A. Goyal, G. Wang, and A. Bates, "R-caid: Embedding root cause analysis within provenance-based intrusion detection," in *2024 IEEE Symposium on Security and Privacy (SP)*, IEEE Computer Society, 2024, pp. 257–257.

[4] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02, Washington, DC, USA: Association for Computing Machinery, 2002, pp. 255–264, ISBN: 1581136129. DOI: 10.1145/586110.586145. [Online]. Available: https://doi.org/10.1145/586110.586145.

[5] A. Goyal, X. Han, G. Wang, and A. Bates, "Sometimes, you aren't what you do: Mimicry attacks against provenance graph host intrusion detection systems," *30th Network and Distributed System Security Symposium*, 2023. [Online]. Available: https://par.nsf.gov/biblio/10412012.

[6] Q. Wang, W. U. Hassan, D. Li, *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis.," in *NDSS*, 2020.

[7] X. Han, *StreamSpot Dataset*, version V1, 2018. DOI: 10.7910/DVN/83KYJY. [Online]. Available: https://doi.org/10.7910/DVN/83KYJY.

[8] E. Manzoor, *Sbustremspot/sbusstreamspot-data*, https://github.com/sbustreamspot/sbustreamspot-data, 2016.

[9] F. DARPA, *Engagement 3*, https://drive.google.com/drive/folders/1QlbUFWAGq3Hpl8wVdzOdIoZLFxkII4EK, Accessed: 2025-02-10, 2018.

[10] a. Jacob Torret, *Transparent*, https://github.com/darpa-i2o/Transparent-Computing, 2020.

[11] The Defense Advanced Research Project Agency, *Tc: Transparent computing*, Accessed: 2025-02-10, Date: Unknown.

[12] K. W. Andy Riddle and A. Bates, *Atlasv2: Atlas attack engagements, version 2*, 2023. arXiv: 2401.01341 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2401.01341.

[13] J. Liu, A. Inam, A. Goyal, K. Westfall, A. Riddle, and A. Bates, *Reapr: Recovery every attack process*, https://bitbucket.org/sts-lab/reapr-ground-truth, 2025.

[14] A. Riddle, K. Westfall, and A. Bates, *Sts-lab/atlasv2*, https://bitbucket.org/sts-lab/atlasv2/src/master/, 2024.

[15] T. Pasquier, X. Han, M. Goldstein, *et al.*, "Practical whole-system provenance capture," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 405–418.

[16] A. Syalim, T. Nishide, and K. Sakurai, "Preserving integrity and confidentiality of a directed acyclic graph model of provenance," in *Data and Applications Security and Privacy XXIV: 24th Annual IFIP WG 11.3 Working Conference, Rome, Italy, June 21-23, 2010. Proceedings 24*, Springer, 2010, pp. 311–318.

[17] U. J. Braun, A. Shinnar, and M. I. Seltzer, "Securing provenance," in *Proceedings of the 3rd USENIX Workshop on Hot Topics in Security (HotSec'08)*, Usenix Association, 2008.

[18] S. Miles, P. Groth, S. Munroe, S. Jiang, T. Assandri, and L. Moreau, "Extracting causal graphs from an open provenance data model," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 577–586, 2008.

[19] P. Chen, L. Desmet, and C. Huygens, "A study on advanced persistent threats," in *Communications and Multimedia Security*, B. De Decker and A. Zúquete, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 63–72, ISBN: 978-3-662-44885-4.

[20] N. I. Che Mat, N. Jamil, Y. Yusoff, and M. L. Mat Kiah, "A systematic literature review on advanced persistent threat behaviors and its detection strategy," *Journal of Cybersecurity*, vol. 10, no. 1, tyad023, Jan. 2024, ISSN: 2057-2085. DOI: 10.1093/cybsec/tyad023. eprint: https://academic.oup.com/cybersecurity/article-pdf/10/1/tyad023/61182350/tyad023.pdf. [Online]. Available: https://doi.org/10.1093/cybsec/tyad023.

[21] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1851–1877, 2019. DOI: 10.1109/COMST.2019.2891891.

[22] E. D. Reilly, "No-op," in *Encyclopedia of Computer Science*. GBR: John Wiley and Sons Ltd., 2003, p. 1241, ISBN: 0470864125.

[23] P. Akritidis, E. Markatos, M. Polychronakis, and K. Anagnostakis, "Stride: Polymorphic sled detection through instruction sequence analysis.," May 2005, pp. 375–392, ISBN: 978-0-387-25658-0. DOI: 10.1007/0-387-25660-1_25.

[24] H. Satilmiş, S. Akleylek, and Z. Y. Tok, "A systematic literature review on host-based intrusion detection systems," *IEEE Access*, vol. 12, pp. 27 237–27 266, 2024. DOI: 10.1109/ACCESS.2024.3367004.

[25] B. Andersen and T. Fagerhaug, *Root cause analysis*. Quality Press, 2006.

[26] G. Cormode, "Data sketching," *Commun. ACM*, vol. 60, no. 9, pp. 48–55, Aug. 2017, ISSN: 0001-0782. DOI: 10.1145/3080008. [Online]. Available: https://doi.org/10.1145/3080008.

[27] D. Yang, B. Li, L. Rettig, and P. Cudré-Mauroux, "Histosketch: Fast similarity-preserving sketching of streaming histograms with concept drift," in *2017 IEEE International Conference on Data Mining (ICDM)*, 2017, pp. 545–554. DOI: 10.1109/ICDM.2017.64.

[28] L. Rhodes, "Data sketches: Fast, approximate analysis of big data," Yahoo, Inc., White Paper, 2015, Originally published December 17, 2015. [Online]. Available: https://datasketches.apache.org/docs/pdf/DataSketches.pdf.

[29] M. Manasse, F. McSherry, and K. Talwar, "Consistent weighted sampling," *Unpublished technical report) http://research. microsoft. com/en-us/people/manasse*, vol. 2, 2010.

[30] P. Li, "0-bit consistent weighted sampling," in *Proceedings of the 21th ACM SIGKDD International conference on knowledge discovery and data mining*, 2015, pp. 665–674.

[31] L. McInnes, J. Healy, S. Astels, *et al.*, "Hdbscan: Hierarchical density based clustering.,"

[32] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, 1996, pp. 226–231.

[33] D. Brain and G. I. Webb, "On the effect of data set size on bias and variance in classification learning," in *Proceedings of the Fourth Australian Knowledge Acquisition Workshop, University of New South Wales*, 1999, pp. 117–128.

[34] A. Kyrola, G. Blelloch, and C. Guestrin, "GraphChi: Large-Scale graph computation on just a PC," in *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, Hollywood, CA: USENIX Association, Oct. 2012, pp. 31–46, ISBN: 978-1-931971-96-6. [Online]. Available: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/kyrola.

[35] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, *UNICORN*, en. [Online]. Available: https://github.com/crimson-unicorn (visited on 02/05/2025).

[36] V. Kotu and B. Deshpande, "Chapter 4 - classification," in *Data Science (Second Edition)*, V. Kotu and B. Deshpande, Eds., Second Edition, Morgan Kaufmann, 2019, pp. 65–163, ISBN: 978-0-12-814761-0. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00004-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128147610000046.

[37] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics*, vol. 9, no. 8, p. 1295, 2020.

[38] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[39] A. Bookstein, V. A. Kulyukin, and T. Raita, "Generalized hamming distance," *Information Retrieval*, vol. 5, pp. 353–375, 2002.

[40] A. Alsaheel, Y. Nan, S. Ma, *et al.*, "{Atlas}: A sequence-based learning approach for attack investigation," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3005–3022.

[41] A. Alsaheel, Y. Nan, S. Ma, *et al.*, *Pureseclab/atlas*, `https://github.com/purseclab/ATLAS`, 2022.

[42] MITRE Corporation, *Common vulnerabilities and exposures*, Accessed: 2025-04-10, 2025. [Online]. Available: `https://www.cve.org/`.

[43] Forum of Incident Response and Security Teams, *Common vulnerability scoring system (cvss)*, `https://www.first.org/cvss/`, Accessed: 2025-05-09, 2025.

[44] T. H. Ptacek and T. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," 1998. [Online]. Available: `https://api.semanticscholar.org/CorpusID:16418229`.

[45] D. A. Wagner and D. Dean, "Intrusion detection via static analysis," *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, pp. 156–168, 2001. [Online]. Available: `https://api.semanticscholar.org/CorpusID:6331338`.

[46] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1283–1296, 2018.

[47] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff, "A sense of self for unix processes," in *Proceedings 1996 IEEE Symposium on Security and Privacy*, 1996, pp. 120–128. DOI: `10.1109/SECPRI.1996.502675`.

[48] [Online]. Available: `https://bitbucket.org/sts-lab/mimicry-provenance-generator/src/master/`.

[49] E. Henriksson and S. Kvaldén, *Github:donkey-parsers*, original-date: 2025-02-11T09:28:57Z, Feb. 2025. [Online]. Available: `https://github.com/HenrikssonErik/donkey-parsers` (visited on 02/27/2025).

[50] R. Brandis and L. Steller, "Threat modelling adobe pdf," *DSTO Defence Science and Technology Origanisation*, 2012.

[51] V. Pappu and P. M. Pardalos, "High-dimensional data classification," *Clusters, Orders, and Trees: Methods and Applications: In Honor of Boris Mirkin's 70th Birthday*, pp. 119–150, 2014.

[52] F. Pedregosa, *Memory-profiler: A module for monitoring memory usage of a python program*, `https://pypi.org/project/memory-profiler/`, Version 0.61.0, released on November 15, 2022, 2022. [Online]. Available: `https://pypi.org/project/memory-profiler/`.

[53] E. Henriksson and S. Kvaldén, *Github:donkey-analyzer*, original-date: 2025-02-11T09:25:12Z, Feb. 2025. [Online]. Available: `https://github.com/HenrikssonErik/donkey-analyzer` (visited on 02/27/2025).

[54] E. Henriksson and S. Kvaldén, *Github:donkey-modeler*, original-date: 2025-02-11T09:27:58Z, Feb. 2025. [Online]. Available: `https://github.com/HenrikssonErik/donkey-modeler` (visited on 02/27/2025).

[55] F. DARPA, *Engagement 3 operational event log*, `https://drive.google.com/file/d/1mnx73nb0KMX4EbgSiBLu0tkKrQ34P96H/view`, Accessed: 2025-06-10, 2018.

A

Appendix

A.1 Ethics

This project does not pose any obvious ethical concerns. Its goal is to strengthen existing HIDS systems against mimicry attacks, thereby reducing the risk of adversaries compromising systems. By improving security, the project contributes positively to cybersecurity.

The only minor concern is that, as an open-source project, adversaries may analyse the solution to identify potential exploits. However, this transparency also allows researchers and security experts to detect and address the same vulnerabilities, ultimately leading to a more robust and secure system. Moreover, security through obscurity is widely regarded as an ineffective security measure. All system developments during this project will be publicly available for further analysis and reproducibility.

```
Input: unsigned long edge_ts, Root node_roots[], Root out_edge_roots[]
Output: bool // notify if edge root list is changed or not

if(node_roots[0].root == 0){
    // A zero as first root indicates an empty root list
        return
    }

Set validRoots; // Track unique root values
Root oldArray[] = copy(out_edge_roots) // Tracks changes

for i = 0 to i < node_roots.length do
        Root root = node_roots[i];

        if (root.root == 0) {
            break; //end of list reach if we see a 0 root
        } else if (! validRoots.contains(root) {
            if (root.ts <= edge_ts){
                validRoots.add(root);
            }
        }
od

for i = 0 to i < out_edge_roots.length do
        Root root = out_edge_roots[i];

        if (root.root == 0) {
            break; //end of list reach if we see a 0 root
        } else if (! validRoots.contains(root) {
            //only roots with ts before the edge was created are seen
            ↪  as valid
            if (root.ts <= edge_ts){
                validRoots.add(root);
            }
        }
od

//Sort depending on creation timestamp, ensures the most recent roots
↪  are kept
sort(validRoots)

for i = 0 to i < out_edge_roots.length do
        Root root = vaidRoots[i];
        out_edge_roots[i] = root //update edge roots
od

return oldArray != out_edges_roots
```

Figure A.1: Root Propagation To Outgoing Edges

```
Input: Root node_roots[], Root in_edge_roots[]
Output: bool // notify if edge root list is changed or not

if(in_edge_roots[0].root == 0){
    // A zero as first root indicates an empty root list
        return
    }

Set validRoots; // Track unique root values
Root oldArray[] = copy(node_roots) // Tracks changes

for i = 0 to i < in_edge_roots.length do
        Root root = in_edge_roots[i];

        if (root.root == 0) {
            break; //end of list reach if we see a 0 root
        } else if (! validRoots.contains(root) {
            //all incomging roots are considered valid
            validRoots.add(root);
        }
od

for i = 0 to i < node_roots.length do
        Root root = node_roots[i];

        if (root.root == 0) {
            break; //end of list reach if we see a 0 root
        } else if (! validRoots.contains(root) {
                validRoots.add(root);
        }
od

//Sort depending on creation timestamp, ensures the most recent roots
↪  are kept
sort(validRoots)

for i = 0 to i < node_roots.length do
        Root root = vaidRoots[i];
        node_roots[i] = root //update edge roots
od

return oldArray != node_roots
```

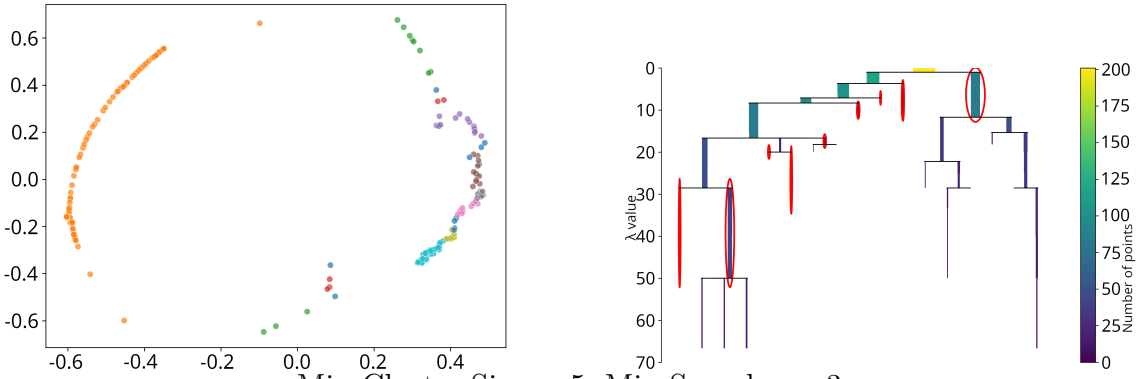
Figure A.2: Root Assignment From Incoming Edges

```
Input: Vertex *v
Output: Void

NodeInfo info = v.nodeInfo

if not (info.checkedIfRoot){

    unsigned long in_edges_ts[]
    unsigned long out_edges_ts[]
    unsigned long minIn = MAX_VALUE
    unsigned long minOut = MAX_VALUE

    //Find smallest incoming ts and populate ts array
    for (i = 0 to i < v.num_inedges) do
        e_ts = vertex.inedge(i).ts
        in_edges_ts[i] = e_ts
        if(minIn > e_ts){
            minIn = e_ts
        }
    od

    //Find smallest outgoing ts and populate ts aray
    for (i = 0 to i < vertex.num_outedges) do
        e_ts = vertex.outedge(i).ts
        out_edges_ts[i] = e_ts
        if(minOut > e_ts){
            minOut = e_ts
        }
    od

    info.node_ts = MIN(minIn, minOut)
    info.vertex_id = v.id
    checkAndAssignRoots(info, v.roots, in_edges_ts, v.num_inedges,
    ↪  out_edges_ts, v.num_outedges)
    v.nodeInfo = info
}

for (i = 0 to i < v.num_inedges) do
    e = vertex.inedge(i)
    updateRootsFromInEdges(info, e.roots, v.roots)
od

if (v.roots[0].root != 0){
    for (i = 0 to i < vertex.num_outedges) do
        e = vertex.outedge(i)
        updateOutedgeFromNode(e.ts, e.roots, v.roots);
    od
}
```

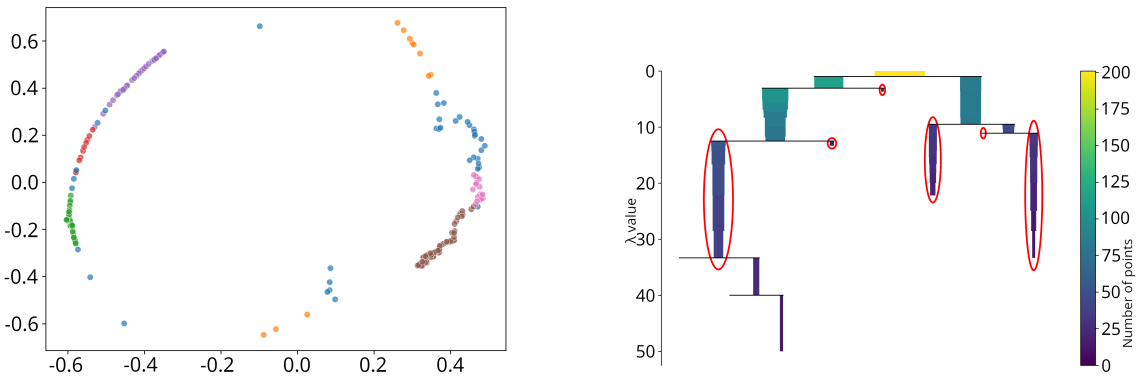
Figure A.3: Helper Function For RCA In Unicorn

B

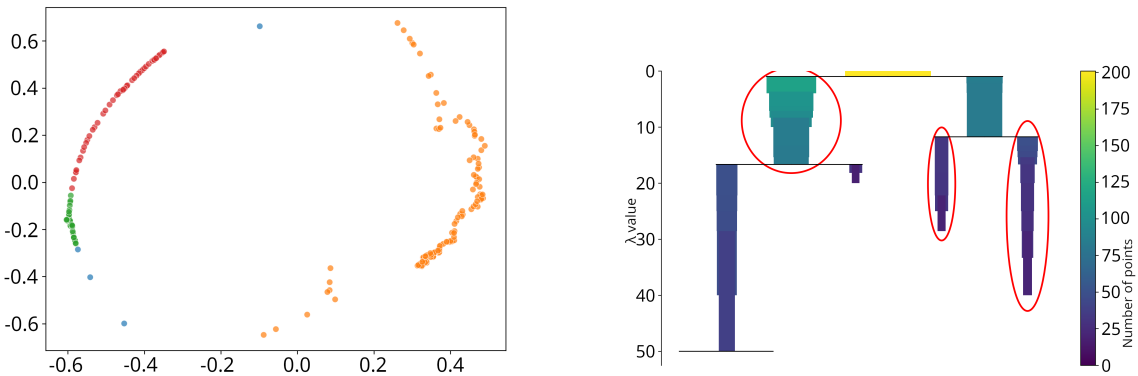
Appendix



Min-Cluster Size = 5, Min-Samples = 3

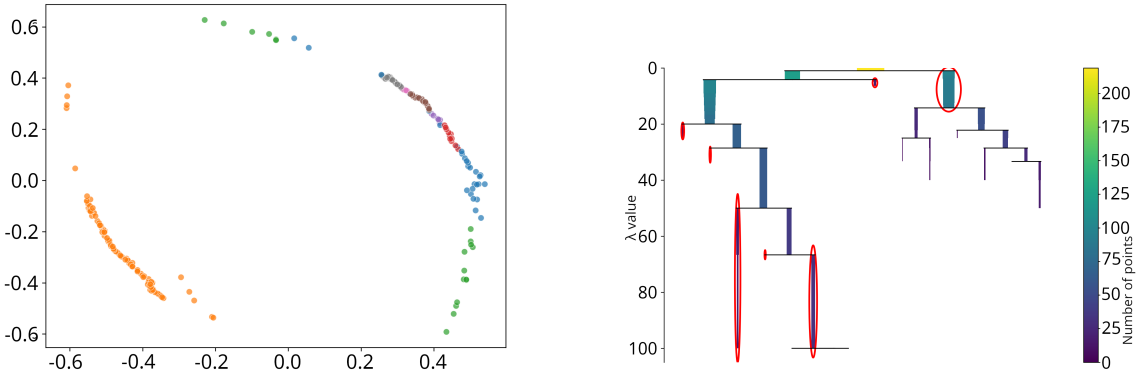


Min-Cluster Size = 10, Min-Samples = 5

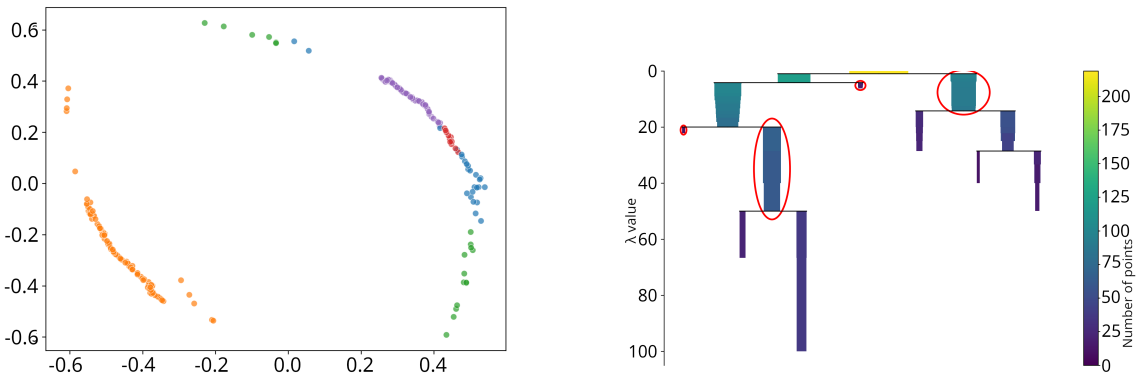


Min-Cluster Size = 15, Min-Samples = 3

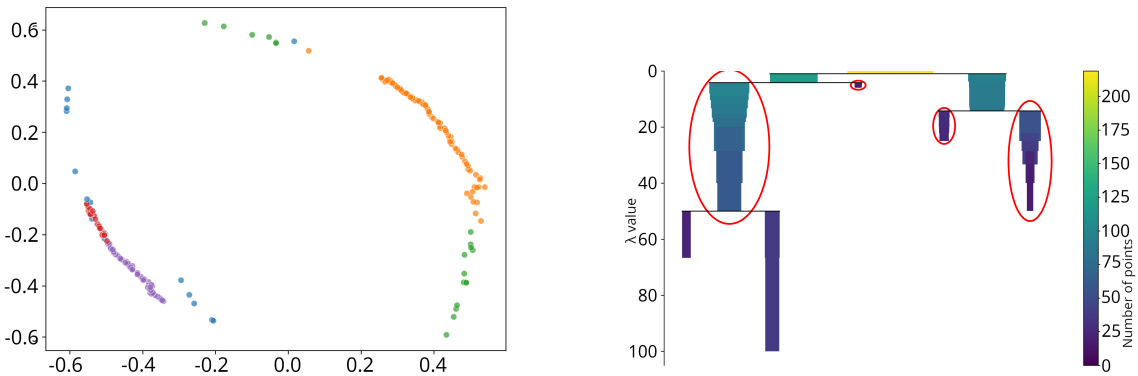
Figure B.1: Hyperparameter Tests, R = 1



Min-Cluster Size = 5, Min-Samples = 3

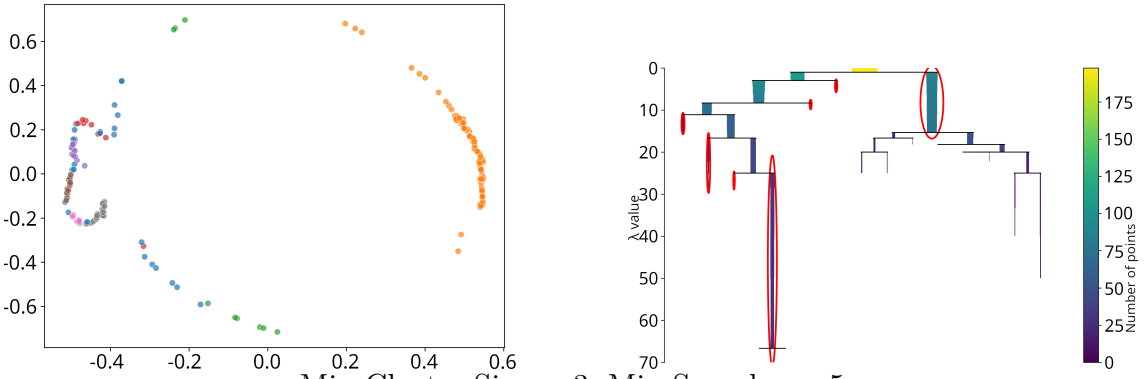


Min-Cluster Size = 10, Min-Samples = 3

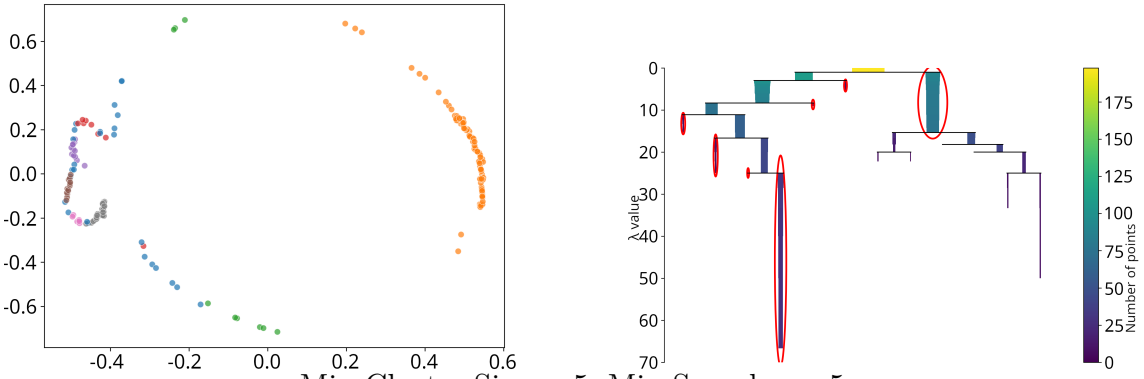


Min-Cluster Size = 15, Min-Samples = 3

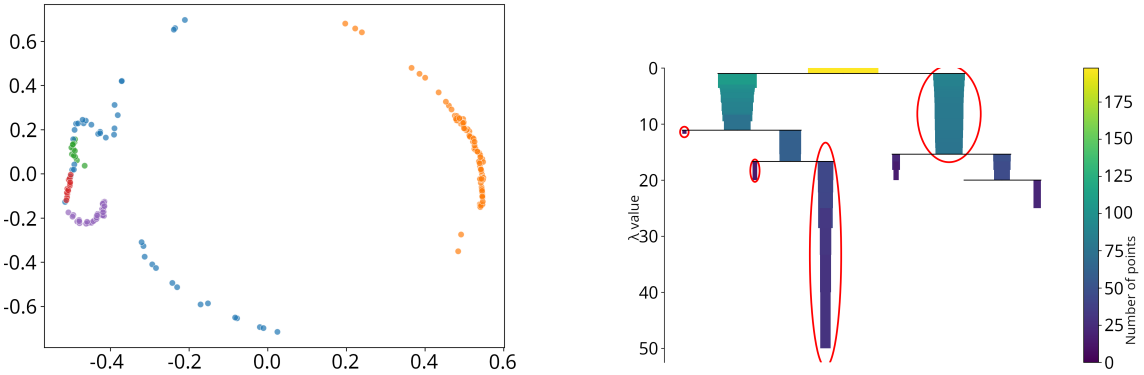
Figure B.2: Hyperparameter Tests, R = 3



Min-Cluster Size = 3, Min-Samples = 5

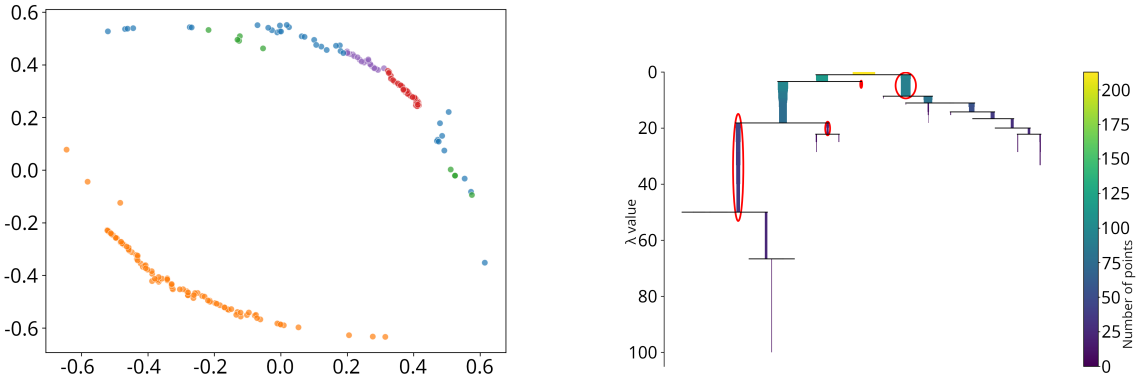


Min-Cluster Size = 5, Min-Samples = 5

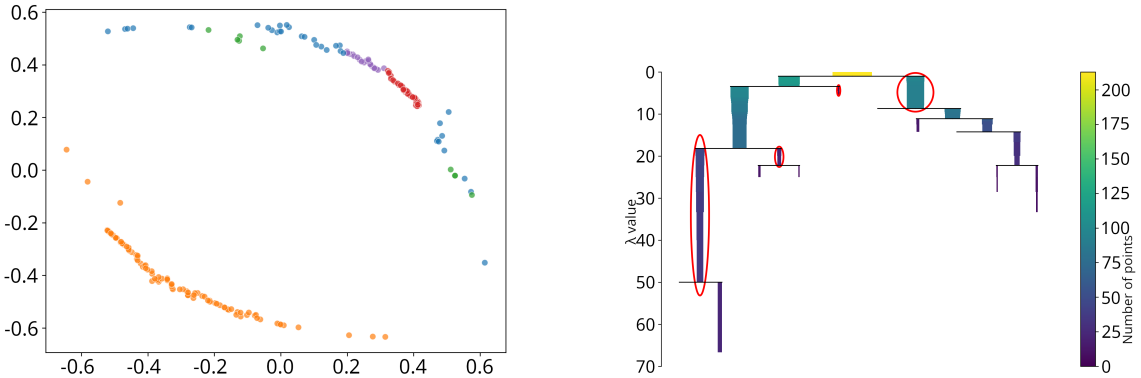


Min-Cluster Size = 10, Min-Samples = 5

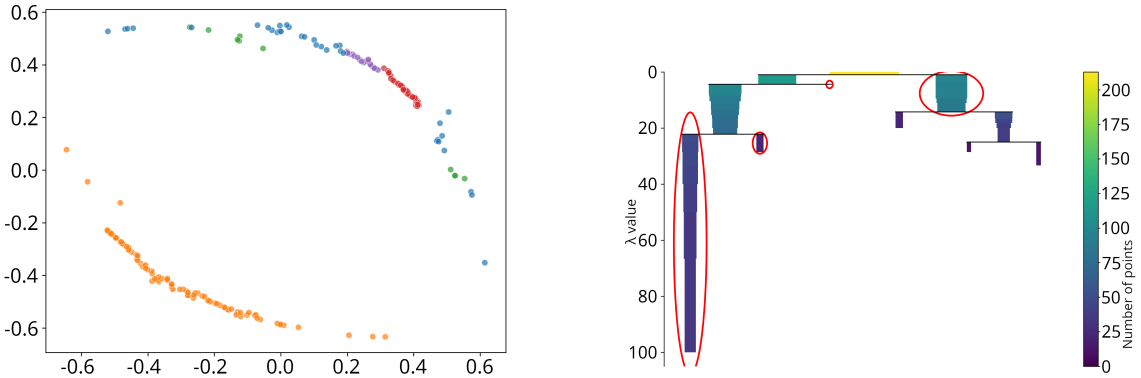
Figure B.3: Hyperparameter Tests, R = 5



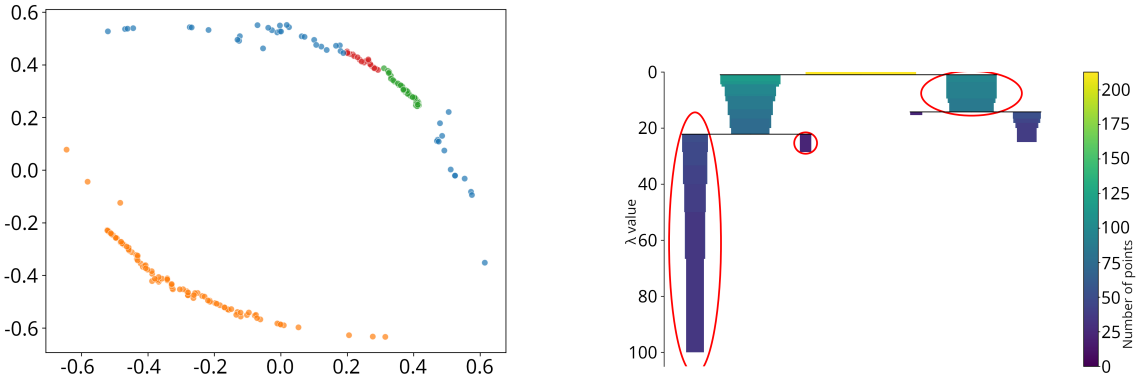
Min-Cluster Size = 3, Min-Samples = 5



Min-Cluster Size = 5, Min-Samples = 5

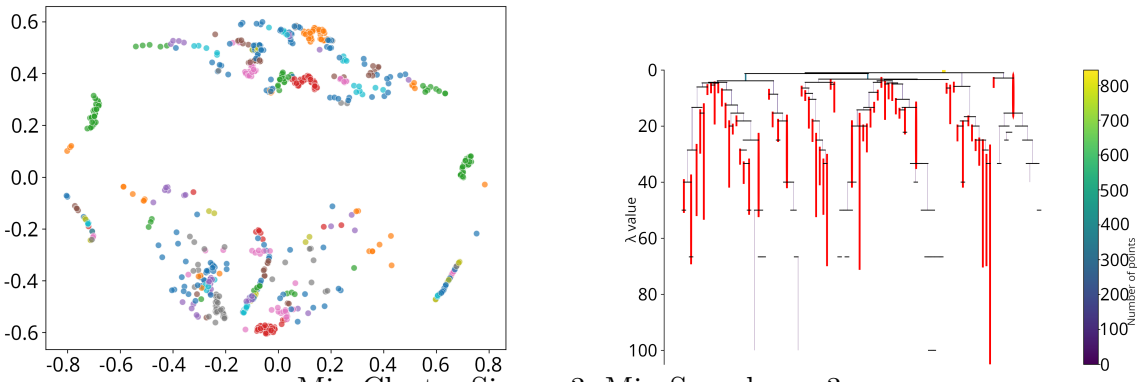


Min-Cluster Size = 10, Min-Samples = 3

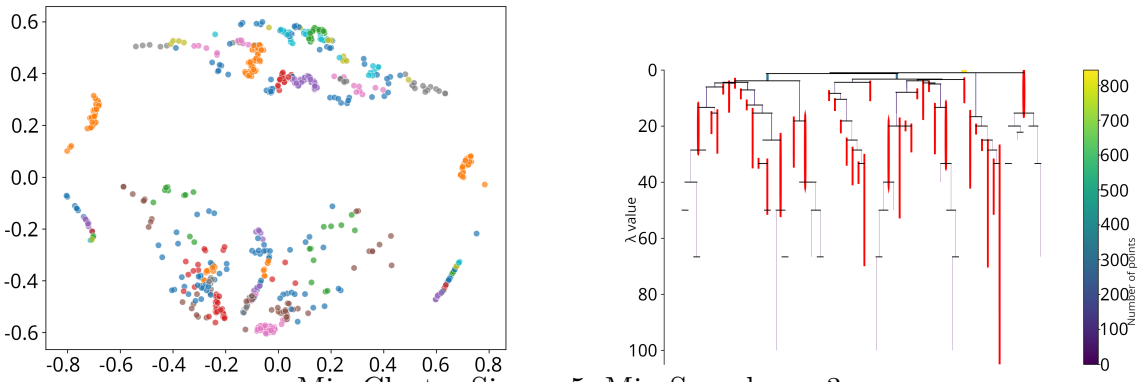


Min-Cluster Size = 15, Min-Samples = 3

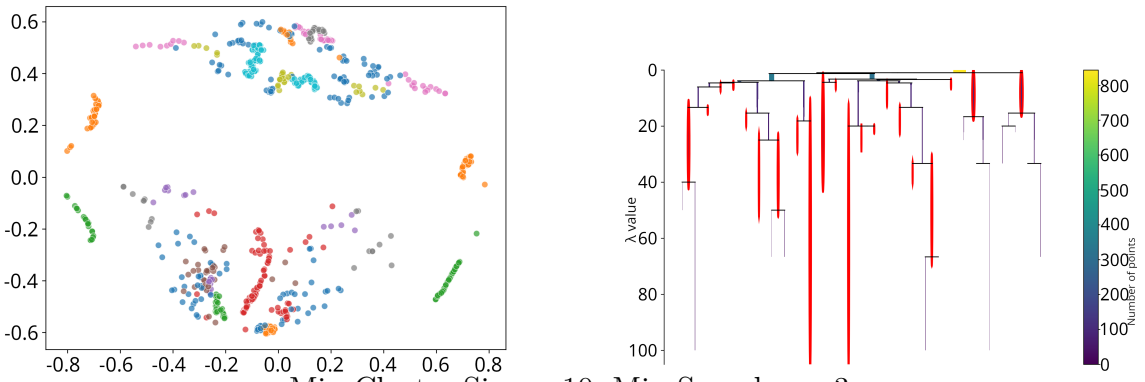
Figure B.4: Hyperparameter Tests, R = 10



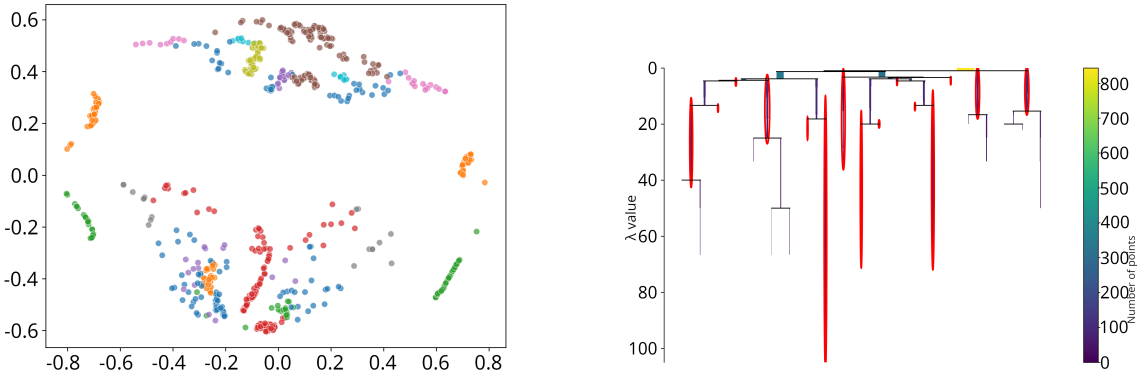
Min-Cluster Size = 3, Min-Samples = 3



Min-Cluster Size = 5, Min-Samples = 3



Min-Cluster Size = 10, Min-Samples = 3



Min-Cluster Size = 15, Min-Samples = 3

Figure B.5: Hyperparameter Tests, R = 20