



NORTHWESTERN UNIVERSITY

Computer Science Department

Technical Report
Number: NU-CS-2026-31

June, 2026

**PhD Dissertation: Bridging Gaps in Operational Cyber Defense: Adaptive
Intrusion Detection and Realistic Evaluation**

Lingzhi Wang

Abstract

Advanced Persistent Threats (APTs) have emerged as one of the most consequential categories of cyberattacks, causing widespread damage to enterprise infrastructure, critical systems, and national security worldwide. In response, the research community has made substantial progress in APT detection and defense, with provenance-based intrusion detection systems (PIDS), which model system activity as causal graphs to capture the multi-stage nature of APT campaigns, establishing themselves as a promising analysis paradigm. Despite these advances, two practical challenges continue to stand in the way of academic research and operational practice, blocking the real-world impact of APT defense. The first is limited deployability at scale: existing detectors struggle to contend with extremely large provenance graphs, streaming and highly fluctuating workloads, and continuously evolving system and attack semantics. The second is inadequate realistic testing and evaluation: the field lacks systematic, scalable methods for generating realistic APT attack scenarios and is further constrained by the scarcity of comprehensive, up-to-date benchmarks, with both deficiencies inflating reported performance and obscuring real progress. This dissertation addresses these two challenges through machine learning and artificial intelligence, with the dual aim of making provenance-based APT detection viable under real-world operational constraints and of advancing the evaluation infrastructure on which the broader research community depends. Concretely, this dissertation defines, analyzes,

and proposes solutions for: (1) deployability challenges in provenance-based intrusion detection, including detection efficiency, online adaptability to evolving system and attack patterns, and consistency under bursty and streaming workloads; and (2) the lack of realistic and rigorous testing and evaluation, including systematic generation of realistic, causality-preserving APT attack scenarios and the construction of comprehensive, realistic, and up-to-date benchmark datasets for APT intrusion detection. The work is organized around five interrelated research projects spanning system design, algorithm development, and empirical evaluation. The dissertation concludes with a discussion of open problems and promising directions for future research.

Keywords

Provenance-Based Intrusion Detection, Adversary Emulation, Advanced Persistent Threats

NORTHWESTERN UNIVERSITY

Bridging Gaps in Operational Cyber Defense: Adaptive Intrusion Detection and Realistic
Evaluation

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Lingzhi Wang

EVANSTON, ILLINOIS

September 2026

© Copyright by Lingzhi Wang 2026

All Rights Reserved

ABSTRACT

Advanced Persistent Threats (APTs) have emerged as one of the most consequential categories of cyberattacks, causing widespread damage to enterprise infrastructure, critical systems, and national security worldwide. In response, the research community has made substantial progress in APT detection and defense, with provenance-based intrusion detection systems (PIDS), which model system activity as causal graphs to capture the multi-stage nature of APT campaigns, establishing themselves as a promising analysis paradigm. Despite these advances, two practical challenges continue to stand in the way of academic research and operational practice, blocking the real-world impact of APT defense. The first is *limited deployability at scale*: existing detectors struggle to contend with extremely large provenance graphs, streaming and highly fluctuating workloads, and continuously evolving system and attack semantics. The second is *inadequate realistic testing and evaluation*: the field lacks systematic, scalable methods for generating realistic APT attack scenarios and is further constrained by the scarcity of comprehensive, up-to-date benchmarks, with both deficiencies inflating reported performance and obscuring real progress.

This dissertation addresses these two challenges through machine learning and artificial intelligence, with the dual aim of making provenance-based APT detection viable under real-world operational constraints and of advancing the evaluation infrastructure on which the broader research community depends. Concretely, this dissertation defines, analyzes, and proposes solutions for: (1) deployability challenges in provenance-based intrusion detection, including detection efficiency, online adaptability to evolving system and attack patterns, and consistency under bursty and streaming workloads; and (2) the lack of realistic and rigorous testing and evaluation, including systematic generation of realistic, causality-preserving APT attack scenarios and the construction of comprehensive, realistic, and up-to-date benchmark datasets for APT intrusion detection.

The work is organized around five interrelated research projects spanning system design, algorithm development, and empirical evaluation. The dissertation concludes with a discussion of open problems and promising directions for future research.

ACKNOWLEDGMENTS

The completion of this dissertation would not have been possible without the many people who supported me along the way.

First and foremost, I am deeply grateful to my advisor, Professor Yan Chen. He gave me this opportunity to pursue my Ph.D. at Northwestern and introduced me to the research on APT that has come to define my work. Beyond his guidance at every critical academic turn and his instincts on research direction, I am thankful for his care and support throughout these years, from providing free housing in China during the pandemic to the warm Thanksgiving dinners every year.

I am also grateful to my committee members, Han Liu, Xinyu Xing, and Zhenkai Liang, for their service, feedback, and questions. I had the privilege of exchanging ideas with each of them, and those conversations shaped my thinking on APT detection, the use of classical planning, and the construction of cyberattack datasets. I also owe special thanks to the mentors and collaborators who guided me at different stages of my Ph.D.: R. Sekar at Stony Brook University, who provided enormous help on the CAPTAIN project; Vinod Yegneswaran and Ashish Gehani at SRI International; Jelena Mirkovic at the University of Southern California; and Zhenyuan Li at Zhejiang University. I also thank Professors Yan Chen, R. Sekar, Vinod Yegneswaran, Xinyu Xing, and Han Liu for their generous help during my job search.

To my fellow lab mates and the master's students I had the pleasure of working with, particularly Xiangmin Shen, Ziyu Li, and Xinyi Shi, thank you for the camaraderie. A special thanks to Xiangmin, with whom I shared the highs and lows of Ph.D. life, leaned on for mutual support, and spent many memorable hours on the tennis court.

To my parents and family in China, who supported me wholeheartedly from afar, and to my girlfriend Yawen Hu, who listened patiently through my most anxious moments and lifted me up

both emotionally and practically, words cannot express my gratitude. To the friends I made at Northwestern, especially those I met through theater productions, thank you for your energy and youthful spirit. Those experiences and friendships gave me far more than I expected.

Finally, I gratefully acknowledge the funding agencies, including the National Science Foundation (NSF), that supported portions of this work.

LIST OF ABBREVIATIONS

AEAS	Actionable Exploit Assessment System
AI	Artificial Intelligence
API	Application Programming Interface
APT	Advanced Persistent Threat
ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge (MITRE framework)
BAS	Breach and Attack Simulation
C2 / C&C	Command and Control
CISA	Cybersecurity and Infrastructure Security Agency
CoT	Chain-of-Thought (prompting technique)
CTI	Cyber Threat Intelligence
CV	Coefficient of Variation
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DAG	Directed Acyclic Graph
DARPA	Defense Advanced Research Projects Agency
DLL	Dynamic Link Library
DNS	Domain Name System
eBPF	Extended Berkeley Packet Filter
EDR	Endpoint Detection and Response
EPSS	Exploit Prediction Scoring System
ETW	Event Tracing for Windows
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
IaC	Infrastructure as Code
IDS	Intrusion Detection System
IP	Internet Protocol
JSON	JavaScript Object Notation
LLM	Large Language Model

LOLBAS	Living Off the Land Binaries and Scripts
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
NDSS	Network and Distributed System Security Symposium
NSA	National Security Agency
NSF	National Science Foundation
OpTC	Operationally Transparent Computing (DARPA program)
OS	Operating System
PDDL	Planning Domain Definition Language
PID	Process Identifier
PIDS	Provenance-Based Intrusion Detection System
POMDP	Partially Observable Markov Decision Process
PPID	Parent Process Identifier
PTES	Penetration Testing Execution Standard
RAG	Retrieval-Augmented Generation
SOC	Security Operations Center
SOTA	State-of-the-Art
SSH	Secure Shell
TC	Transparent Computing (DARPA program)
TDSC	IEEE Transactions on Dependable and Secure Computing
TIBER-EU	Threat Intelligence-Based Ethical Red Teaming (European framework)
TN	True Negative
TP	True Positive
TPR	True Positive Rate
TTP	Tactics, Techniques, and Procedures
UUID	Universally Unique Identifier
VM	Virtual Machine
YAML	YAML Ain't Markup Language

Glossary

Advanced Persistent Threat (APT) A prolonged, targeted cyberattack in which a well-resourced adversary gains unauthorized access to a network and remains undetected for an extended period to achieve specific objectives such as data exfiltration, espionage, or sabotage. APT actors are typically nation-state-sponsored or highly organized criminal groups that continuously evolve their techniques to evade detection.

Attack Emulation The systematic reproduction of real-world cyberattack behavior—based on threat intelligence reports or structured frameworks such as MITRE ATT&CK—in a controlled environment. Attack emulation is used to generate realistic datasets for training and evaluating intrusion detection systems, and to stress-test defensive tools against up-to-date adversary behavior, while preserving the causal structure of the original attacks.

Audit Log A chronological, kernel-level record of system events generated by the operating system, including process creation and termination, file reads and writes, network connections, and inter-process communication. Audit logs are the primary raw data source from which provenance graphs are constructed for host-based intrusion detection.

Classical Planning A subfield of artificial intelligence concerned with automatically finding a sequence of actions that transforms an initial world state into a desired goal state, given a formal model of available actions with their preconditions and effects. Planning problems are commonly specified using the Planning Domain Definition Language (PDDL). In this

dissertation, classical planning is used to structure automated penetration testing as a goal-directed search problem.

Cyber Threat Intelligence (CTI) Structured or unstructured information about adversary tactics, techniques, procedures, infrastructure, and motivations, derived from the analysis of past cyberattacks. CTI reports are commonly used to guide defensive prioritization, inform red team planning, and drive attack emulation by providing ground-truth descriptions of how specific threat actors operate.

Graph Neural Network (GNN) A class of neural networks designed to operate on graph-structured data by iteratively aggregating feature information from neighboring nodes to produce node- or graph-level representations. In provenance-based intrusion detection, GNNs are applied to provenance graphs to learn behavioral patterns of system entities for the purpose of anomaly detection and attack classification.

Kill Chain A conceptual model describing the sequential stages of a cyberattack from initial reconnaissance to final objective completion. In cybersecurity, the term is used to characterize the phases of an APT campaign—such as reconnaissance, initial access, lateral movement, privilege escalation, persistence, and exfiltration—and to reason about where in the sequence a defense can most effectively intervene.

MITRE ATT&CK Framework A publicly accessible, continuously updated knowledge base of adversary tactics, techniques, and procedures (TTPs) observed in real-world cyberattacks, maintained by MITRE Corporation. It provides a structured taxonomy for describing attacker behavior across the full attack lifecycle and is widely used for threat intelligence, red team planning, detection engineering, and security product evaluation.

Penetration Testing A security assessment methodology in which authorized practitioners simulate real-world attacks against a target system to identify exploitable vulnerabilities before malicious actors can. Unlike automated scanning, penetration testing involves manual, goal-directed attack chains that mimic adversary behavior and typically culminates in a detailed report of discovered vulnerabilities and recommended mitigations.

Procedure In the MITRE ATT&CK framework, a procedure is the concrete, observed implementation of a technique by a specific threat actor or malware tool—for example, the exact command-line invocation or script used in a documented attack. Procedures describe *what exactly* was done in a real incident, and serve as the finest level of granularity in the ATT&CK taxonomy.

Provenance Graph A directed graph derived from system audit logs in which nodes represent system entities—such as processes, files, and network sockets—and directed edges represent interactions between them, such as file reads, process creation, and network connections. Provenance graphs explicitly encode the causal and temporal dependencies among system events, making them well-suited for tracking multi-stage attack behavior and supporting forensic investigation.

Red Teaming A structured security exercise in which a dedicated adversarial team simulates the tactics, techniques, and procedures of a real threat actor to evaluate the detection and response capabilities of an organization's defenses. Red teaming goes beyond penetration testing by emulating the behavior of specific adversaries over an extended, realistic engagement scenario.

Tactic In the MITRE ATT&CK framework, a tactic represents an adversary's high-level objective

during a specific phase of an attack—for example, Initial Access, Persistence, Lateral Movement, or Exfiltration. Tactics answer the question *why* the adversary is performing a given action, and the ATT&CK framework currently defines 14 enterprise tactics.

Technique In the MITRE ATT&CK framework, a technique describes a specific method an adversary uses to accomplish a tactic—for example, Spearphishing Attachment (under Initial Access) or OS Credential Dumping (under Credential Access). Techniques answer the question *how* the adversary achieves a tactical goal, and may have associated sub-techniques that describe more granular variations.

TABLE OF CONTENTS

Abstract	3
Acknowledgments	5
List of Abbreviations	7
Glossary	9
List of Figures	21
List of Tables	23
Chapter 1: Introduction	25
1.1 Motivation: Operational Challenges in Defending Against Advanced Persistent Threats	25
1.2 Two Operational Challenges in APT Defense	28
1.2.1 Deployability Challenges in Provenance-Based Intrusion Detection	28
1.2.2 Inadequate Realistic Testing and Evaluation	29
1.3 Thesis Statement and Contributions	31
1.4 Thesis Roadmap	33

Chapter 2: Background and Related Work	34
2.1 Advanced Persistent Threats and the MITRE ATT&CK Framework	34
2.2 Provenance-Based Intrusion Detection	35
2.2.1 Provenance Graphs and Their Construction	35
2.2.2 Rule-Based, Learning-Based, and Hybrid Detectors	36
2.3 Testing and Evaluation of APT Defense	38
2.3.1 Adversary Emulation	38
2.3.2 Benchmark Datasets in Intrusion Detection Research	41
Chapter 3: Operational Gaps in Provenance-Based Intrusion Detection	43
3.1 Lightweight and Adaptive Detection via Gradient-Guided Rules	43
3.1.1 Motivation and Threat Model	43
3.1.2 Motivation Examples	47
3.1.3 Design	50
3.1.3.1 System Overview	50
3.1.3.2 Adaptive Detection Module	50
3.1.4 Learning Module	55
3.1.4.1 Loss Function	56
3.1.4.2 Differentiable Detection Framework	57
3.1.4.3 Training and Testing	60
3.1.5 Evaluation	61

3.1.6	Experiment Settings	62
3.1.6.1	Datasets	62
3.1.6.2	Experiment Setup	63
3.1.6.3	Baseline Detectors	64
3.1.7	Detection Accuracy	65
3.1.7.1	False Alarm Events Reduction	65
3.1.7.2	False Alarm Entity Reduction	67
3.1.8	Efficiency	69
3.1.8.1	Detection Latency	69
3.1.8.2	Runtime Overhead	71
3.1.9	Resilience Against Adversarial Attacks	73
3.1.9.1	Adversarial Mimicry Attack	73
3.1.9.2	Training Set Poisoning Attack	75
3.1.10	Ablation Study	76
3.1.10.1	Individual Adaptive Parameters	76
3.1.10.2	Learning Rate in Optimization	76
3.1.11	Case Study	77
3.1.11.1	Better Detection Capability due to Fine-Grained Rules	77
3.1.11.2	Robustness brought by Differentiable Tag-Propagation Framework	79
3.1.12	Discussion	80

	16
3.1.12.1 Overfitting in Parameter Learning	80
3.1.12.2 More Optimization Algorithms and Online Learning	80
3.1.13 Conclusion	81
3.2 Online, Adaptive Provenance-Based Intrusion Detection	81
3.2.1 Motivation	82
3.2.2 Online Learning Module	82
3.2.3 Evaluation Highlights	83
3.2.4 Section Summary	85
3.3 Burst-Resilient Real-Time Detection via Serverless GNN Inference	85
3.3.1 Motivation and Threat Model	85
3.3.2 Background	87
3.3.3 Design	89
3.3.4 Evaluation	91
3.3.5 Conclusion and Future Work	92
3.4 Chapter Summary	93
Chapter 4: Realistic Testing and Evaluation Infrastructure	94
4.1 AURORA: Causality-Preserving Cyberattack Emulation with LLM-Powered Classical Planning	95
4.1.1 Motivation and Technical Challenges	95
4.1.2 Threat Model and Scope	97

4.1.3	Background: Classical Planning	98
4.1.4	Cyberattack Planning	99
4.1.4.1	Attack Actions	99
4.1.4.2	Attack Action Linking Model	101
4.1.5	System Design	104
4.1.5.1	Attack Action Formalization	105
4.1.5.2	Attack Planning	107
4.1.5.3	Attack Execution	111
4.1.6	Evaluation	111
4.1.6.1	Evaluation Setup	111
4.1.6.2	Success Rate of Emulated Attacks	112
4.1.6.3	Causality in Emulated Attacks	114
4.1.6.4	Customization of Attack Emulation	116
4.1.6.5	Diversity of Emulated Cyberattacks	119
4.1.6.6	Evaluating Existing Detection Systems	120
4.1.6.7	Case Study	121
4.1.6.8	Ablation Study	123
4.1.7	Discussion and Limitations	124
4.1.8	Section Summary	124
4.2	Building Next-Generation Benchmark Datasets for Provenance-Based Intrusion Detection	125

4.2.1	Motivation: The Dataset Gap in PIDS Research	125
4.2.2	Requirements for Modern APT Benchmark Datasets	127
4.2.2.1	Time-Weighted Detection Accuracy for Real-Time APT Detection	127
4.2.2.2	TTP Coverage at a Quality Threshold	128
4.2.2.3	Requirements for the Dataset.	130
4.2.3	Dataset Generation Pipeline	130
4.2.3.1	Leveraging Aurora for Attack Emulation	131
4.2.3.2	Background Workload and Environment Setup	131
4.2.3.3	Data Collection and Formalization	134
4.2.3.4	Labeling and Ground Truth	135
4.2.4	Analysis of the Generated Data	137
4.2.4.1	Evaluation Setup.	137
4.2.4.2	Detection Performance and Analysis	138
4.2.4.3	Time-Weighted Recall Analysis	140
4.2.5	Discussion and Limitations	143
4.2.5.1	Experimental Design and Evaluation Sensitivity	143
4.2.5.2	Scale and Diversity of Attack Traces	144
4.2.5.3	Benign Behavior Coverage and Data Editing	144
4.2.5.4	Future Work	145
4.2.6	Section Summary	145

4.3 Chapter Summary	146
Chapter 5: Conclusion and Future Work	147
5.1 Summary of Contributions	147
5.2 Reflections on the Operational-Gap Framing	149
5.3 Limitations	150
5.4 Future Research Directions	151
References	166
Appendix A: CAPTAIN Supplementary Materials	168
A.1 Attack Scenarios in Datasets	168
A.2 CAPTAIN Policies	169
A.3 Mathematical Proof of the Optimization	170
A.4 Mathematical Proof of the Heuristic Hyperparameter Setting	171
A.5 Additional Experiment Results	172
A.5.1 Gradients Storage During Training	172
A.5.2 Mimicry Attack Experiment	173
A.5.3 Ablation Study on Learning Rate	174
Appendix B: GraphFaaS Supplementary Materials	175
B.1 Algorithm for Subgraph Partition & Packing	175

Appendix C: AURORA Supplementary Materials	177
C.1 Searching for Multiple Attack Chains using Symbolic Planners	177
C.2 Supplementary Experimental Results	177
Vita	180

LIST OF FIGURES

2.1	A brief overview of the workflow and commonly used techniques in mainstream PIDS.	35
3.1	Three motivating examples from the real-world dataset where more fine-grained rules are needed in the rule-based PIDS.	47
3.2	The overall framework of CAPTAIN. Phases ①-⑤ show the lifecycle of the detection module and the learning module within one training epoch.	50
3.3	Comparison of resource consumption when detecting on DARPA Engagement 3 CADETS. In Fig. 3.3b, the memory usage curve of CAPTAIN and MORSE are overlapped, showing their similar efficiency performance.	72
3.4	Adversarial mimicry attack against CAPTAIN and FLASH (use the attack entity /tmp/test as an example)	74
3.5	Ablation study on each parameter.	77
3.6	Case study from Engagement 3 TRACE. We omitted some irrelevant details to keep the graphs clean and succinct.	78
3.7	Cumulative false positives of CAPTAIN + vs. CAPTAIN over time, measured by the number of processed events, on three DARPA Engagement 3 datasets.	84
3.8	Fluctuations in detection workload over time in the DARPA TC dataset [54]	86
3.9	Overview of GraphFaaS Framework with three components: (i) graph construction, (ii) serverless node-level embedding, and (iii) serverless GNN inference. . . .	89

3.10	Detection latency over time. GraphFaaS: mean = 2.10, STD (standard deviation) = 1.09, CV (coefficient of variation) = 0.52; Baseline: mean = 14.16, STD = 4498.92, CV = 1.46.	92
4.1	Causality in emulated attack chains. Without explicit modeling of inter-step dependencies, automated systems produce sequences of actions that do not reflect how real attackers chain operations together.	96
4.2	An attack chain example connected by AALM.	104
4.3	System overview.	105
4.4	An example showing how AURORA analyzes preconditions and effects of an attack action using AALM.	106
4.5	Comparison of shared artifact rates.	116
4.6	Fidelity of emulated attack chains to source CTI reports.	117
4.7	Customized attacks for different environments.	118
4.8	The case study showing how AURORA constructs a complete attack chain for three “Silver Fox” attack actions.	122
A.1	The distribution of the numbers of non-zero gradients to A and G that each nodes need to save during tag propagation.	173
A.2	Training loss using different learning rates.	174

LIST OF TABLES

2.1	Comparison of existing PIDS. Buffer time refers to the waiting time before threat analysis.	38
2.2	Comparison of existing approaches to adversary emulation along the four desired properties.	40
3.1	Tags in CAPTAIN	51
3.2	Comparison with the baselines on the TRACE dataset from Engagement 3 in terms of false alarm events	66
3.3	Comparison with the baseline on different datasets regarding false alarms. T-3, T-5, C-3, and C-5 represent the TRACE and CADETS datasets from Engagement 3 & 5. S-1 to S-3 denote Cloud, Streaming, and Dev datasets from the SOC.	66
3.4	Comparison with the baselines in terms of node-level detection accuracy	67
3.5	Comparison of detection latency	71
4.1	Attributes of attack actions	100
4.2	Comparison of success rates(with 95% confidence intervals).	114
4.3	Comparison of causality preservation.	115
4.4	Comparison of cyberattack dataset scale and TTP diversity.	119

4.5	Comparison of detection performance against attacks generated by AURORA (with 95% bootstrap confidence intervals).	120
4.6	Detection results on PROVATTACK datasets	139
4.7	Real-time APT Detection recall(%) on generated datasets	141
A.1	Simulated Scenario Summary	169
A.2	Tag propagation policies	169
A.3	Alarm generation policies	170
C.1	Average Time and Monetary Costs of AURORA	178
C.2	The Performance Comparison of Attack Technique Extraction between LADDER and AURORA	178

CHAPTER 1

INTRODUCTION

1.1 Motivation: Operational Challenges in Defending Against Advanced Persistent Threats

Advanced Persistent Threats (APTs) have emerged as one of the most severe categories of cyberattacks confronting modern organizations. Compared to traditional intrusions, APTs are deliberate, well-resourced campaigns conducted by sophisticated adversaries pursuing specific objectives, ranging from large-scale data exfiltration and intellectual property theft to long-term system compromise and sabotage. In recent years, APTs have been causing severe damages to the digital infrastructures around the world. The 2013 Target data breach, where intrusion alerts were correctly generated by security tools but overlooked by overwhelmed analysts, demonstrated that even timely detection is insufficient without operational processes capable of acting on it [1]. The 2017 Equifax breach exposed the personal and financial data of approximately 148 million individuals, illustrating the catastrophic privacy and economic implications of a successful APT campaign [2]. The 2020 SolarWinds supply-chain compromise saw a sophisticated adversary insert a backdoor into the SolarWinds Orion software update channel, reaching thousands of customer organizations including multiple U.S. federal agencies and remaining undetected for approximately nine months before public disclosure [3], [4]. More recently, the Volt Typhoon campaign, attributed by U.S. and allied cybersecurity agencies to a state-sponsored actor, demonstrated how an adversary can use “living-off-the-land” techniques to sustain stealthy access to U.S. critical infrastructure across the communications, energy, and water sectors [5]. The scale of the APT threat has driven explosive investment in endpoint detection and response (EDR) technologies: the market capitalization of the

top 37 EDR companies exceeded \$320 billion by 2022, with the broader EDR market projected to surpass \$3.7 billion in size [6]. Yet despite this investment, APT attacks continue to succeed at scale, underscoring a persistent gap between the sophistication of existing defense research and the demands of real-world operational deployments.

Detecting APT attacks is fundamentally difficult for several compounding reasons. First, APT adversaries operate methodically and patiently, executing their campaigns through multiple stages, reconnaissance, initial access, lateral movement, privilege escalation, persistence, and exfiltration, that may unfold over weeks or months [7]. Each individual step may appear benign in isolation; only by correlating events across time, processes, and system components does the attack become visible. Second, attackers continuously refine their techniques to evade detection: by exploiting legitimate system utilities (“living-off-the-land” techniques), blending malicious actions within normal system activity, and deliberately minimizing their operational footprint. Third, the sheer volume of telemetry generated in modern enterprise environments makes manual analysis untenable. Security analysts in large organizations are estimated to handle an average of 5,000 alerts per day, the vast majority of which are false positives [8]; organizations as a whole may receive upwards of 100,000 alerts daily [9]. This alarm fatigue creates conditions under which correctly generated alerts are missed or deprioritized, as the Target breach demonstrated, while the cognitive burden of sifting through thousands of daily alerts degrades capacity for effective incident response. Traditional network-based detection and prevention systems, designed for earlier threat models, are structurally ill-suited to the temporal complexity and multi-stage nature of modern APT kill chains [6].

Provenance-based intrusion detection systems (PIDS) have emerged as a promising paradigm for addressing this detection challenge, attracting sustained attention from both academia and industry [8], [9]. In this approach, raw system audit logs are transformed into *provenance graphs*, di-

rected acyclic graphs in which nodes represent system entities such as processes, files, and network sockets, and directed edges represent interactions between them such as file reads, process creation events, and network connections. By explicitly encoding the causal and temporal dependencies among system activities, provenance graphs provide a natural structure for reasoning about multi-stage attack behavior that spans long time horizons. Two complementary families of PIDS have been developed within this paradigm. *Rule-based* systems [8], [9] operate on expert-defined propagation rules to assign trust scores to system entities and trigger alarms; they are favored for their low runtime overhead, real-time detection capability, and interpretable outputs. *Embedding-based* systems employ machine learning techniques, including Graph Neural Networks (GNNs) [10] and graph/word embedding methods, to learn complex behavioral patterns, achieving stronger detection performance at the cost of higher computational requirements, longer detection latency, and reduced interpretability. Together, these approaches have established PIDS as a central and active research direction in host-based security.

Despite this progress, the translation of PIDS research, and APT defense research more broadly, into operational practice remains hampered by several unresolved challenges. This dissertation identifies two such challenges. The first concerns the *practical deployment of detection systems*: existing rule-based PIDS rely on static, manually configured parameters that cannot adapt to the diversity of real-world environments, producing high false-positive rates that exacerbate alarm fatigue; embedding-based systems impose computational overheads and detection latencies that conflict with the real-time demands of security operations centers; and neither family adequately handles the highly irregular and bursty workload patterns characteristic of production network environments [8], [9], [10]. The second challenge concerns the *realistic testing and evaluation*: the field lacks both systematic, scalable methods for generating realistic, causally coherent APT attack scenarios and the comprehensive, up-to-date benchmark datasets needed to evaluate detection

methods credibly, with both deficiencies inflating reported detection performance and obscuring real progress [6], [11], [12]. Section 1.2 examines each of these challenges in detail and articulates the specific contributions of this dissertation toward addressing them.

1.2 Two Operational Challenges in APT Defense

1.2.1 Deployability Challenges in Provenance-Based Intrusion Detection

Provenance-based intrusion detection sits at the intersection of a fundamental tension: accuracy versus deployability. The two dominant families of PIDS each resolve this tension differently, and each pays a price.

Embedding-based PIDS, systems such as FLASH [13], KAIROS [14], and SHADEWATCHER [15], apply Graph Neural Networks and graph embedding techniques to provenance graphs and achieve strong detection accuracy. However, they require log data to be buffered into fixed-length time windows before inference can begin. In the worst case, after an attack is launched, the system must wait out the entire window before detection starts. Beyond this latency penalty, embedding-based systems carry high memory and compute costs from caching models and graph embeddings, making large-scale deployment impractical in resource-constrained SOC environments [8]. Their outputs are also difficult to interpret: most systems report a malicious probability score without indicating which specific events triggered the alarm, forcing analysts to manually re-examine entire log traces.

Rule-based PIDS, such as SLEUTH [16] and MORSE [17], take the opposite approach: they propagate hand-crafted tags along provenance graph edges, flagging entities that accumulate suspicious scores. This makes them lightweight, real-time, and interpretable. The problem is that their rules are universal and static, the same thresholds and propagation parameters applied across all environments. In practice, benign system activity varies considerably across organizations and

workloads, so a rule that works well in one environment produces excessive false positives in another. Security analysts already handle an estimated 5,000 alerts per day on average, most of them false positives; tuning rule-based PIDS manually for each deployment adds yet more burden [9].

Beyond the accuracy-efficiency tradeoff, a second deployment challenge has received little attention: the inherently *bursty* nature of real-world detection workloads. Malicious activity accounts for only a small fraction of overall system events, resulting in highly irregular, spike-prone analysis loads. As shown in real DARPA TC traces, the number of system entities requiring analysis can vary by orders of magnitude within a single day [10]. Statically provisioned GNN inference pipelines, sized for average load, either over-provision and waste resources during quiet periods, or become bottlenecks during spikes, missing the real-time detection window when it matters most. Reducing average detection latency by $6.7\times$ and stabilizing latency variance (64% reduction in coefficient of variation) requires rethinking how GNN inference is resourced [10].

1.2.2 Inadequate Realistic Testing and Evaluation

Robust defense depends on realistic testing and evaluation: knowing how detectors behave under APT-level attacks and being able to compare detection methods on representative ground truth. The infrastructure supporting these activities is inadequate on two related fronts, the generation of realistic attack scenarios and the construction of credible benchmark datasets, and the two deficiencies compound one another.

The first deficiency is the lack of a scalable mechanism for producing realistic, causally coherent attack scenarios against which detectors can be exercised. A useful attack emulation system must be: *actionable*, providing concrete commands and tools so that attacks can be actively replayed against live systems rather than existing only as static log traces; *causally coherent*, ensuring that each attack step is executed based on the artifacts produced by the previous step, as a real

attacker would operate; *customizable*, generating attack chains tailored to a specific target environment or threat actor, not just fixed scripts; and *scalable*, incorporating new ATT&CK techniques as they emerge without requiring manual re-engineering.

No existing system satisfies all four properties simultaneously. Human-orchestrated red team campaigns (e.g., DARPA TC, OpTC) are causally coherent but produce non-replayable static traces. Single-point tools such as Atomic Red Team execute isolated actions but cannot chain them into causally coherent multi-stage attacks. Procedural orchestration tools (e.g., Caldera, PurpleSharp) support automation and scale but do not model inter-step dependencies, producing attack chains that do not reflect real adversary behavior. Expert-crafted playbooks from MITRE Engenuity evaluations are high-fidelity but require months of manual effort and cover only a handful of threat actors [11].

The second deficiency is in the benchmark datasets used to train and evaluate PIDS. The datasets the community has relied on for years, DARPA Transparent Computing (TC), OpTC, ATLAS, were produced in controlled lab environments using attack techniques that were current at the time of collection. They now have three problems. First, the attack scenarios they capture are outdated. APT tactics evolve continuously; techniques that were representative in 2018–2021 do not reflect what adversaries deploy today. A detector trained or evaluated on stale attack patterns may appear to perform well while remaining blind to current threats. Second, existing datasets offer coarse-grained labeling. Many provide only high-level annotations, which hosts were attacked, which broad phases occurred, without event-level ground truth. This makes it impossible to precisely evaluate which specific detection rules or model decisions are correct, undermining both model development and failure analysis. Third, ML and GNN-based PIDS trained on these restricted benchmarks tend to overfit: they achieve strong numbers on test splits drawn from the same narrow distribution, but fail to generalize when attack behavior deviates even modestly from

what is in the training set [12]. The dataset bottleneck is thus not merely a measurement inconvenience; it actively inflates reported performance and slows the field’s real progress.

Building new datasets from scratch using human red teamers is prohibitively expensive and slow; it is also exactly what produces non-scalable, non-customizable corpora. The community needs a principled, automated pipeline for generating large-scale, diverse, fine-grained, and up-to-date datasets that can keep pace with the threat landscape [12].

1.3 Thesis Statement and Contributions

This dissertation argues that the two operational challenges described above, deployability issues and a lack of realistic testing and evaluation, are not independent obstacles but a connected system of gaps that together prevent APT defense research from reaching operational maturity. Addressing them requires not only individual technical solutions, but solutions that are designed with real-world constraints in mind and that reinforce one another: realistic attack emulation enables better datasets; better datasets enable more credible detection evaluation; more deployable detectors close the loop from research to practice. To this end, this dissertation presents five research projects that collectively bridge these gaps through machine learning, generative AI, and symbolic reasoning.

Gap I: Deployability Challenges in Provenance-Based Intrusion Detection

- **CAPTAIN , Lightweight, Adaptive Provenance-Based Intrusion Detection [8].** We identify that the accuracy-efficiency tradeoff in rule-based PIDS stems from static, manually configured propagation rules that cannot adapt to environment-specific behavior. CAPTAIN reformulates rule configuration as a differentiable optimization problem and uses gradient descent to automatically tune detection parameters on benign training data, reducing false alarms by over $11\times$ compared to prior rule-based systems while preserving their lightweight, real-time, and interpretable properties.

- **CAPTAIN + , Online, Adaptive Provenance-Based Intrusion Detection [9].** We extend CAPTAIN with an online learning module that continuously updates detection parameters during inference using real-time feedback, enabling the system to adapt to concept drift without periodic retraining. This addresses the continual-learning challenge in long-running production deployments where system behavior evolves over time.
- **GraphFaaS, Burst-Resilient, Real-Time GNN Inference for Intrusion Detection [10].** We identify bursty, irregular detection workloads as an independent deployment bottleneck for GNN-based PIDS that is orthogonal to detection accuracy. GraphFaaS re-architects GNN inference on a serverless platform to scale dynamically with workload, reducing average detection latency by $6.7\times$ and latency variance (CV) by 64% compared to a statically provisioned baseline while maintaining detection accuracy.

Gap II: Realistic Testing and Evaluation Infrastructure

- **AURORA: Actionable, Causality-Preserving Cyberattack Emulation [11].** We define four properties that a practical attack emulation system must satisfy simultaneously, actionable, causally coherent, customizable, and scalable, and show that no existing system achieves all four. AURORA uses LLM-powered classical planning to automatically generate and execute attack chains from CTI reports and MITRE ATT&CK techniques, producing a dataset $15\times$ larger than the leading expert-crafted benchmark from 250 attack reports.
- **Next-Generation Datasets for Provenance-Based Intrusion Detection [12].** We document three critical limitations of existing PIDS benchmarks, outdated attack techniques, coarse-grained labeling, and overfitting-induced performance inflation, and present a dataset construction pipeline built on AURORA that generates large-scale, fine-grained, and up-to-date attack traces. Evaluating eight representative PIDS on this dataset reveals systematic

generalization failures that are invisible on prior benchmarks, underscoring how stale evaluation infrastructure conceals real weaknesses in the field.

1.4 Thesis Roadmap

The remainder of this dissertation is organized as follows.

Chapter 2 provides background on the APT threat landscape, the provenance graph abstraction, the two families of PIDS, and the attack planning and emulation frameworks that underpin the dissertation’s contributions.

Chapter 3 addresses Gap I by presenting three complementary approaches to making PIDS viable under real-world operational constraints: CAPTAIN, which uses gradient descent to automatically tune detection parameters and reduce false positives; CAPTAIN +, which extends CAPTAIN with online continual adaptation to handle concept drift; and GraphFaaS, which re-architects GNN inference on a serverless platform to absorb bursty detection workloads.

Chapter 4 addresses Gap II by presenting two complementary contributions that together form an end-to-end pipeline for realistic testing and evaluation. Section 4.1 introduces AURORA, an LLM-powered classical-planning system that generates actionable, causally coherent, customizable, and scalable cyberattack chains from CTI reports and MITRE ATT&CK techniques, producing a dataset $15\times$ larger than the leading expert-crafted benchmark. Section 4.2 builds on AURORA to construct a large-scale, fine-grained, and continuously updatable PIDS benchmark, documenting three critical limitations of existing benchmarks, outdated attacks, coarse labeling, and overfitting-induced performance inflation, and showing that evaluation on the new dataset reveals systematic generalization failures invisible on prior benchmarks.

Chapter 5 synthesizes the dissertation’s contributions, discusses limitations and open problems, and identifies promising directions for future research.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Advanced Persistent Threats and the MITRE ATT&CK Framework

Advanced Persistent Threats (APTs) are prolonged, targeted cyberattacks carried out by well-resourced adversaries, typically nation-state actors or organized criminal groups, against high-value targets such as critical infrastructure, government agencies, and large enterprises [6]. As discussed in Chapter 1, APT campaigns are characterized by stealth, persistence, and multi-stage execution: an adversary may gain initial access through a single phishing email, then move laterally across hosts, escalate privileges, establish persistence, and exfiltrate data over weeks or months. What distinguishes APTs from commodity malware is this combination of patience and intent: each individual action may look benign in isolation, and the attack only becomes recognizable when events are correlated across time, processes, and system components.

The dominant framework for systematically describing APT behavior is the MITRE ATT&CK Matrix [7], a publicly maintained knowledge base of adversary behavior observed in real-world incidents. The framework organizes adversary behavior along three levels of granularity, collectively known as *tactics*, *techniques*, and *procedures (TTPs)*. A *tactic* represents the adversary's high-level goal during a specific phase of the attack lifecycle, such as *Initial Access*, *Persistence*, *Privilege Escalation*, or *Exfiltration*, with the current Enterprise ATT&CK matrix defining fourteen such tactics covering the full attack lifecycle. A *technique* describes a specific method by which an adversary accomplishes a tactic, for example *Spearphishing Attachment* (under Initial Access) or *OS Credential Dumping* (under Credential Access); many techniques are further broken down

into sub-techniques that capture more granular variations. A *procedure* refers to the concrete, observed implementation of a technique by a specific threat actor or malware tool, such as the exact command-line invocation or script used in a documented incident.

ATT&CK serves as the de facto vocabulary for describing adversary behavior throughout this dissertation: attack scenarios in Section 4.1 are generated and labeled by ATT&CK technique, and the dataset construction pipeline in Section 4.2 uses ATT&CK as its coverage taxonomy.

2.2 Provenance-Based Intrusion Detection

2.2.1 Provenance Graphs and Their Construction

Provenance-based intrusion detection has drawn wide attention for its powerful correlation and causal analysis capabilities. In provenance analysis, the system behaviors are modeled as directed acyclic graphs, called provenance graphs [18], [19], in which nodes represent system entities, such as processes, files, sockets, pipes, memory objects, etc., and edges represent interactions between entities, such as reading files and connecting to a remote host. Fig. 2.1 illustrates an overview of the typical PIDS workflow.

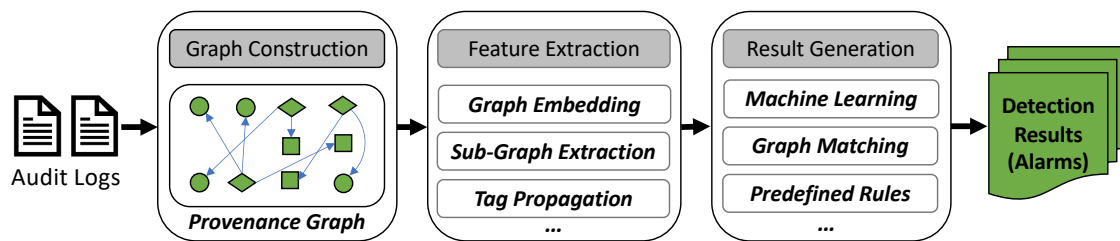


Figure 2.1: A brief overview of the workflow and commonly used techniques in mainstream PIDS.

2.2.2 Rule-Based, Learning-Based, and Hybrid Detectors

Based on how to model and aggregate the entity information, the data/control flow, and the system behaviors, we identify two major groups of PIDS: the embedding-based approach and the rule-based approach. **Embedding-based PIDS.** Embedding-based systems [13], [14], [15], [20], [21], [22], [23], [24] use numerical vectors to represent, propagate, and aggregate the information of system entities, events, data/control flow in the graph. Various techniques in machine learning have been employed to embed the graph features into numerical vectors. Since these techniques typically require the input structured as graphs or paths, embedding-based PIDS usually adopt time windows or log batches in their design, leading to the additional detection delay, which we named as *buffer time* in this dissertation. Concretely, buffer time refers to the time window duration in which all streaming logs are stored to create a graph structure for subsequent processing. Previous studies have implemented a fixed-length window based on time [14] or the number of events [13], while some works [25] utilize the variable length window. Buffer time is crucial in real-world detection scenarios because, in the worst-case scenario, after an attack has been launched, the PIDS would have to wait for the duration of the buffer time before starting the detection procedure.

The methods for embedding graphs differ across various systems. For instance, PROVDETECTOR [26] selects rare paths based on historical event frequency to generate embedding vectors for sequence learning and anomaly detection. ATLAS [23] classifies the extracted sequences from the graph with Long Short-term Memory (LSTM). UNICORN[21] embeds the graph histograms and clusters the embedded graph sketches. WATSON [22] learns the node embedding and event semantics from training traces. More recent embedding-based systems [13], [14], [24] employed graph neural networks in detection. While such systems can achieve better detection accuracy, they fail to provide the rationale for generated alarms. For example, SHADEWATCHER [15] provides the probability of malicious system events without further context. FLASH [13] highlights suspicious

entities with causal links but does not provide further attack semantics of those interactions. In practice, more manual efforts are required to review the relevant audit logs thoroughly for further response.

Rule-based PIDS. Rule-based systems [16], [17], [27], [28], [29] leverage human-designed rules to map system entities and events to predefined semantic units (e.g., tags [16], [17], [27], TTP [28], etc.) and propagate these units along the information flow to capture causality. NODOZE[29] assigns an anomaly score to each edge based on the frequency it appears in the benign data and propagates the scores on the graph. POIROT [30] models the detection as a graph-matching problem that aligns manually constructed query graphs on the provenance graph. SLEUTH [16] and MORSE [17] assign tags to system entities and propagate tags among the graph. Rule-based systems have become popular due to their real-time processing capabilities and lightweight properties. However, they usually suffer low detection accuracy due to the lack of fine-grained rules and environment-specific configurations. Most rule-based PIDS [16], [17], [27], [28] implement simple and universal rules, making them inflexible to adapt to different environments. These systems typically rely on a discrete classification to denote system entity properties, such as private/public, and trusted/untrusted, and apply identical rules to all nodes and events. As a result, they lack the nuance required to accurately distinguish between similar graphs, often being either overly lenient or overly strict, which leads to false alarms or missed true alarms.

Table 2.1 summarizes the detection granularity of existing PIDS and indicates whether the detection requires buffer time.

Table 2.1: Comparison of existing PIDS. Buffer time refers to the waiting time before threat analysis.

	Detection Granularity	Require Buffer Time
MARLIN	Edge	No
FLASH	Node	Yes
R-CAID	Node	Yes
KAIROS	Graph ¹	Yes
NODLINK	Node	Yes
PROGRAPHER	Node	Yes
SHADEWATCHER	Edge	Yes
POIROT	Path	Yes
MORSE	Edge	No
UNICORN	Graph	Yes
HOLMES	Edge	No
PROVDETECTOR	Path	Yes
NODOZE	Path	Yes

¹Although KAIROS triggers alarms on the graph level, it can highlight anomaly nodes and edges in the subgraphs.

2.3 Testing and Evaluation of APT Defense

2.3.1 Adversary Emulation

The continuous evolution of cyberattack techniques makes comprehensive and up-to-date adversary emulation essential for benchmarking defense systems, identifying their weaknesses, and enabling threat-informed defense. In contrast to vulnerability scanning or single-point penetration testing, adversary emulation reproduces the full behavior of known threat actors, multi-stage, multi-host, and operationally coherent, in order to evaluate whether a defense system can detect or contain a realistic APT campaign. Its importance is widely recognized across both industry and government: the European Central Bank’s threat intelligence-based ethical red teaming (TIBER-EU) framework [31], joint purple-team exercises conducted by CISA and the NSA, the annual

MITRE Engenuity ATT&CK Evaluations attracting over 50 vendors worldwide, and a growing commercial sector [32], [33], [34] all attest to the operational demand. Market projections [35] forecast the automated cyberattack emulation market to grow from \$729.2 million in 2024 to over \$2.4 billion by 2029.

A practical adversary emulation system must satisfy four properties simultaneously, properties that are individually achievable but collectively rare in existing systems [11]. *Actionability*: the system must provide concrete commands, tools, and execution scripts so that attacks can be replayed against live systems rather than existing only as static log traces. *Causal coherence*: each attack step must depend on the artifacts and state produced by previous steps, reflecting how real attackers chain actions together rather than emitting isolated, independent events. *Customizability*: the system must adapt attack chains to a specific target environment or to the behaviors of a particular threat actor, rather than relying on a fixed set of manually authored playbooks. *Scalability*: the system must incorporate new ATT&CK techniques as they emerge without architectural changes, and ideally generate diverse attack chains automatically.

Table 2.2 surveys representative existing efforts along these four axes. Several widely used datasets [23], [36], [37] are derived from human-orchestrated red-team campaigns and consist only of post-incident audit logs; while causally coherent, they are static and cannot be replayed against live systems. Abstract attack modeling approaches [38], [39], [40] represent attacks at the TTP level but lack concrete execution semantics, making them unable to be instantiated as runnable attacks. Single-point testing tools such as Atomic Red Team [41], Metasploit [42], and LOLBAS [43] provide executable implementations of individual techniques but cannot chain them into multi-step campaigns; in isolation, a benign-looking action such as file compression carries very different semantics than the same action preceded by data staging and followed by exfiltration. Procedural attack orchestration tools such as Caldera [44], PurpleSharp [45], and Attack

Range [46] sequence multiple atomic actions but do not explicitly model the causal dependencies between them, producing chains that read as procedural rather than adversarial. Expert-crafted playbooks such as those from the MITRE Engenuity ATT&CK Evaluations [47] are fully actionable, coherent, and CTI-aligned, but their reliance on deep domain expertise limits scalability: only 17 attack chains across 15 adversary groups have been produced in eight years.

Table 2.2: Comparison of existing approaches to adversary emulation along the four desired properties.

Category	Examples	Actionable	Causally Coherent	Customizable	Scalable
Human-Orchestrated Campaigns	DARPA TC, OpTC	✗	✓	✗	✗
Abstract Attack Modeling	AttacKG	✗	✓	✗	✓
Single-point Testing Tools	Atomic Red Team, Metasploit	✓	✗	✓	✓
Procedural Orchestration Tools	PurpleSharp, Caldera	✓	✗	✗	✓
Human-Crafted Playbooks	MITRE Engenuity Evaluations	✓	✓	✗	✗

The persistence of this gap is explained by three core challenges. First, cyberattack knowledge is highly unstructured, spread across CTI reports, exploit code, and tool documentation, and substantial expertise is required to compose it into coherent attack chains; even state-of-the-art automated planners such as ChainReactor [48] cover only narrow slices of the technique space. Second, modeling the causal dependencies between attack steps requires a formal framework that the literature has yet to fully provide [48], [49], [50]. Third, tailoring attacks to specific environments and to specific CTI-described threat actors, while keeping the result executable, remains an open problem. Section 4.1 of this dissertation presents AURORA, which addresses these challenges by combining LLM-based attack action extraction with symbolic planning over causality-preserving action models.

2.3.2 Benchmark Datasets in Intrusion Detection Research

Empirical evaluation of provenance-based intrusion detection depends critically on the quality and representativeness of available benchmark datasets. Early provenance datasets produced by systems such as PASS [51], Hi-Fi Provenance [52], and SPADE [53] served as proofs of concept for capturing system-level events, but they were small-scale, not publicly released, and not designed for cross-paper comparison. The release of larger, publicly available, attack-annotated corpora subsequently established a more standardized evaluation landscape for PIDS research; that landscape, however, has remained narrow and aging.

Dominant benchmarks. A small number of public datasets account for the overwhelming majority of evaluations in the recent PIDS literature: the DARPA Transparent Computing (TC) Engagements [54], [55], DARPA Operationally Transparent Computing (OpTC) [56], and ATLAS [23] together with its extension ATLASv2 [57]. StreamSpot [37], an earlier and smaller corpus of 600 graph instances, is also occasionally used for evaluating graph-based anomaly detection. A recent review [11] of intrusion detection papers shows that the field’s benchmarking is dominated by a few non-scalable datasets and that detection systems are routinely evaluated against attack techniques several years older than the papers themselves.

Limitations. Several prior analyses [12], [58] have identified specific limitations of these dominant benchmarks that constrain the credibility of reported detection results. DARPA TC’s benign workloads were generated by scripts and are highly repetitive, oversimplifying normal system behavior and inflating the contrast between benign and malicious activity. OpTC offers more operationally realistic workloads but contains substantial background noise and undocumented distribution shifts between its training and evaluation segments. ATLAS and ATLASv2 focus on well-defined attack engagements, but the conspicuousness of attack behaviors varies considerably across scenarios, and the granularity of ground-truth labeling differs widely. At a more

general level, the underlying attack techniques in these datasets reflect adversary tradecraft from 2018–2021, and all three corpora contain experimental artifacts such as logging gaps, workload reconfiguration, and machine downtime, which introduce distributional discontinuities orthogonal to the attack signal.

Recent efforts to mitigate dataset limitations. A handful of recent works attempt to mitigate these issues without constructing new attack corpora. ProvSyn [59] synthesizes provenance graphs to augment existing training data, but it inherits the attack patterns of its source datasets and cannot introduce techniques that those datasets do not already contain. Other studies [60], [61] construct adversarial perturbations of existing traces to study evasion, but they operate on the same outdated underlying attack scenarios. To the best of our knowledge, no public dataset captures execution traces of recently observed APT techniques at the scale and labeling granularity required for credible modern PIDS evaluation. Section 4.2 of this dissertation presents a dataset generation pipeline that addresses this gap.

CHAPTER 3

OPERATIONAL GAPS IN PROVENANCE-BASED INTRUSION DETECTION

3.1 Lightweight and Adaptive Detection via Gradient-Guided Rules

3.1.1 Motivation and Threat Model

Advanced Persistent Threats (APT) are becoming a growing threat to both government and industrial sectors, causing significant societal impacts [62]. The Equifax breach in 2017 resulted in the theft of vast amounts of personal data, highlighting the severe privacy and security risks [63], [64]. Moreover, attackers continuously innovate to find new ways to penetrate systems and remain undetected for extended periods. In recent years, provenance-based intrusion detection systems (PIDS) have gained attention from both the security industry and academia for their causality analysis capability. However, prevailing alarm fatigue, excessive runtime overhead, long detection latency, and opaque detection processes (based on black-box) are still open research problems in real-world scenarios [6], [29], [65], [66], [67].

Alarm fatigue [68], [69] is a significant issue plaguing the security industry. A recent survey [70] indicates that security analysts are required to handle an overwhelming average of 5,000 alarms daily, with a majority being false alarms. The excessive volume of alarms can lead to serious consequences. For instance, in the 2013 Target data breach [63], the malicious activities were detected and reported by security tools but overlooked by analysts, resulting in delayed response and expanded losses [71]. In practice, false alarms are equally detrimental as missed alarms. Moreover, significant challenges also arise from runtime overhead and detection latency. High runtime overhead compromises the scalability of detection systems, hindering their deployment on a large

scale [65]. Moreover, detection latency critically affects performance, as prolonged latency delays the response to threats, thereby impeding analysts’ efficiency in managing alarms.

In academia, a recent trend of PIDS [13], [14], [15], [25] leverage embedding techniques like word2vec [13] and graph2vec [25] to encode system entities and events, and neural networks like Graph Neural Networks (GNNs) [13], [14], [15], [24] and Recurrent Convolutional Neural Networks (RCNNs) [25] to analyze information flow and produce detection results. We refer to them as *embedding-based PIDS*. Although these systems have demonstrated notable achievements in detection performance, they face the following challenges:

- **High Computational Resource Cost.** Embedding-based systems need to embed the graph features into vectors, usually with deep learning techniques [13], [14], [15], [25], which require a significant amount of computational resources. Moreover, many systems require caching of embeddings and deep learning models, leading to high memory consumption [65];
- **Long Detection Latency.** Embedding-based systems take the input structured as graphs or paths, requiring time windows [14], [20] or log batches [13], [25] in their design, which leads to the additional detection delay.
- **Uninterpretable Results.** Many systems [13], [14], [15], [20], [21], [72] only flag deviations from normal behaviors without attack semantics. Moreover, we are unable to open the “black box” of the deep learning models to better understand the detection process.

On the other hand, a group of PIDS [16], [17], [27], [28], [29] operates based on human-defined mechanisms to generate representations for system entities, propagate information flow, and trigger alarms. We refer to them as *rule-based PIDS*. Leveraging these rules, they take actionable leads, allowing themselves to provide semantic-rich alarms and pinpoint specific suspicious events for further investigation. Besides, the rule-based approach operates on simple arithmetic

calculations, which require far fewer computational resources than complex matrix calculations by neural networks. Furthermore, some rule-based approaches process streaming data once it arrives, significantly reducing storage requirements and enabling rapid response with minimal delay. In summary, rule-based PIDS shows advantages in real-world detection tasks due to its fine-grained detection, semantic-rich alarms, lightweight overhead, and minimal detection delay.

Despite rule-based PIDS excelling in the aforementioned perspectives, they also face challenges in real-world deployment. These systems often employ simplistic and universal rules to distinguish between graphs (such as discrete trustworthiness levels of node in [16], [17], [28], universal system parameters in [16], [17], [28], [29]). The inflexible rules lead these PIDS to be either too lenient or too strict. In security operations centers (SOCs), analysts have to manually configure the models, which is a time-consuming process [73]. This highlights the need for an automated configuration methodology and scheme. To achieve this, a rule-based PIDS must be capable of adjusting its rules autonomously based on detection feedback from the training set, without human intervention. This requires an automatic feedback mechanism that establishes a direct connection between the rules and detection results, allowing the system to dynamically adjust its configuration based on the detection outcomes.

In this paper, we propose CAPTAIN, a rule-based PIDS with adaptive configuration learning capability. We aim to leverage the advantages of traditional rule-based PIDS while enabling the system to acquire suitable configurations autonomously. Specifically, we introduce three adaptive parameters to the rule-based PIDS, giving it more flexibility during detection. Moreover, we design a learning module to adjust these parameters automatically based on the detection results during training. We calculate and record the gradients of each adaptive parameter, transferring the rule-based system to a differentiable function. With the help of a loss function, the gradient descent algorithm can be utilized to optimize the adaptive parameters based on the benign training data and

thus reduce the false alarms in testing.

We evaluate CAPTAIN in diverse detection scenarios, which are drawn from the widely acknowledged public datasets [74] provided by DARPA Engagement [54] and datasets collected from simulated environments in collaboration with a SOC. The evaluation results demonstrate that CAPTAIN can reduce false alarms by over 90% (11.49x) on average compared to traditional rule-based PIDS. Additionally, CAPTAIN achieves much better detection accuracy with less than 10% CPU usage and significantly lower memory usage and latency compared to the SOTA embedding-based PIDS. We then select a few scenarios as cases to study the explainability of the learned configurations.

In summary, this paper makes the following contributions:

- We propose CAPTAIN, a rule-based PIDS that can adjust its rules using benign data to reduce false alarms. Our design endows CAPTAIN with the lightweight nature, low latency, and interpretability of rule-based PIDS while also having fine detection capability and adaptability.
- We parameterize the rule-based PIDS and propose a novel differentiable tag propagation framework that allows us to optimize adaptive parameters using the gradient descent algorithm.
- We systematically evaluate CAPTAIN across various scenarios, including widely used DARPA datasets and simulated datasets from a partner SOC. The evaluation result demonstrates that CAPTAIN can automatically adapt to various environments, offering superior detection accuracy with reduced latency, lower overhead, and more interpretable detection.

We make the code of our system, all datasets, and all experiments publicly available for future analysis and research¹.

¹<https://github.com/LexusWang/CAPTAIN>

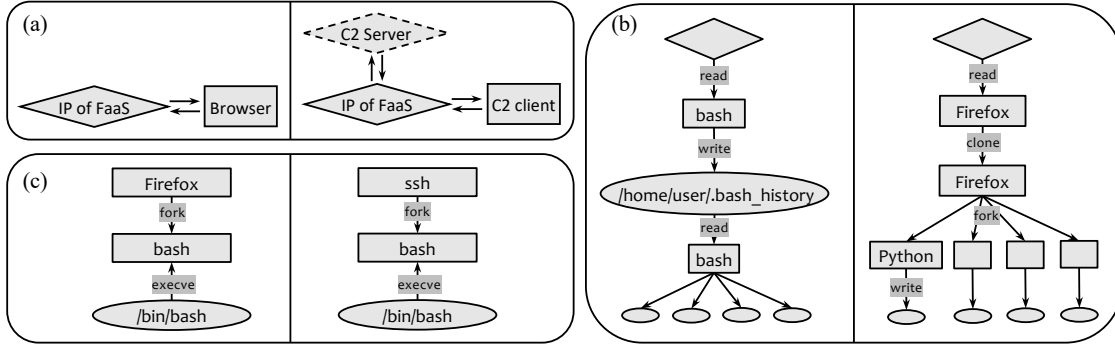


Figure 3.1: Three motivating examples from the real-world dataset where more fine-grained rules are needed in the rule-based PIDS.

3.1.2 Motivation Examples

In this section, we present motivating examples to elucidate the challenges encountered by traditional, coarse-grained rule-based PIDS, as depicted in Fig. 3.1. These scenarios contain similar benign and malicious behaviors, which pose challenges to existing rule-based PIDS [16], [17], [28], [29] in adjusting their rules to effectively select an optimal trade-off point between excessive false alarms and missing true alarms.

“Grey” Nodes: With the advancement of cloud computing, connections to Function as a Service (FaaS) platforms like AWS Lambda and Cloudflare Workers have become a typical pattern in provenance graphs. However, recent reports indicate that attackers are exploiting these commercial FaaS services to redirect traffic to Command and Control (C2) servers, as illustrated in Fig. 3.1a. Existing rule-based PIDS are often too coarse to depict those IPs. For example, SLEUTH [16] only defined three trustworthiness levels: *Benign Authentic*, *Benign*, and *Unknown* for the nodes, and MORSE [17] used a binary initial tag for the trustworthiness. But in this scenario, since both benign and malicious behavior can involve communications with the external FaaS IPs, the coarse-grained rules either lead to excessive false alarms or missing true alarms. A more fine-grained configuration to reflect the “grey scale” of these IPs may help solve the issue, i.e., we can assign a trustworthiness

value (a float between 0 and 1) to every node in the provenance graph.

Dependency Explosion: Dependency explosion refers to the scenario where all the subsequent events are treated as they depend on the previous ones, and thus a single suspicious event can lead to millions of system entities considered suspicious [17]. As shown in Fig. 3.1b, a prolonged and furcate event chain can spread the maliciousness carried by the suspicious entities to entities involved in all subsequent events. Many existing efforts have been made to solve this issue [75], [76], [77], [78], [79]. Unfortunately, their solutions usually require extensive instrumentation of applications/OS [77], [78], [79] and expertise, which limits the usage in the real-world system, especially in end-point host deployment.

To deal with the dependency explosion problem without extensive instrumentation, existing rule-based PIDS adopt methods like event whitelisting [16], cost-based pruning, or dividing suspicious events into different stages in a typical APT lifecycle [28]. MORSE [17] proposed the decay and attenuation mechanisms to reduce the maliciousness carried by the system entities over time to mitigate the dependency explosion. However, the decay and attenuation factor is the same for all nodes and edges. Such identical configuration can lead to the dilemma between excessive false alarms and missing true alarms as attackers may deliberately fork many irrelevant processes or connect to benign entities to evade detection. As shown by the two provenance graphs in Fig. 3.1b, we want to confine the dependency explosion in the left graph while preserving the maliciousness in the right (the right graph depicts a rare pattern where a `firefox` process forks a `Python` process, indicating the possibility of suspicious script execution from a website.) Ideally, we aim to set a propagation rate for each individual event, allowing for more customized control. This ensures that the maliciousness of unusual events does not fade away too quickly, while it can reset rapidly for more common events.

Customized Alarm Triggering: In rule-based PIDS, alarms are triggered based on predefined

conditions that are uniformly applied to all events, regardless of their context. These one-fits-all approaches can lead to suboptimal detection performance. For instance, MORSE sets a series of detection rules and thresholds to trigger alarms and the rules are identical for all processes and events. As the motivating example shows in Fig. 3.1c, `firefox` and `sshd` are two processes that often have network activities with various IP addresses. While `sshd` may commonly use `bash` for script execution, it is unusual and suspicious for `firefox` to do the same. Despite the similarities in event types and related entities between `firefox` and `sshd`, applying a uniform threshold for all “fork” events could either result in false alarms for `sshd` or miss genuine threats involving `firefox`. To distinguish between such two behaviors, a specific rule is needed: for the common events, we want to set a higher threshold to avoid excessive false alarms, while for rare events, we want to keep the threshold low to avoid missing true alarms. A knottier question then follows: how to set different thresholds about the “execute bash” event for each process?

The three motivating examples underscore the challenges that existing rule-based PIDS face in balancing false and true alarms due to their coarse-grained rules. Additionally, these systems lack the flexibility to dynamically adjust rules to accommodate different environments. Theoretically, embedding-based PIDS, leveraging the power of deep learning, can effectively create customized “rules” for every event and entity in the graph by embedding them into the feature space. However, training and deploying deep neural networks require numerous calculation resources and storage space, leading to high runtime overhead. Moreover, these PIDS usually lack the explainability of the detection process and result, making it difficult to pinpoint the malicious behaviors at the event level.

To overcome these limitations, we aim to refine rule-based PIDS using fine-grained rules. By allowing the rule-based detector to learn and adapt from training data, we can enhance detection accuracy without the extensive resource demands and runtime overhead. Our system improves the

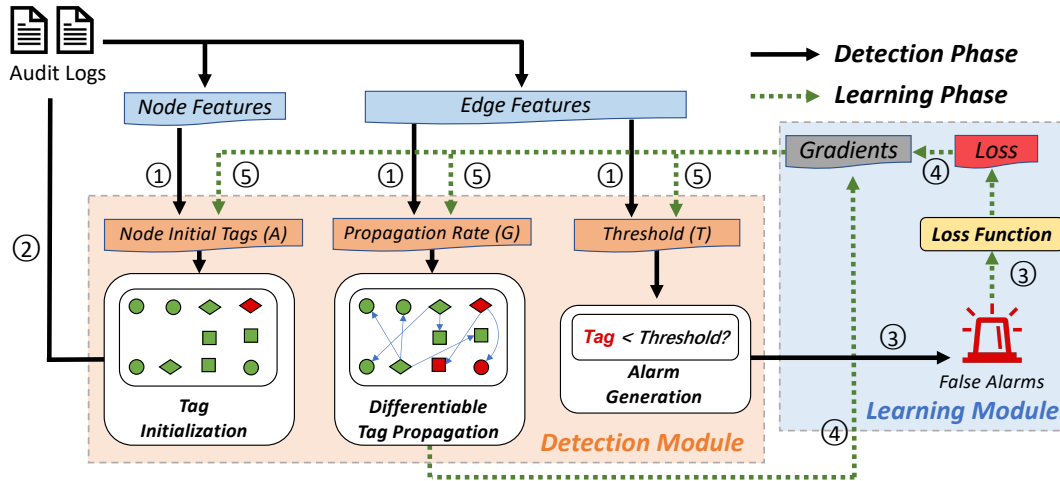


Figure 3.2: The overall framework of CAPTAIN. Phases ①-⑤ show the lifecycle of the detection module and the learning module within one training epoch.

overall efficacy of rule-based PIDS but also maintains its operational simplicity and clarity.

3.1.3 Design

3.1.3.1 System Overview

As shown in Fig. 3.2, CAPTAIN consists of two major parts: the detection module and the learning module. The detection module takes the audit logs as input and produces detection results, while the learning module leverages the false alarms during the training phase to fine-tune the detection module.

3.1.3.2 Adaptive Detection Module

The detection module in CAPTAIN comprises three major components: tag initialization, tag propagation, and alarm generation. System entities are assigned initial tags when they appear in the system. As system events happen, tags get updated among system entities, which we refer to as “tag propagation”. At the same time, CAPTAIN determines whether to generate alarms based on

the pre-defined rules.

Tag Design: Inspired by previous provenance-based methodology based on data flow and control flow [16], [17], [80], [81], CAPTAIN designs two types of tags: data tags and code tags. As shown in Table 3.1, data tags exist on all nodes in the provenance graph. A data tag is represented as a numerical vector $\langle c, i \rangle$, where c denotes the confidentiality score of the data, and i denotes the integrity score of the data. In contrast, code tags only exist on process nodes, denoted by $\langle p \rangle$ to indicate the integrity of the code. All scores (c , i , and p) are real numbers ranging from 0 to 1. A low integrity score suggests that the data/code might be compromised and, therefore, is considered untrusted. A low confidentiality score is counterintuitively associated with highly sensitive data (to make the scoring scheme consistent), such as passwords.

Table 3.1: Tags in CAPTAIN

Tag Type		Value Range
Data Tag	Confidentiality (c)	0 (Most Confidential) to 1 (Public)
	Integrity (i)	0 (Lowest) to 1 (Highest)
Code Tag	Integrity (p)	0 (Lowest) to 1 (Highest)

The tag system used by CAPTAIN is inspired by previous rule-based PIDS [16], [17]. The focus of this paper is not on the design of the tag system. In essence, any new type of tag can be integrated with CAPTAIN as long as it satisfies the following conditions: 1) The tags are numerical; 2) The tags are updated through arithmetic calculation; 3) Alarms are triggered based on the tag values. For example, we can define an additional “exploitability” tag for all web service nodes and propose the corresponding propagation rules and alarm-triggering rules. We leave tag designing as a practical problem in specific scenarios and use the tags mentioned above to show the improvement purely brought by CAPTAIN’s learning module.

We then introduce three adaptive parameters, which are the core of the fine-grained rule-based

PIDS:

- **Tag Initialization Parameter** (A) determines the initial tags of system entities.
- **Tag Propagation Rate Parameter** (G) adjusts the propagation effect of system events on tags.
- **Alarm Generation Threshold Parameter** (T) makes adjustments to the alarm generation rules.

We will elucidate how CAPTAIN uses the three parameters to achieve more flexible rule-based detection.

Tag Initialization: Proper tag initialization rules play a crucial role since they determine the initial states of system entities. For instance, if a socket associated with a benign IP address is assigned low integrity, it could lead to many false alarms. However, it is relatively less discussed in previous works [16], [17]. They usually assign the initial tags based on domain knowledge without further adjustments. Although MORSE admits tag initialization policies could be learned from the previous tracing data, it did not discuss it in detail or propose a practical methodology.

We initialize the tags on the nodes when they are initially added to the provenance graph. As illustrated in Fig. 3.2 step ①, the detection system first checks for new system entities in the event upon receiving the system audit logs. If new entities are identified, they are added to the provenance graph and assigned initial data tags. Please note that when adding a new process node, we do not specifically initialize its code tag and data tag. A process node's code tag value is inherited from the data tag of the files loaded by the process, and its data tag is propagated from its parents. For non-process nodes, we define the adaptive parameter A at the node level to facilitate fine-grained tag initialization rules. Specifically, for any node $n_i \in N$ in the provenance graph, A stores its initial data tag with respect to its node feature. The node features depend on the information granularity of audit logs, such as file path, process name, command line, IP address, port, etc. For example, if the node feature for a socket is its IP address, then all sockets with the same IP address will be

assigned the same initial data tag.

CAPTAIN’s learning module refines A through the training process, which will be elaborated upon in §3.1.4. The training process starts from the default value A_0 . Since we train CAPTAIN on the benign data, we set A_0 conservatively. In this paper, “conservative” means the assumption of the worst-case. Under conservative settings, we aim to capture true attacks as much as possible without considering excessive false alarms. Rule-based PIDS such as MORSE has to use the conservative setting to capture all potential attacks [17], while CAPTAIN only uses the conservative setting as the initial state for training on benign data. For example, we assign all IP addresses an initial integrity of 0 so we do not miss any attackers entering through network communications.

Although the confidentiality scores could be customized in CAPTAIN, we find it hard to learn them from the false alarms on the benign training data. Instead, it often requires specific domain knowledge or personal, subjective judgments to determine what files are sensitive. Therefore, we manually set the initial confidentiality scores and do not adjust them with the learning module.

Tag Propagation: As shown in Fig. 3.2 step ②, once the tags are initialized, specific system events will trigger the propagation of these tags along the direction of the information flow, resulting in changes to code and data tags. In CAPTAIN, tag propagation will pass and accumulate malicious intentions through tag values. For example, the `(Firefox, read, IP)` event updates the data tag of `Firefox` to the lower value between the data tags of `Firefox` and `IP`. This ensures that potential malicious intentions carried in the information flow are preserved. We designed the propagation rules based on existing work [16], [17] as detailed in Table A.2.

Unfortunately, such a tag propagation mechanism can suffer from the dependence explosion issue [17], leading to excessive alarms. For example, as shown in Fig. 3.1, `(bash, write, /home/user/.bash_history)` is a common event that appears in many propagation chains of false alarms, which means this event causes large-scale maliciousness propagation on the prove-

nance graph. However, our investigation shows this is a commonly seen benign activity.

To mitigate such a problem, we introduce another adaptive parameter G , to regulate the propagation rate. Specifically, for any edge $e \in E$ identified with $(src_node_feature, event_type, dest_node_feature)$ in the provenance graph, G stores its propagation rate parameter $g_e \in [0, 1]$. Given g_e , the source node src , and the destination node $dest$, we update tag_{dest} as follows during tag propagation.

$$tag_{dest}^{new} = g_e \cdot tag_{rule} + (1 - g_e) \cdot tag_{dest} \quad (3.1)$$

Where tag_{rule} is the tag value given by the propagation rules defined in Table A.2. Usually, it equals to tag_{src} . If g_e is close to 0, the corresponding tag propagation will lead to small changes in the tag values and vice versa. In this way, G allows CAPTAIN to fine-tune the propagation rules at the edge level.

CAPTAIN refines G through the training process in the learning module, which will be elaborated upon in §3.1.4. We also conservatively establish the default values of G (G_0) to ensure all true alarms can be captured before the training starts. Particularly, we set $g_e = 1$ for all edges $e \in E$ to make all propagation fully effective.

Alarm Generation: The alarm generation rules determine whether an alarm should be triggered. As shown in Fig. 3.2 step ③, whenever an event happens, we will assess whether it satisfies the criteria to trigger alarms. For example, if a process with a data integrity value smaller than 0.5 writes to a normal file, a file corruption alarm will be triggered. As shown in Table A.3, we generate alarms based on the event type, the subject and object tags, and the threshold. The threshold could be fine-tuned to control the number of alarms. For example, if event `(sshd, execve, bash)` triggers many false alarms during training, it suggests that we should decrease the alarm-triggering threshold for this event.

We introduce an adaptive parameter T to enable fine-grained adjustment of the alarm threshold.

Specifically, for any edge $e \in E$ identified with $(src_node_feature, event_type, dest_node_feature)$, T stores its alarm threshold $thr_e \in [0, 1]$. For an edge e , the detection function $f(e)$ is defined as follows.

$$f(e) = tag - thr_e = \begin{cases} malicious, & \text{if } f(e) < 0 \\ benign, & \text{otherwise} \end{cases} \quad (3.2)$$

where tag is the relevant tag on the node of interest on edge e_i . A low threshold prevents the system from generating the alarms, while a high threshold encourages the system to generate the alarms. In this way, T allows CAPTAIN to adjust the alarm-triggering rules at the edge level.

Like A and G , CAPTAIN's learning module refines T can be refined through the training process (§3.1.4). Before the training, we set the default thresholds (T_0) neutrally. In other words, we set $thr_i = 0.5$ for every edge e_i so that we do not encourage or suppress all alarms.

Please note that CAPTAIN's methodology is independent of specific rules or tags, making it broadly applicable to various rule-based PIDS, especially those using taint-analysis methods like SLEUTH [16], HOLMES [28], and CONAN [27].

3.1.4 Learning Module

With the three adaptive parameters (A , G , and T) we defined in the previous section, the problem of adaptative configuration for the detector is then converted to the optimization problem of the multi-variable function aiming to find the optimal values for A , G , and T . In this section, we present our efforts to solve this optimization problem with the gradient descent algorithm, including defining the objective function and constraints, calculating gradients, and searching for the optimal values.

3.1.4.1 Loss Function

The first step of solving the optimization problem is to define the objective function, i.e., the loss function in the context of learning tasks. The learning module aims to find the parameters that can reduce false alarms while maintaining the sensitivity to the true malicious events. Therefore, the loss function comprises two terms: the term of false alarms and the regularizer term.

The false alarm term mainly focuses on penalizing erroneously triggered alarms. Specifically, an event e triggers a false alarm means $f(e)$ should be greater than 0 but it does not. Since $f(e) \in [-1, 1]$, we can use the Mean Squared Error $(y_e - f(e))^2$ as the loss function for all events that trigger a false alarm. y_e is set as 1 for those benign events, and -1 for the malicious events. Because we only use benign data for training, all y_e should be 1. Please also note that we do not calculate the loss for the correctly classified events, i.e., we do not make $f(e)$ close to 1 if $f(e)$ is already greater than 0. This is because the events that do not cause alarms are significantly more than the alarm-triggering events, considering them, therefore, is inefficient and would make the detection system insensitive to the malicious events. Thus, the false alarm term in the loss function can be formalized as

$$\mathcal{L}(e) = \max(0, (1 - f(e))^2 - 1) \quad (3.3)$$

Next, we introduce the second component: the regularizer term. As we mentioned, CAPTAIN is trained on benign data to learn the normal behaviors in the detection environment. This is due to the fact that benign behaviors on a system are more consistent than attack activities. Another reason is the malicious training data is much harder to acquire. However, a big challenge is how to guarantee that the detection capability would not be affected if there is no malicious sample in the dataset, i.e., we would not be too lenient to capture the true alarm when reducing false alarms.

To capture all potential malicious events, the adaptive parameters are configured conservatively

before training. During training, we only want to make small adjustments to a small portion of the parameters according to the false alarms on training data. For the rest parameters, we would like to keep them as conservative as possible so that the sensitivity to the maliciousness is not compromised. In the area of machine learning, one-class classification (OCC) algorithms are proposed to deal with the situation where only one class of samples is available in the training set [82]. Inspired by [83], [84], we add a regularizer term in the loss function to avoid the parameters becoming too lenient when the malicious training sample is absent. The essential thought is to make the adaptive parameters as close to the default values (A_0, G_0, T_0) as possible. By minimizing the l_2 distance between the adaptive parameters and the default values, we protect the detection capability during training. Therefore, the detection loss can be formalized as

$$Loss = \sum_{e \in E} \mathcal{L}(e) + \alpha \|A - A_0\|_2 + \gamma \|G - G_0\|_2 + \tau \|T - T_0\|_2 \quad (3.4)$$

where, α , γ , and τ are the regularizer coefficients. In the evaluation section, we discuss the effect of the regularizer coefficients (§3.1.9.2) and methods used to fine-tune them (§3.1.6.2).

3.1.4.2 Differentiable Detection Framework

One of the fundamental steps of using gradient descent algorithms is calculating the gradient of each variable. To adjust the adaptive parameters based on the loss function, we need to design a differentiable detection framework, which allows us to keep the gradients of the adaptive parameters with respect to $Loss$.

According to the chain rule, for the parameter a_n , we have

$$\frac{\partial Loss}{\partial a_n} = \sum_{e \in E} \frac{\partial \mathcal{L}(e)}{\partial f} \cdot \frac{\partial f}{\partial a_n} + \alpha \cdot (a_n - a_0) \quad (3.5)$$

The equations for g_e and thr_e (presented in Appendix §A.3) hold similarly to Eq. 3.5. The most challenging part of building an adaptative rule-based PIDS is to compute the gradients, i.e., $\partial f / \partial a_n$, $\partial f / \partial g_e$, and $\partial f / \partial thr_e$. This is because the rule-based detection process is usually modeled as the branch selection based on rules, rather than a differentiable function. To the best of our knowledge, no previous work has formalized or recorded the gradients of parameters in a rule-based PIDS. In CAPTAIN, we model the detection process as a differentiable function and calculate the gradients of each adaptive parameter. When the tags are propagated, the corresponding gradients are updated and recorded. In so doing, we associate the adaptive parameters with the loss, making it possible to perform the gradient descent algorithm to find the optimal parameters that minimize the loss.

As per Eq. 3.2, $\partial f / \partial thr_e = -1$. This means every time when we want to increase the value of f (the direction of benign), we have to decrease the value of thr_e , and vice versa. This aligns with our intuition because a lower tag value means more maliciousness in our system. Therefore, if we want to be more lenient in the detection, we should set a lower threshold.

The formalization and calculation of $\partial f / \partial a_n$ and $\partial f / \partial g_e$ are more complicated. According to Eq. 3.2, calculating the gradients of the detection function f is essentially computing the gradients with respect to the variable tag for each node.

We start from the gradients of a_n . According to the propagation policies shown in Table A.2, the updated tag value tag_{rule} is either the lower tag values of involved nodes $\min(tag_{src}, tag_{dest})$ or a constant value c . Note the subscripts src and $dest$ denote the propagation direction. If $tag_{rule} = c$, then

$$\frac{\partial tag_{dest}^{new}}{\partial a_n} = (1 - g_e) \frac{\partial tag_{dest}}{\partial a_n} \quad (3.6)$$

If $tag_{rule} = tag_{src}$, then

$$\frac{\partial tag_{dest}^{new}}{\partial a_n} = g_e \frac{\partial tag_{src}}{\partial a_n} + (1 - g_e) \frac{\partial tag_{dest}}{\partial a_n} \quad (3.7)$$

Since a_n is defined as the initial value of the tag on node n , the initial gradient of a_n is

$$\frac{\partial tag_{n'}}{\partial a_n} = \begin{cases} 1, & \text{if } n' = n \\ 0, & \text{if } n' \neq n \end{cases} \quad (3.8)$$

Then we focus on calculating gradients of g_e , which is more complicated. From Eq. 3.1, we know that tag_{dest}^{new} is determined by tag_{dest} , tag_{rule} , and g_e . And tag_{dest} , tag_{rule} are related to the propagation rates of previous events. Therefore, tag_{dest}^{new} is influenced by the propagation rates of the current event (denoted by g_e) and all previous events (denoted by $g_{e'}$) happened on src and $dest$ nodes. Consequently, for each propagation, we must update $\partial tag_{dest}^{new} / \partial g_e$ and $\partial tag_{dest}^{new} / \partial g_{e'}$ simultaneously. The case of $g_{e'}$ is similar to that of a_n , we have

$$\frac{\partial tag_{dest}^{new}}{\partial g_{e'}} = g_e \frac{\partial tag_{rule}}{\partial g_{e'}} + (1 - g_e) \frac{\partial tag_{dest}}{\partial g_{e'}} \quad (3.9)$$

Calculating gradients of g_e involves the Product Rule in calculus, which is

$$\frac{\partial tag_{dest}^{new}}{\partial g_e} = g_e \frac{\partial tag_{rule}}{\partial g_e} + (1 - g_e) \frac{\partial tag_{dest}}{\partial g_e} + tag_{rule} - tag_{dest} \quad (3.10)$$

The full proof of Eq. 3.9 and Eq. 3.10 can be found in the Appendix §A.3.

As previously stated, g_e are set according to the event features ($src_node_feature$, $event_type$,

dest_node_feature). Therefore, before the propagation happens,

$$\frac{\partial tag_n}{\partial g_e} = 0, \forall e \in E, \forall n \in N \quad (3.11)$$

We use Eq. 3.11 to initialize the gradient of g_e and update them according to Eq. 3.9 and 3.10.

In summary, to employ the gradient descent algorithm, we calculate the gradients of the adaptive parameters with respect to the loss. We first calculate the initial gradient for each node. Afterward, the gradients are updated according to our equations when the tags are propagated. Please note that unlike GNN models such as GCN [85] and GraphSage [86], which only consider the n -hop neighbor of a node, CAPTAIN fine-tunes every edge and node in the graph that can influence the detection results by propagating their gradients.

3.1.4.3 Training and Testing

After calculating the gradients of the adaptive parameters with respect to the loss using the differentiable detection framework, we can now utilize the gradient descent algorithm to optimize the parameters. In machine learning, this process is referred to as “training”. The learned parameters are stored as a customized configuration, which is then used to set up CAPTAIN prior to testing.

As shown in Fig. 3.2, an epoch in training comprises the forward and backward propagation. Before training starts, all adaptive parameters, A , G , and T , are configured as the default settings A_0 , G_0 , and T_0 . Instead of random initialization[87], we use the most conservative setting as the starting point to keep the sensitivity to the maliciousness during training.

When CAPTAIN processes the events, tags are propagated among the graph, updating the gradients according to Eq. 3.7, 3.9 and 3.10, and generate the detection results. Next, we calculate the loss and back-propagate the gradients of loss according to Eq. 3.5. Finally, parameters are updated

in the opposite direction of the gradient as follows:

$$p_{new} = p_{old} - l \cdot \frac{\partial Loss}{\partial p_{old}} \quad (3.12)$$

where p is the adaptive parameters (which could be a_n , g_e , or thr_e), p_{new} is the new parameters, p_{old} is the old one, $\frac{\partial Loss}{\partial p_{old}}$ is the gradient, and l is the learning rate. We repeat the training process when the maximum epoch is reached or the changes in the result become sufficiently small. Before testing, we configure the parameters according to what we learned in the training stage. Then, CAPTAIN can process audit data and conduct detection as introduced in § 3.1.3.1.

3.1.5 Evaluation

The entire system (including data parsing, tag initializing, tag propagating, alarm generating, and training/testing framework) consists of 5KLoC of Python. We implement the differentiable detection framework by creating two dictionaries for each tag to store the gradients with respect to A and G . As for T , because its gradients are unrelated to the node tags, there is no need to store them within the node. The space used to store those gradient dictionaries during training is analyzed in § 3.1.8.2 and § A.5.1.

Our evaluation aims to answer the following five research questions: 1) How effectively can CAPTAIN detect the attacks, especially in terms of reducing false alarms? 2) How efficient is CAPTAIN compared with the SOTA PIDS in terms of detection latency and runtime overhead? (§3.1.8) 3) How robust is CAPTAIN against adversarial attacks such as mimicry attacks and data poisoning attacks? (§3.1.9) 4) How do different components affect the training outcome and the detection performance of CAPTAIN? (§3.1.10) 5) Can CAPTAIN acquire explainable knowledge via our learning module? (§3.1.11)

3.1.6 Experiment Settings

3.1.6.1 Datasets

We evaluate CAPTAIN using public forensic datasets from the DARPA Transparent Computing program and datasets generated within simulated environments in collaboration with an SOC.

DARPA Datasets. The DARPA Transparent Computing program was organized between 2016 and 2019 to perform several red team assessments. In two weeks, the data collecting teams deploy collectors on several target hosts [54]. We use the public-available datasets from Engagement 3 (E3) and Engagement 5 (E5) in our evaluation. [88], [89] provides a detailed description of attacks performed in relevant DARPA datasets.

Simulated Environments. Moreover, we collaborate with an industry SOC to acquire additional datasets within realistically simulated scenarios to avoid the problem of “close-world data” [20]. Specifically, the SOC furnishes detailed host setups from real-world operating environments. Subsequently, we simulate APT attacks [90] employing the Atomic Red Team [91] and online malware repositories [92]. The simulated scenarios encompass five APT attacks across three distinct real-world operational settings, elaborated in detail in §A.1.

Data Labeling. For each attack scenario, we label entities and events on the kill chains as malicious according to the attack reports. Although previous work [13], [72] provided entity-level data labels, they did not specify the data labeling strategy or context. We also observed that the amount of malicious labels in their dataset is excessively large (e.g., over 12 thousand system entities were marked as malicious within a 30-hour period on the CADETS from DARPA Engagement 3), which is impractical for a real-world SOC. Thus, we decided to use our data labels in the evaluation for a fair and unbiased comparison between CAPTAIN and the baseline systems. We also make our data labels publicly available to facilitate future research.

3.1.6.2 Experiment Setup

We deployed CAPTAIN and performed all experiments on an Ubuntu 22.04.3 Linux Server with an Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz and 1.0 TB memory. We partition each dataset into training and testing sets, adhering to the assumption that the training set should not contain any malicious activities. Specifically, we find the starting time of the first attack. Then, all data produced before that date becomes the training set, whereas the remaining data becomes the testing set.

To avoid biased results due to overfitting, we performed cross-validation when evaluating the performance of CAPTAIN. However, standard k -fold cross-validation is unsuitable for streaming logs because the time series cannot be freely split into k groups, as the subsequent tags depend on the previous ones. Therefore, we utilized time series cross-validation, which preserves the chronological order of data. We set a time window and trained the parameters using the data within this window. Afterward, we moved the time window forward until creating k different training sets. In our experiment, we set $k=3$ and the length of the time windows to be around $2/3$ of the total length of the training set. We also discuss overfitting in § 3.1.12.1.

Like many other machine learning systems, CAPTAIN also relies on appropriate hyperparameters, especially three regularizer coefficients α, γ, τ , to train the accurate and robust model (the effect of these hyperparameters are shown in §3.1.9.2). We provide two methods to fine-tune the hyperparameters, depending on whether a validation set is available. If a validation set exists, we perform grid searching, a generic hyperparameter tuning approach. We train multiple models with different hyperparameter combinations and evaluate their performance on the validation set. The default conservative detector is run on the validation set as the baseline. We select the hyperparameters from the model with the fewest false alarms and equivalent true alarms to the baseline. For instance, the optimal hyperparameters of E3-CADETS is $\alpha = 0.1, \gamma = 0.1, \tau = 0.1$ and the

learning rate is 0.01. However, a validation set is usually not available for a PIDS in practice. Even if it exists, it may not encompass all possible attacks. In this case, we propose a heuristic-based method to fine-tune the regularizer coefficients. Recall that the training starts from the conservative adaptive parameters, false alarms in the training set will loosen the conservative settings. The regularizer coefficients are added to the loss function to avoid being too lenient and missing the true alarms. Therefore, the values of the regularizer coefficients α , γ , and τ are determined by the following question: how many false alarms caused by a node/edge can we tolerate in the training set to avoid being overly lenient? Before training starts, we set a number for the allowed false alarms (N), and we can estimate an approximate value of the regularizer coefficients based on N : $\alpha \approx 3N$, $\gamma \approx 3N$, $\tau \approx 12N$ (see Appendix §A.3 for the mathematical proof). Please note that these are approximate estimations depending on the extent to which we trust the “benign” training data. A practical way might be: set regularizer coefficients using heuristics; if missing alarms occur compared to the conservative baseline, multiply the coefficients by a factor (e.g., 10) and retrain. Conversely, if no alarms are missed for a long time T , reduce the coefficients by a factor and retrain.

3.1.6.3 Baseline Detectors

We compare CAPTAIN with five SOTA PIDS: FLASH [13], KAIROS [14], SHADEWATCHER [15], NODLINK [20], and MORSE [17] to evaluate their performance from different perspectives. We chose FLASH, KAIROS, and NODLINK because they are the SOTA embedding-based PIDS. In addition, their implementations are open-sourced, allowing us to test them on different datasets. Since some other embedding-based PIDS [21], [28], [72] have already been evaluated in these works and the result shows that FLASH, KAIROS, and NODLINK outperform them, we didn’t include them in our evaluation. We chose MORSE since it is the SOTA rule-based PIDS, and we

want to evaluate the improvements from our differentiable adaptation framework in CAPTAIN. To get a fair and unbiased comparison, we use the settings from MORSE in the evaluation. According to [13], [14] and our communication with the authors, the detection systems of SHADEWATCHER and PROGRAPHER are not fully open-source due to proprietary license restrictions. Although the data preprocessing module of SHADEWATCHER is open-sourced, we tried our best but could not locate the preprocessing output files used for the following training and testing. We then realized that it was not feasible for us to perfectly replicate their systems for an unbiased comparison. Therefore, we used the code to evaluate the efficiency and latency of data preprocessing and the detection results reported in their paper to evaluate detection accuracy. For FLASH, KAIROS, and NODLINK, we used their open-sourced code as the basis for the evaluation. We reimplemented MORSE according to their paper.

3.1.7 Detection Accuracy

3.1.7.1 False Alarm Events Reduction

We first focus on the reduction of false alarm events, which requires the detection granularity at the event level. We compare CAPTAIN with the SOTA event-level PIDS SHADEWATCHER and the classical rule-based PIDS MORSE. Due to the close-source nature of SHADEWATCHER, we used the reported detection results in their paper. The results show that both MORSE and CAPTAIN successfully detected all attacks in the testing dataset. But CAPTAIN reduces the false alarm rate by over 93% (15.66x) compared to SHADEWATCHER and over 95% (20.45x) compared to MORSE. Additionally, SHADEWATCHER holds 80% of events for training and only 10% for testing, while the testing set of CAPTAIN is over seven times larger than the testing set of SHADEWATCHER. This shows that CAPTAIN does not require as much data as SHADEWATCHER for training.

One advantage of the rule-based PIDS like CAPTAIN and MORSE is that it can offer semantic-

Table 3.2: Comparison with the baselines on the TRACE dataset from Engagement 3 in terms of false alarm events

	MORSE	SHADEWATCHER	CAPTAIN
# of Consumed Events	5,188,230	724,236	5,188,230
# of False Alarm Events	22,500	2,405	1,099
False Alarm Rate	0.434%	0.332%	0.0212%

rich alarms, while SHADEWATCHER only flags deviations from patterns observed during training without explaining the alarms. We then compared the number of false alarms generated by CAPTAIN and MORSE into different categories, shown in Table 3.3.

Table 3.3: Comparison with the baseline on different datasets regarding false alarms. T-3, T-5, C-3, and C-5 represent the TRACE and CADETS datasets from Engagement 3 & 5. S-1 to S-3 denote Cloud, Streaming, and Dev datasets from the SOC.

Datasets	FileExe		MemExec		ChPerm		Corrupt		DataLeak		Escalate		Total False Alarms		
	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Reduction
T-3	9	0	49.3K	3.76K	1	1	626	2	955	14	673	0	52.1K	3.78K	13.78x
C-3	41	14	N/A	N/A	6	6	13.2K	272	96	50	81	1	13.4K	343	39.07x
T-5	181	170	403K	79.8K	0	0	34.4K	19.9K	26.9K	1.20K	18.4K	28	483K	101K	4.78x
C-5	1.63K	0	N/A	N/A	N/A	N/A	1.81M	6.64K	7.87K	2	1.42K	0	1.83M	6.64K	276x
S-1	3.16K	0	N/A	N/A	26	0	100	73	22	21	N/A	N/A	3.31K	94	35.21x
S-2	177	53	N/A	N/A	0	0	12	9	14	7	N/A	N/A	203	69	2.94x
S-3	18	0	N/A	N/A	0	0	29	16	23	22	N/A	N/A	60	38	1.58x

Table 3.3 shows that CAPTAIN can reduce false alarms by over 90% (11.49x) on average for all datasets compared with the non-adaptive MORSE. It outperforms MORSE in every alarm category. Those remaining false alarms, especially in C-5, S-2, and S-3, cannot be removed for the following reasons. First, some “false” alarms are not purely benign. They are related to some (potential) attack nodes (e.g., configuration files in `/tmp/atScript/atomic-red-team-Gray_dev1.0/*` in S-2) but cannot be mapped to specific attack steps directly, thus not labeled as “malicious.” Second, some nodes and edges do not exist or trigger any alarm in the training set. Third, since our approach aims to be as conservative as possible, some parameters are tuned just enough to eliminate false

alarms in the training set. However, these conservative settings may still cause false alarms when applied to the testing set.

Table 3.4: Comparison with the baselines in terms of node-level detection accuracy

	TP	FP(0-hop)	FP(1-hop)	FN(0-hop)
Engagement3 CADETS				
FLASH	16	4503	4485	10
KAIROS	15	1017	1003	11
NODLINK ¹	3	120	114	2
MORSE	16	51	43	10
CAPTAIN	16	34	26	10
Engagement3 TRACE				
FLASH	5	27202	27178	19
NODLINK ¹	4	170	170	0
MORSE	10	243	234	14
CAPTAIN	10	12	11	14
Engagement3 THEIA				
FLASH	2	53230	53050	13
KAIROS	12	3566	3422	3
NODLINK ¹	4	62	58	0
MORSE	11	220	213	4
CAPTAIN	11	194	187	4

¹Since NODLINK only provides detected process, we evaluate it on process detection accuracy.

3.1.7.2 False Alarm Entity Reduction

While CAPTAIN operates as an event-level detector, we still compare CAPTAIN with the SOTA entity-level PIDS FLASH, KAIROS, and NODLINK(KAIROS and NODLINK give the result at graph-level, but they also support entity-level detection). We transformed the alarm events triggered by MORSE and CAPTAIN into alarm entities using the following method: an entity is considered an alarm entity if it is involved in any alarm events.

It is noteworthy that only a subset of each DARPA dataset is utilized for training and testing

in [13], [14], [72]. To simulate the real scenarios in SOC, we utilize the entire dataset for evaluation, dividing the training and testing sets as described in §3.1.6.2. We retrained the model and performed detection using their code. Additionally, we noticed FLASH and THREATTRACE did not count the FPs within the two-hop distance from the labeled attack entities in the ground truth while counting the TPs within the two-hop distance from the detected entities [14]. To ensure a fair comparison without providing excessive leniency, we report the TP and FP results, as well as the one-hop FP result, in which we exclude the FP entities within a one-hop distance from the ground truth. We also did not count the attack entities in the 2-hop neighborhood of the detected entities as TP like [13], [14], [72] did. These strict experiment settings and data labeling methods can explain the difference between our results and the results in their papers, but we believe it is fair and necessary to assess the performance of PIDS in real-world scenarios.

We evaluated the detection accuracy on CADETS, TRACE, and THEIA from Engagement 3 since those datasets were commonly used by the baselines. Table 3.4 illustrates the comparison among CAPTAIN and the baselines. On all datasets, CAPTAIN demonstrates superior performance in identifying more TPs while maintaining fewer FPs. FLASH can report a fair amount of TPs but produce an excessive amount of FPs. It leverages GNN to learn k -hop neighborhood structures. While this technique shows promising performance when there is a significant anomaly within k hops, it could degrade when the training set is limited [13]. KAIROS can filter out FPs with the Anomalous Time Window Queue [14]. Although it guaranteed decent detection accuracy at the time-window level; the entity-level accuracy, however, is not satisfactory. NODLINK detects most attack processes. However, missing other relevant entities, such as files and network sockets, requires additional expert efforts to investigate the reported alarms. MORSE and CAPTAIN achieve a lower false positive rate in node-level detection. This is attributed to the semantic-rich alarms that provide additional information for alarm filtering. For instance, a file corruption alarm involves

two entities: the process and the corrupted (benign) files. As the corrupted files should not be reported as malicious entities, they can be easily filtered out. This explains why MORSE triggers thousands of false alarms at the event level, but only hundreds of nodes are incorrectly alarmed. For embedding-based PIDS, there is no such semantics for alarm filtering. We include the filtered results in Table 3.4 as we believe it highlights an advantage of rule-based PIDS.

CAPTAIN successfully detects all attacks in the dataset. As an event-level detector, CAPTAIN does not report entities that are not directly related to the attack events. For example, if a malware is downloaded and executed, CAPTAIN reports the MalFileCreation alarm and the FileExec alarm, which reveals the malware process, file, and parent process. We do not immediately report the entry network node used by attackers to avoid alarm fatigue, which explains the FNs of CAPTAIN. The investigations on the relevant nodes can be conducted after the first response.

3.1.8 Efficiency

In previous sections, we highlighted the simplicity of CAPTAIN’s rule-based detection framework compared to other embedding-based PIDS. In this section, we compare CAPTAIN with the SOTA PIDS FLASH, KAIROS, NODLINK, SHADEWATCHER², and MORSE to evaluate their efficiency by running their code on our test environment.

3.1.8.1 Detection Latency

The detection latency analysis encompasses three dimensions: buffer time, preprocessing time, and detection time. We have already introduced buffer time in §2.2.1. Preprocessing time refers to the duration taken to convert the raw audit logs into a data structure that the detection systems can process. It involves data parsing, data cleaning, noise reduction, preprocessing, feature extraction,

²Our evaluation was based solely on the open-sourced portion of SHADEWATCHER.

and so on. Detection time refers to the interval between the completion of data processing and the moment the result is produced. It is noteworthy that since the detection granularity is different in different PIDS, the detection time might not reflect the actual latency of each system. For instance, a detector based on the whole graph may take longer to deliver detection results of each graph compared with a detector on the entity/event level. However, this does not necessarily mean the former is slower because a large graph could contain many entities and events. Therefore, we calculate the total time spent on detection across the entire dataset.

We evaluated the latency on TRACE and CADETS from DARPA Engagement 3 since they are covered in the experiments of all baselines. Our evaluation focuses on the testing stage since training can be conducted offline. We modified their code to obtain unbiased results. For example, FLASH divides the data processing into two stages in their code and uses files to store intermediate results. And KAIROS employs the PostgreSQL database and stores the intermediate results in files. These are the steps not required in a real-time streaming pipeline. Consequently, we exclude the I/O time to focus solely on measuring the “pure” preprocessing and detection time.

The result is shown in Table 3.5. Unlike other PIDS that embeds the graphs and thus require a buffer time ranging from several minutes to hours, CAPTAIN processes audit logs in a streaming fashion, eliminating the need for any buffer time. CAPTAIN is also faster in preprocessing because the logs are not preprocessed for the machine-learning model. Moreover, the tags of CAPTAIN are much simpler than the state vectors used by FLASH and KAIROS. By avoiding the use of text embedding or GNNs to aggregate semantic and contextual graph information, CAPTAIN achieves detection speeds over 10 times faster than the baseline methods. Lastly, compared to the rule-based PIDS MORSE, CAPTAIN only takes a few seconds longer to detect logs spanning 4 to 7 days, once again demonstrating the superiority of rule-based PIDS in terms of latency.

Table 3.5: Comparison of detection latency

	Buffer Time	Preprocessing Time	Detection Time
Engagement3 TRACE			
FLASH	57:49	107:50	64:24
SHADEWATCHER	N/A ¹	100:22	3:40 ²
NODLINK	00:10	135:42	2:48
MORSE	0	58:20	1:29
CAPTAIN	0	58:20	1:31
Engagement3 CADETS			
KAIROS	15:00	15:34	29:46
FLASH	82:52	18:57	7:41
NODLINK	00:10	6:18	6:41
MORSE	0	7:22	1:19
CAPTAIN	0	7:22	1:23

¹We did not find a clear number in their codes or paper.

²SHADEWATCHER extracts the last 10% interactions as the testing set, while the testing set of us is around 2.5 times larger.

3.1.8.2 Runtime Overhead

Another important metric reflecting the efficiency of a PIDS is the runtime overhead to the system. In this section, we evaluate the runtime overhead of CAPTAIN during detection and conduct a comparative study with other SOTA PIDS. We also analyze the memory consumption of CAPTAIN during the training stage.

Runtime Overhead in Detection: We use the Python `resource` module to evaluate the resource consumption of detectors on CPU mode. As admitted in [15], GPUs may not be available in most real-life threat detection scenarios. We calculate the total CPU time in user mode. The result is shown in Fig. 3.3a. Please note that the total CPU time used in Fig. 3.3a could be more than the detection time in Table 3.5 because of the multi-core CPU usage by deep learning models. Since CAPTAIN only relies on straightforward arithmetic operations on real-number tags, it requires significantly less CPU time than the embedding-based PIDS (around 2% of FLASH and

around 0.02% of KAIROS), which need complex matrix and vector operations due to their use of neural networks. Moreover, CAPTAIN’s fine-grained detection rules only increase CPU time by a modest 5.6% compared to MORSE. Given the significant improvement on MORSE in detection accuracy (over 90% reduction in false alarms) and the substantial decrease in CPU time compared to other embedding-based PIDS, we believe the slightly higher CPU usage than MORSE is acceptable in real-world detection scenarios.

We use the `psutil` library in Python to monitor the live memory usage during detection. The comparison of memory usage over time is shown in Fig 3.3b. CAPTAIN finishes detection much faster than the embedding-based PIDS (as already shown in Table 3.5) and achieves the lowest memory usage throughout the entire detection process. The memory usage and detection time of CAPTAIN and MORSE are similar, shown by the overlapping curves in Fig. 3.3b. The slight difference in memory usage between CAPTAIN and MORSE is due to the storage of fine-grained rules. In general, CAPTAIN retains the lightweight and fast characteristics of the rule-based PIDS.

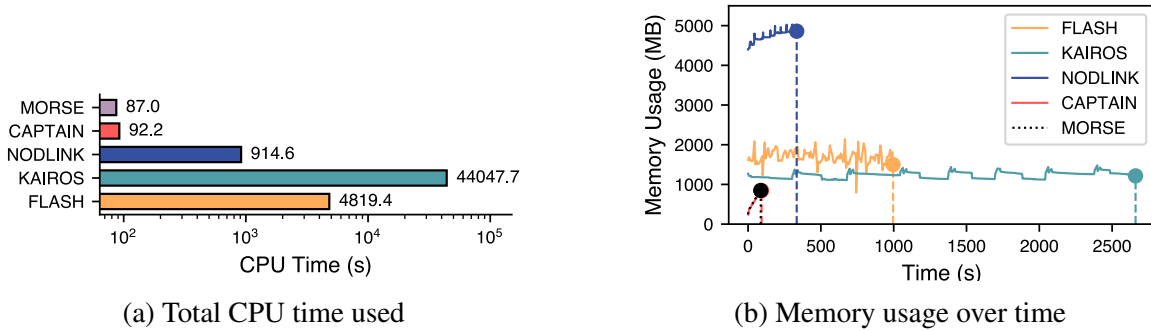


Figure 3.3: Comparison of resource consumption when detecting on DARPA Engagement 3 CADETS. In Fig. 3.3b, the memory usage curve of CAPTAIN and MORSE are overlapped, showing their similar efficiency performance.

Runtime Overhead in Training: The training of the detectors can be performed offline, where users can allocate ample time and computing resources, including GPUs. Therefore, we did not evaluate the runtime overhead and the time used for training. However, as CAPTAIN records and

propagates the gradients for each tag among the big provenance graph, a reasonable concern is that saving those gradients would consume much memory when there is a dependency explosion problem. We evaluated the number of non-zero gradients saved by CAPTAIN during training. The results, shown in Fig. A.1, illustrate that most nodes only need to store a small number of non-zero gradients during training. Due to the space limit, the detailed analysis can be found in §A.5.1.

3.1.9 Resilience Against Adversarial Attacks

In this section, we discuss the robustness of CAPTAIN against two main-stream adversarial attacks: adversarial mimicry attack and training set poisoning attack.

3.1.9.1 Adversarial Mimicry Attack

Mimicry attacks on PIDS involve altering the provenance data and incorporating more benign features to “mimic benign behaviors”, thereby evading detection. In this section, we evaluated the robustness of CAPTAIN against mimicry attacks using the attack methodology in [13], [60], i.e., inserting benign structures into the attack graphs. Our mimicry attack contains two steps. First, we extract some events of benign system entities from the normal training data. Next, we create “fake” events by substituting the benign entities with the attack entity, simulating the attack entity performing activities similar to those of the benign entities. To verify the effectiveness of the mimicry attack, we use FLASH as the baseline. For evaluation purposes, we use the CADETS from Engagement 3 because of its relatively small scale. The details about our mimicry experiment can be found in the Appendix §A.5.2. The result is shown in Fig. 3.4.

In general, both CAPTAIN and FLASH showed robustness against our mimicry attack attempt to some extent. The attack entity `/tmp/test` is still detected by both systems even after we add some normal activities. However, we see a significant drop in the anomalous score of FLASH

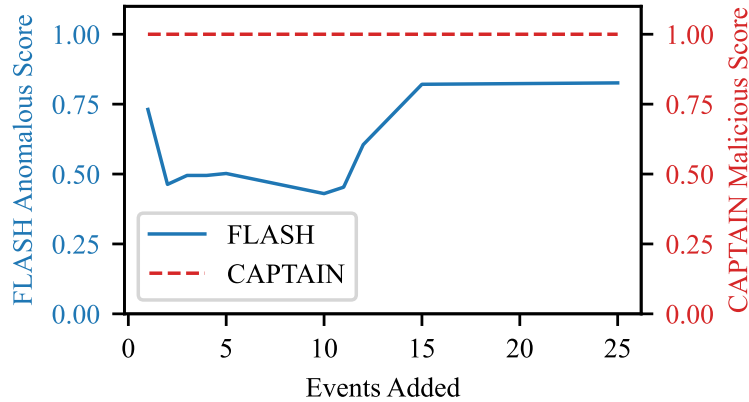


Figure 3.4: Adversarial mimicry attack against CAPTAIN and FLASH (use the attack entity /tmp/test as an example)

when we introduce a relatively small number of events (fewer than 10), which is also confirmed in their paper [13]. On the other hand, CAPTAIN remains unaffected against the mimicry attack, demonstrating superior robustness compared to the baseline.

The robustness of CAPTAIN results from three main reasons. First of all, the propagation of CAPTAIN is guided by heuristic-based rules, while the propagation and aggregation of FLASH and other embedding-based PIDS are based on the neural network. Second, CAPTAIN is an event-level detector with a finer granularity than entity-level detectors. As mentioned before, CAPTAIN gives the detection result for each streaming event without any buffer time, which means any mimicry insertion after the real attack activities is useless. For those PIDS with the buffer time, any events within the same time window and related to the same entity can affect the final detection result. Third, CAPTAIN is not an anomaly-based detector. In other words, the detection is performed based on the conservative, heuristic rules, while the normal data is only used to reduce the false alarms. Therefore, adding normal features can not compromise the detection capability as it is guaranteed by the conservative detection rules.

3.1.9.2 Training Set Poisoning Attack

Training set poisoning refers to the malicious manipulation of the training set. In the context of cyber security and intrusion detection, it usually involves polluting the benign training set with adversarial events/entities. When the detector is trained on a “benign” dataset that has been polluted, it learns patterns or features introduced by attackers. These patterns and features, once learned, could assist attackers in evading detection in future instances.

We add the regularizer term to the loss function to control the sensitivity of CAPTAIN to the training set. This enhances CAPTAIN’s robustness against dataset poisoning attacks. In this section, we present a real case of Pine Backdoor & Phishing Email attack from Engagement 3 TRACE. In this attack, a vulnerable `pine` connected to the email server `128.55.12.73`, downloaded and executed an email attachment. However, IP `128.55.12.73` had multiple activities and caused some “false” alarms in the training set. Please note that this violates our assumption that the training set must not be compromised by attack entities or events. Nevertheless, this makes it a perfect example to test the robustness against data poisoning attacks.

We analyze the critical parameters for detecting this attack: the initial integrity tag (a) of IP node `128.55.12.73`, to illustrate the effect of the regularization term. Without the regularization term in the loss function, `128.55.12.73` will be assigned a high integrity score (near 1.0) due to its activities in the training set, which lets us miss the phishing email attack starting from it. However, with the help of the regularizer terms ($\alpha = 10$), CAPTAIN is more cautious during training. Like the “grey node” motivating example in §3.1.2, CAPTAIN assigns the integrity score as the mediocre 0.488, which removes many false alarms caused by this IP while successfully capturing the phishing email attack during testing (more details can be found in §3.1.11). The propagation rate of event (`pine, read, 128.55.12.73`) is also more conservative against the polluted training set due to the regularizer term.

3.1.10 Ablation Study

3.1.10.1 Individual Adaptive Parameters

We conduct an ablation study on each adaptive parameter. As each parameter can be learned independently, we have seven separate experiments, each optimizing a subset of the parameters, i.e., $\{A\}$, $\{G\}$, $\{T\}$, $\{A, G\}$, $\{A, T\}$, $\{G, T\}$, and $\{A, T, G\}$.

According to Fig.3.5, adjusting the threshold (T) alone may not have much effect on the detection results because without changing the initial tags and the propagation rates, the updated tags still tend to be 0 or 1. Fig.3.5 also shows that tuning all these three parameters can accelerate the training process. Although $\{A, T\}$ and $\{A, T, G\}$ achieve similar good detection results on the testing set, $\{A, T, G\}$ converges using fewer epochs during training. It should be noted that tuning A plays the most major role in mitigating false alarms among these three adaptive parameters. In certain datasets (S-3 and C-3), adjusting A alone sometimes results in fewer false alarms than tuning parameters A , G , and T together because, without the benefit of learning propagation rates and thresholds, we have to assign higher integrity scores to more nodes to counteract false alarms during training. However, reducing false alarms more crudely carries the risk of being excessively lenient.

3.1.10.2 Learning Rate in Optimization

We conducted an experiment on different learning rates on the DARPA Engagement 3 CADETS. The detailed results can be found in §A.5.3. As illustrated in Fig. A.2, a higher learning rate usually leads to a sharper loss decline during the initial training epochs. However, for ongoing training, a lower learning rate may yield a finer-tuned search process and lower loss, necessitating a balance between training speed and training quality.

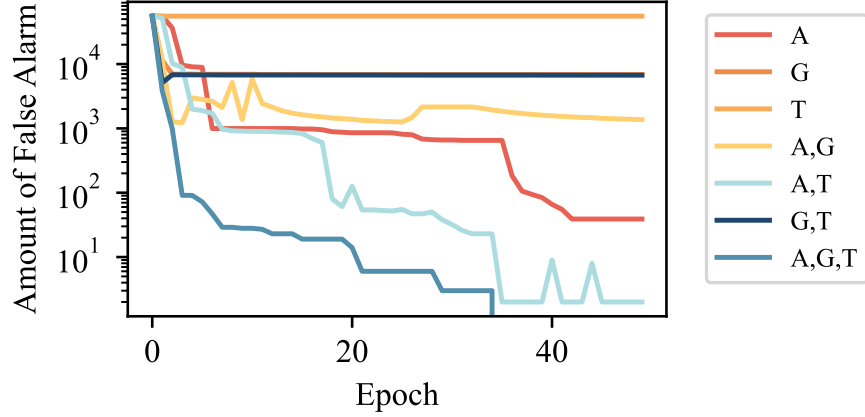


Figure 3.5: Ablation study on each parameter.

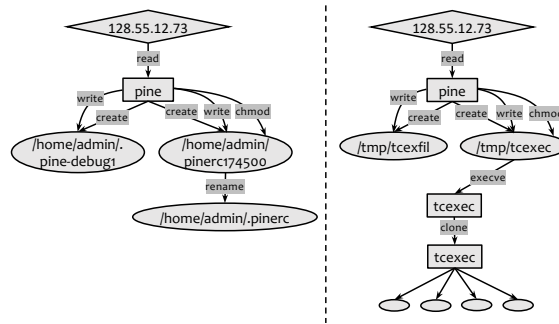
3.1.11 Case Study

To illustrate the interpretability of CAPTAIN’s detection process, we delve into the training outcomes of the E3-TRACE dataset as a case study. These cases also show the improvement of CAPTAIN compared to existing rule-based PIDS.

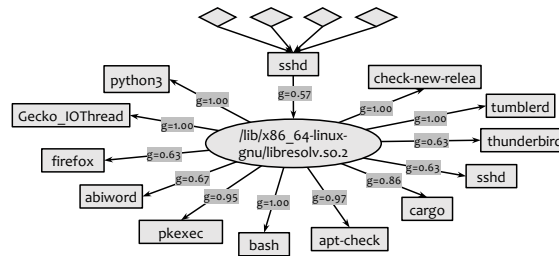
3.1.11.1 Better Detection Capability due to Fine-Grained Rules

The most significant improvement of CAPTAIN is its fine-grained detection capability, enabled by the adaptive parameters. Unlike the discrete, human-assigned trustworthiness levels and universal rules used in [16], [17], [28], the fine-tuned numerical parameters of each entity and event in the graph can better help distinguish nuances between benign and malicious patterns. Fig. 3.6a demonstrates how CAPTAIN outperforms the baselines in distinguishing between two similar behaviors using fine-tuned parameters. In both cases, process `pine` reads from `128.55.12.73` (possibly the email server) and creates, writes, and changes the permissions of a new file. It is difficult for previous rule-based PIDS [16], [17] to assign a tag for `128.55.12.73`. Take the SOTA MORSE as an example: If MORSE initialize the node `128.55.12.73` as a benign node with the data in-

tegrity score as 1.0, it would miss at least the first stage of the attack. On the other hand, if MORSE assigns the initial tag as “Untrusted” (this is what they did in order to capture all attacks), it would trigger multiple false “Malicious File Creation” and “Change Permissions” alarms on the benign graph. Instead, CAPTAIN assigns a moderate initial tag a_n , 0.488, to this IP and slightly tunes down the threshold thr for the event $(pine, create, /home/admin/pinerc174500)$ from 0.5 to 0.478. As $0.478 < 0.488$, the creation of the file will not be alarmed, but in real attacks, the thr for the event $(pine, create, /tmp/tcexec)$ is still 0.5, meaning creating a malicious file would still trigger alarms ($0.5 > 0.488$).



(a) The graph on the left side represents normal behavior in the training set, while the graph on the right side depicts an attacker executing a malicious email attachment.



(b) CAPTAIN assigns different propagation rates (g_e) to events to control dependency explosion more precisely.

Figure 3.6: Case study from Engagement 3 TRACE. We omitted some irrelevant details to keep the graphs clean and succinct.

CAPTAIN’s fine-grained rules also help address the dependency explosion issue caused by `libresolv.so.2`. Unlike MORSE, which sets universal decay/attenuation factors for all processes, CAPTAIN learns distinct propagation rates for different events. As shown in Fig. 3.6b, CAPTAIN assigns lower propagation rates to events that frequently cause dependency explosions (e.g., being read by `sshd` or `firefox`) while maintaining high propagation rates for other events. Even for the same process, `firefox`, CAPTAIN learns to suppress the dependency propagation when reading this file ($g_e = 0.63$) and remains sensitive to maliciousness propagation when loading this file ($g_e = 0.99$). Consequently, CAPTAIN avoids being overly lenient for potential attack events when addressing the dependency explosion issue.

The fine-grained parameters of CAPTAIN also outperform many embedding-based PIDS in detecting mimicry attacks. For instance, in Fig. 3.6a, the attacker can modify the file name of `/tmp/tcexec` with the knowledge that `pine` might interact with a file like `/home/admin/pinerc+numbers`. Since generalization techniques like Word2Vec and Sentence2Vec are widely used in embedding-based PIDS, they can manipulate the file name by appending random digits to `pinerc`, making it similar to `pinerc174500` after embedding. However, to evade CAPTAIN, the attacker must know the exact file name of `pinerc174500`. In addition, deep learning usually relies on the amount of training data. But there are only four events related to file `/home/admin/pinerc174500` during the 7-day training set, making it challenging for the neural networks to learn such patterns.

3.1.11.2 Robustness brought by Differentiable Tag-Propagation Framework

Some previous rule-based PIDS [29] can customize their detection system using benign training data. However, the adjustments are based on the training set rather than the detection results. For example, NODOZE assigns a distinct anomaly score to each event based on its frequency in

the training set. However, due to training set poisoning and living-off-the-land attack techniques, the frequent events in the training set cannot be fully trusted. For instance, the events related to `dbus-daemon` are very common in the training set of TRACE in Engagement 3. Typical events such as `dbus-daemon` writes or reads `/var/run/dbus/system\_bus\_socket` occur over 100,000 times during seven days. Such events and the `dbus-daemon` entity would be assigned with a low anomaly score in frequency-based systems like NODOZE. However, the low anomaly scores allow the attackers to deceive the detectors using living-off-the-land techniques related to the D-Bus process [93]. In contrast, CAPTAIN does not change the parameters related to `dbus-daemon` because, despite the large number of events associated with `dbus-daemon` in the training set, no false alarms are triggered by it. CAPTAIN adjusts the parameters related to an event based on the false alarms caused by it rather than the frequency of it.

3.1.12 Discussion

3.1.12.1 *Overfitting in Parameter Learning*

Overfitting is an important issue in machine learning. As CAPTAIN learns the benign patterns in the one-class manner starting from conservative settings, overfitting is, however, a preferred and safer strategy. In one-class learning, we want to generate a tightest boundary for the target class (benign patterns). Therefore, CAPTAIN avoids generalizing excessively to minimize the risk of evasion, as demonstrated in our example in §3.1.11.

3.1.12.2 *More Optimization Algorithms and Online Learning*

In this paper, we utilize the gradient descent algorithm for optimization due to its simplicity and clearness. We can reach optimal values more quickly and accurately, using algorithms such as *Nesterov Accelerated Gradient (NAG)* [94], *Adadelata* [95], *RMSprop*, *Adagrad* [96], *Adam* [97],

AdaMax, *Nadam*, and so on. These algorithms require the first-order gradient, offered by the differentiable detection framework in CAPTAIN. We leave the implementation of these algorithms as the extensions of CAPTAIN.

Long-term maintenance is important in real-world scenarios. Although we assess CAPTAIN in an offline setting, CAPTAIN’s learning module can compute loss upon receiving the feedback for each event. Subsequently, the learning module can promptly update the adaptive parameters for subsequent detection.

3.1.13 Conclusion

This paper introduces CAPTAIN, a rule-based PIDS capable of automatically adapting to detection environments, enhanced by three adaptive parameters: tag initialization (A), propagation rate (G), and threshold (T). The differentiable detection framework enables the optimization using the gradient descent algorithm. To our knowledge, this is the first effort to incorporate gradient descent methods in optimizing rule-based PIDS. We evaluated CAPTAIN on several datasets. The results demonstrate the superior detection capability, significantly reducing false alarms, detection latency, and runtime overhead, outperforming the SOTA baselines.

3.2 Online, Adaptive Provenance-Based Intrusion Detection

CAPTAIN, presented in Section 3.1, adapts rule-based PIDS to a target environment through one-shot offline training on benign data. This section presents CAPTAIN +, an extension that addresses a complementary deployment challenge: real-world detection environments evolve continuously, and parameters that are optimal at deployment time gradually become suboptimal as system behavior and attack patterns drift. Because the system design of CAPTAIN + is built directly on the architecture, threat model, and differentiable detection framework introduced in Section 3.1, the

discussion below focuses on what CAPTAIN + adds rather than re-deriving the shared components.

3.2.1 Motivation

Production SOC environments are not static. Software updates, configuration changes, and shifting user behavior cause the distribution of benign provenance events to drift, while adversaries continuously refine their tradecraft to evade existing detectors. A PIDS whose parameters were tuned once at deployment will, over time, accumulate false positives on newly common benign behavior and miss attacks that diverge even modestly from the training distribution. The offline gradient-based tuning introduced in Section 3.1 addresses environment-specific configuration but does not by itself sustain accuracy under such drift. CAPTAIN + closes this gap by introducing an online learning module that updates detection parameters during inference in response to analyst feedback, without requiring periodic offline retraining.

3.2.2 Online Learning Module

CAPTAIN + retains CAPTAIN’s differentiable detection framework and its three adaptive parameters, the initial tag value A , the tag propagation rate G , and the alarm threshold T , with T extended to operate at fine-grained, per-entity granularity to support targeted online updates. On top of this shared substrate, CAPTAIN + adds three online-learning-specific design elements.

Feedback-driven update trigger. Whereas offline training drives gradients from an aggregated loss over a static dataset, online updates in CAPTAIN + are event-driven. Each false positive flagged by an analyst, and each false negative discovered through downstream investigation, triggers a parameter update localized to the entities and tag-propagation paths involved in the misclassified event. This mechanism allows the system to respond immediately to operationally relevant errors rather than waiting for batched retraining cycles.

Dynamic, error-type-aware learning rate. Online learning cannot tolerate the slow, fixed learning rate used in offline training: feedback arrives sparsely, and each individual correction must move the parameters meaningfully toward the right answer without destabilizing previously learned behavior. CAPTAIN + uses an adaptive learning rate that differentiates between false positives and false negatives. The false-positive learning rate scales gradually with the observed count of similar errors, allowing the system to correct systematic biases without overreacting to a single noisy report. The false-negative learning rate is set to a larger, fixed value, since a missed attack is a critical failure that warrants a stronger correction; in addition, each false negative inflates the subsequent false-positive learning rate by a multiplicative factor, since a missed attack implies the system has drifted toward excessive leniency overall.

No-regularization regime. The regularization terms used during offline training to prevent overfitting are deliberately omitted in the online phase. In a streaming setting with sparse feedback, regularization would dampen the very updates that the system relies on to respond to drift; without it, each feedback signal exerts its full intended effect on the parameters it touches.

3.2.3 Evaluation Highlights

CAPTAIN + was evaluated on the same DARPA Transparent Computing and OpTC datasets used in Section 3.1, together with the PKU ASAL dataset, which contains multi-host APT emulations spanning Linux and Windows environments. Because most of the experimental procedures and baseline comparisons mirror those already reported for CAPTAIN in Section 3.1, this section focuses on the result that is specific to the online-learning extension: long-term performance under concept drift.

Long-term false-positive control under concept drift. The most distinctive evaluation of CAPTAIN + tracks how the cumulative number of false positives grows over time as the system pro-

cesses a long stream of provenance events. Figure 3.7 reports this comparison on three DARPA Engagement 3 datasets, CADETS, TRACE, and THEIA, using the number of processed events as a proxy for wall-clock progression.

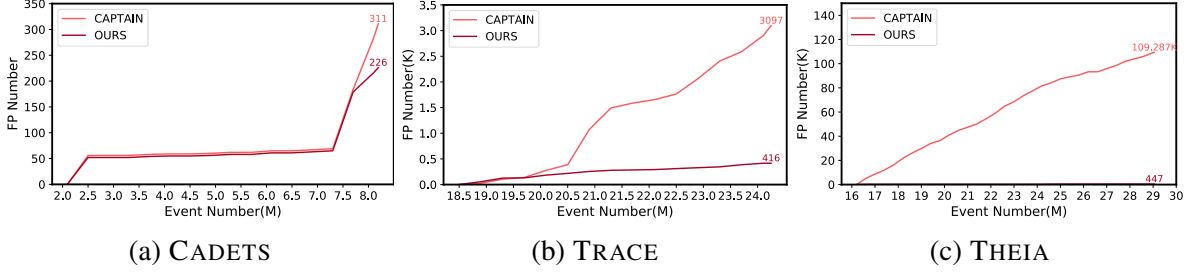


Figure 3.7: Cumulative false positives of CAPTAIN + vs. CAPTAIN over time, measured by the number of processed events, on three DARPA Engagement 3 datasets.

The pattern is consistent across all three datasets. CAPTAIN controls false positives well in the early portion of the stream, but its cumulative false-positive count rises rapidly as detection progresses and the benign-behavior distribution drifts away from the offline-training snapshot. CAPTAIN +, by contrast, maintains a substantially flatter trajectory throughout the run; the gap is most pronounced on TRACE and THEIA, where the benign workload undergoes the most pronounced drift, and narrower on CADETS, where the offline-trained CAPTAIN baseline already achieves a low false-positive rate and leaves limited room for further reduction. The online learning module is also responsible for catching attacks that CAPTAIN misses on repeat occurrence: when a previously unseen malicious event reappears later in the stream, the parameter and tag updates triggered by the first detection allow CAPTAIN + to flag the second instance correctly.

Robustness and overhead. Two additional results round out the evaluation. First, the dynamic learning rate combined with localized per-entity updates makes CAPTAIN + resistant to feedback-poisoning attacks: an adversary injecting misleading benign feedback can cause only a bounded, gradual increase in false positives without ever flipping a true malicious event to benign. Sec-

ond, the runtime overhead introduced by online learning is dominated by per-event tag-update bookkeeping rather than gradient computation, keeping CAPTAIN + within the same lightweight operating regime as CAPTAIN and preserving the real-time, interpretable properties that motivated the rule-based design.

3.2.4 Section Summary

CAPTAIN + extends CAPTAIN’s offline, environment-specific adaptation with an online learning module that sustains detection accuracy under concept drift without periodic retraining. By reusing the differentiable detection framework of Section 3.1 and adding only an event-driven update trigger, a dynamic FP/FN-aware learning rate, and a no-regularization update regime, CAPTAIN + keeps the additional complexity localized and preserves the lightweight, real-time, interpretable properties of its predecessor while making the system viable for long-running production deployments.

3.3 Burst-Resilient Real-Time Detection via Serverless GNN Inference

3.3.1 Motivation and Threat Model

Graph-based intrusion detection has emerged as a critical application of graph analysis in cybersecurity. In this paradigm, system behaviors recorded in audit logs are transformed into provenance graphs, where nodes represent system entities (e.g., processes, files, network nodes), and edges represent system-level events (e.g., file read/write, process creation). These graphs provide a natural way to model the causal and temporal relationships underlying cyberattacks and anomalous behavior. Recently, Graph Neural Networks (GNNs) have demonstrated strong promise in this domain by learning complex patterns that enable context-aware intrusion detection. However, existing GNN inference architectures struggle to scale in this setting, as they must (1) maintain stable

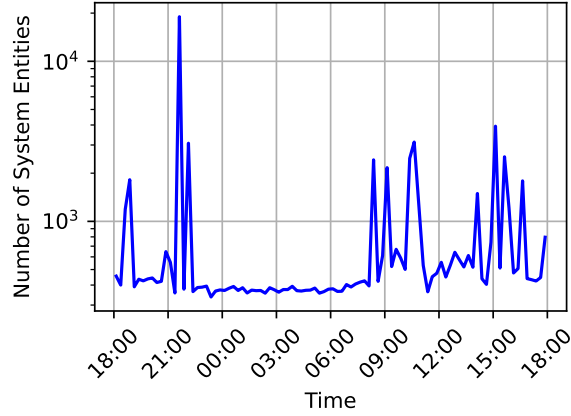


Figure 3.8: Fluctuations in detection workload over time in the DARPA TC dataset [54]

and low detection latency, while also (2) handling inherently irregular and bursty workloads (i.e., workloads with sudden surges in demand). First, in intrusion detection systems, it is particularly crucial for GNN inference to maintain stability and low latency. If the inference time is too long, there is a high risk of missing the window to mitigate real attacks, potentially leading to irreversible consequences. Therefore, ensuring fast and consistent inference performance is essential for effective threat response. Second, such systems often face unstable and bursty workloads (Figure 3.8), while malicious activities account for only a small fraction of overall network behavior, leading to highly unpredictable and sporadic patterns in the analysis workload.

Serverless design offers advantages in handling bursty workload and maintaining consistent low detection latency. A serverless system is a cloud model where code runs on-demand without managing servers. It provides a cost-efficient pay-per-use model with no charges for idle resources, while scaling instantly and seamlessly to handle traffic spikes. Additionally, it enables faster development through an event-driven model and ensures finer-grained resource utilization, making it especially well-suited for workloads that are intermittent, unpredictable, or highly event-driven.

Contributions. This paper introduces GraphFaaS, a serverless architecture for low-latency, scalable GNN-based intrusion detection. Our contributions include: (i) A **provenance-aware graph**

construction pipeline that avoids redundant computation by leveraging temporal locality in systems and applying two-stage filtering, based on structural proximity and frequency, to isolate relevant subgraphs. (ii) A **serverless node embedding layer**, which parallelizes node-level feature transformations across dynamic workloads with a feature-length-aware execution strategy to ensure each unit completes within a fixed latency bound while minimizing overhead. (iii) A **scalable GNN inference mechanism**, adapted for serverless execution, which partitions large provenance graphs into balanced subgraphs using a greedy best-fit algorithm and a vertical-scaling fallback to handle extreme cases where graph fragments exceed preset limits, (iv) An **end-to-end implementation** of GraphFaaS on the OpenFaaS platform, evaluated on the DARPA TC dataset [54]. Our system achieves a 6.7x reduction in average detection latency and a 64% reduction in coefficient of variation (CV) compared to a state-of-the-art baseline, while maintaining equivalent detection accuracy.

3.3.2 Background

Serverless for GNN. Recent efforts have tackled the challenges of efficient GNN execution from both hardware and system perspectives. GNNAdvisor [98] optimizes GPU utilization by tailoring runtime behaviors based on GNN model and workload characteristics, but focuses mainly on training/inference under static resources. λ Grapher [99] enables efficient GNN serving by exploiting request-level graph locality and fine-grained resource control. Dorylus [100] leverages serverless threads with CPU servers for scalable GNN training, achieving cost-efficiency via computation separation, yet it does not address serving workloads. Fograph [101] targets low-latency GNN inference for IoT by using fog nodes close to data sources, mitigating cloud communication overhead. However, it relies on specialized edge deployments rather than general-purpose serverless infrastructures. While some works explore migrating graph processing to FaaS plat-

forms, they suffer from poor scalability due to high communication overhead and coarse-grained execution [102]. Unlike prior systems that focus on static provisioning, training performance, or specialized edge deployments, GraphFaaS uniquely enables low-latency, scalable GNN inference for intrusion detection by combining fine-grained graph filtering, adaptive partitioning, and dynamic resource scaling within a serverless framework.

Typical Detection Workflow of GNN-based PIDS Log processing and graph construction.

The source of intrusion detection is the log stream, i.e., system log data arranged in chronological order. Graph construction extracts system entities and events from the log stream, representing them as nodes and edges in the provenance graph, and stores them in a graph database.

Node-level embedding. The second step is node-level embedding, also known as featurization [103], which transforms the static attributes of each node into numerical vectors. These vectors act as the initial node representations for the GNN. In provenance graphs, node attributes are typically represented as textual data, for example, process names and command lines for process nodes, file paths for file nodes, and IP addresses and ports for network nodes. To embed these textual features into a vector space, common techniques such as word2vec and doc2vec are often employed.

GNN. The node-level embeddings provide the initial representations for GNN. During message passing, the GNN propagates and aggregates node features across its neighbors. The scope of this propagation, i.e., the receptive field, is determined by the number of GNN layers. After propagation, each node is assigned an updated vector representation. To train the GNN parameters, typical optimization objectives include node classification, edge type prediction, or graph reconstruction.

Result Aggregation. GNN generates an embedding for each node. For an intrusion detection system, these embeddings are further analyzed to determine whether an intrusion has occurred. Common approaches in existing work include applying clustering methods, fixed thresholds, or

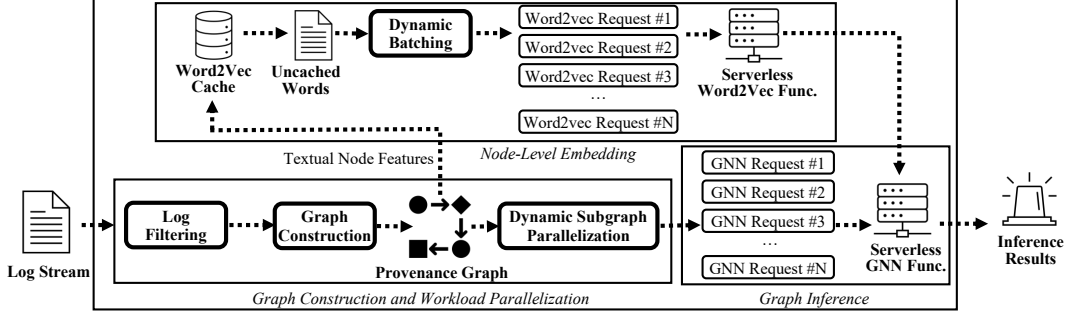


Figure 3.9: Overview of GraphFaaS Framework with three components: (i) graph construction, (ii) serverless node-level embedding, and (iii) serverless GNN inference.

outlier detection techniques to identify anomalous nodes.

3.3.3 Design

As shown in Figure 3.9, the GraphFaaS framework is composed of three main components: (i) graph construction, (ii) serverless node-level embedding, and (iii) serverless GNN inference.

Graph Construction. We start by constructing the provenance graph from the log stream. Although the provenance graph can be very large, we find that most of its structure stays the same between two detection intervals. To take advantage of this, we use log filtering: instead of reprocessing the entire graph, we focus only on the parts that have changed. Specifically, we keep nodes that are within a $2K$ -hop distance of the active nodes (where the GNN has K layers), which avoids redundant computation. We can also apply frequency-based filtering [29], [104]: we keep only the edges, nodes, and their $2K$ -hop neighborhoods that occur infrequently in the training data, since rare patterns are usually more important for detection. Finally, the filtered subgraphs are split into subtasks and run in parallel, allowing us to fully leverage serverless auto-scaling.

Serverless Node-level Embedding. Our goal is to ensure consistent and low inference latency under bursty workloads. The core idea is to divide the workload into small, parallel execution units, each requiring only a short processing time below a predefined threshold. As the workload

changes, the number of execution units adjusts accordingly. Leveraging the automatic scaling capability of the serverless architecture, computing resources are allocated adaptively to match demand. This design guarantees that all execution units complete within the time threshold, regardless of their number. For instance, when a large workload arrives, it is split into more execution units, and the serverless platform automatically scales up by launching additional instances to handle them. Conversely, when the workload is low, the platform scales down to release resources and avoid waste. GNN inference typically consists of two stages: node-level embedding and message passing with aggregation. We present how each of these stages can be adapted to a serverless architecture.

As introduced in §3.3.2, node-level embedding transforms node attributes into vectors that serve as the initial representations for GNNs. Because static node attributes are independent of one another, the embedding process is naturally parallelizable across nodes. We implement a serverless function that takes a node attribute as input and returns its vector embedding. Since node attributes are typically represented as textual strings, and execution time for most embedding methods depends on string length, we partition execution units according to feature length. Shorter strings are grouped into a single execution unit, while longer strings are processed separately. This strategy ensures that each execution unit completes within the designated time threshold while also reducing the overhead of excessive parallelization, such as network transmission and packet processing costs.

GNN Inference. We implement a serverless function that takes the initial node embeddings and edges as input and outputs the updated node vectors. Similar to the embedding stage, the workload is divided into execution units so that the serverless platform can scale automatically to handle bursty workloads. Since the model remains fixed during detection, inference latency is primarily determined by the graph size. To efficiently process large graphs, we partition them

into subgraphs whose sizes are close to a predefined threshold. This partitioning strategy needs to address the key challenge of workload balancing, specifically, how to partition or pack the graph such that each subgraph stays within the desired size limit while minimizing the total number of subgraphs, thereby avoiding excessive waste of computational resources. We design a greedy best-fit algorithm to partition the graph (See Appendix §B.1 for more details). Due to the dependency explosion problem in provenance graphs, even the smallest subgraph (a central node and its k-hop neighbors) may exceed the preset size limit. In such cases, vertical scaling is triggered, allocating more compute resources (e.g., CPU cores, memory) to each serverless function instance, instead of spawning more instances.

3.3.4 Evaluation

We implemented GraphFaaS in Python based on OpenFaaS, an open-source serverless platform. We compared GraphFaaS with Flash [13], a state-of-the-art PIDS. To ensure a fair comparison, we reimplemented Flash to adapt it to a server-client detection model, where data is sent from the monitored machine to a detection server for analysis. The only difference is that we used a Docker container to simulate a statically provisioned detection environment for Flash, whereas GraphFaaS was deployed using a serverless architecture based on OpenFaaS. All other experimental conditions were kept identical between the two systems. We used a widely adopted dataset from DARPA TC Engagement 3 [54], which contains 11 days of audit logs and includes four attack campaigns.

We verified that GraphFaaS achieves the same detection accuracy as Flash, which is expected since the underlying detection models of Flash remain unchanged. Therefore, the detection results are identical. However, GraphFaaS significantly outperforms Flash in detection latency. As shown in Figure 3.10, on a 11-day dataset, it achieves an average latency of 2.1 seconds, compared to 14.16 seconds for the baseline. In addition, GraphFaaS is more resilient to bursty traffic.

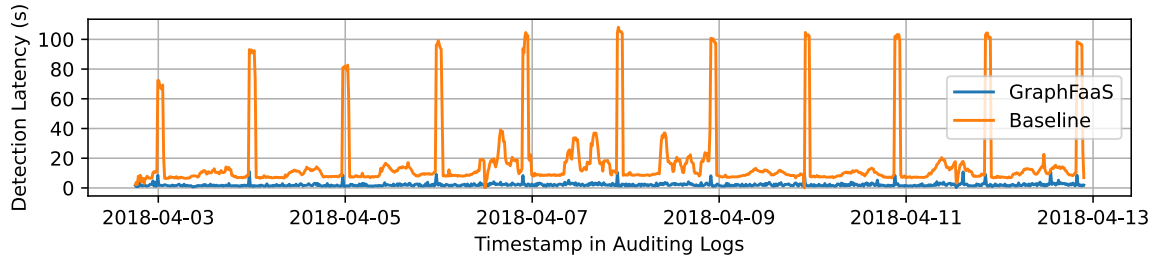


Figure 3.10: Detection latency over time. GraphFaaS: mean = 2.10, STD (standard deviation) = 1.09, CV (coefficient of variation) = 0.52; Baseline: mean = 14.16, STD = 4498.92, CV = 1.46.

It maintains low latency even during sudden surges in workload, as evidenced by its lower standard deviation and coefficient of variation. While occasional latency spikes occur, the delay never exceeds 10 seconds.

3.3.5 Conclusion and Future Work

GraphFaaS leverages the elasticity and agility of a serverless design to ensure consistently low intrusion detection latency. Several challenges remain for future work. First, provenance graphs are particularly vulnerable to the dependency explosion problem [8], [17], where certain super-nodes dominate the graph. Even with workload parallelization, these super-nodes remain a major latency bottleneck. Second, during cyberattacks, the scale of the provenance graph can fluctuate dramatically. To effectively capture long-range dependencies while preserving low latency, it is necessary to dynamically adjust the number of GNN layers and invoke the appropriate serverless functions on demand. We believe GraphFaaS can serve as the basis to address these issues with serverless design. Finally, we plan to evaluate GraphFaaS against a broader class of PIDs and across more datasets.

3.4 Chapter Summary

This chapter addressed the deployability challenges of provenance-based intrusion detection from three complementary angles. CAPTAIN recast rule configuration as a differentiable optimization problem, enabling rule-based detectors to adapt to environment-specific behavior and reducing false alarms by over an order of magnitude without sacrificing their lightweight, real-time, and interpretable properties. CAPTAIN + extended this idea with an online learning module that continuously updates detection parameters during inference, sustaining accuracy under concept drift without periodic retraining. GraphFaaS turned to an orthogonal bottleneck, the bursty and irregular workloads of real detection traffic, and re-architected GNN inference on a serverless platform, reducing detection latency and stabilizing tail performance.

CHAPTER 4

REALISTIC TESTING AND EVALUATION INFRASTRUCTURE

Robust APT defense depends on the ability to test detection systems against realistic threat scenarios and to evaluate them against rigorous, representative benchmarks. The consequences of failing to do so are concrete and measurable. An empirical analysis of the MITRE Engenuity ATT&CK Evaluations [6] found that even leading commercial EDR systems exhibit substantial blind spots when measured against expert-emulated APT campaigns. The results indicate that the operational shortfall of existing detection systems is not hypothetical, and that closing it requires both better detectors and a more realistic evaluation infrastructure capable of revealing such weaknesses in the first place.

As argued in Chapter 1, the field’s current infrastructure for realistic testing and evaluation is inadequate on both fronts: there is no scalable mechanism for generating causally coherent attack scenarios at scale, and the publicly available benchmark datasets are outdated, coarsely labeled, and prone to inflating reported detection performance. This chapter presents two complementary contributions that address these gaps. Section 4.1 introduces AURORA, an LLM-powered classical-planning system that automatically generates actionable, causality-preserving cyberattack chains from CTI reports and MITRE ATT&CK techniques. Section 4.2 builds on AURORA to construct a large-scale, fine-grained, and continuously updatable benchmark dataset, and uses it to expose systematic generalization failures in existing PIDS that prior benchmarks fail to surface. Together, the two efforts form an end-to-end pipeline for realistic testing and evaluation, in which attack emulation feeds dataset generation, and dataset generation in turn drives credible PIDS evaluation.

4.1 AURORA: Causality-Preserving Cyberattack Emulation with LLM-Powered Classical Planning

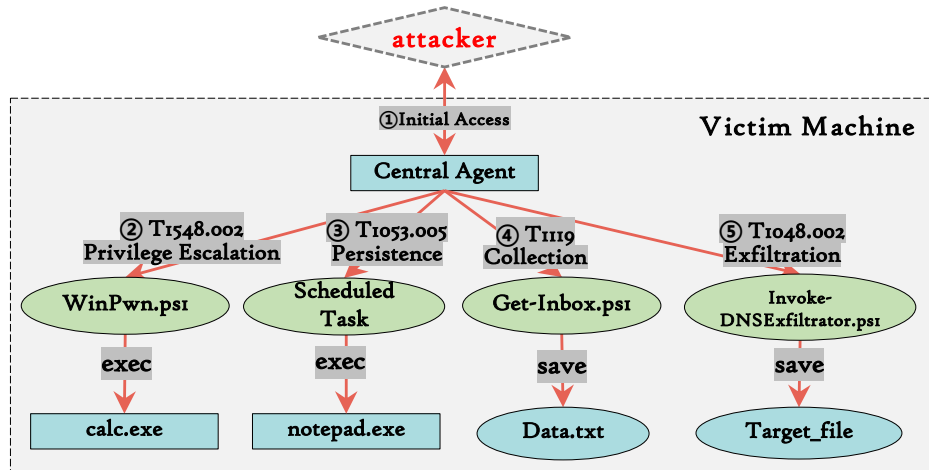
4.1.1 Motivation and Technical Challenges

As established in § 2.3.1, no existing approach to adversary emulation simultaneously satisfies the four properties required for credible defense evaluation, actionability, causal coherence, customizability, and scalability. This section identifies three specific technical challenges that explain the persistence of that gap and that directly motivate AURORA’s design.

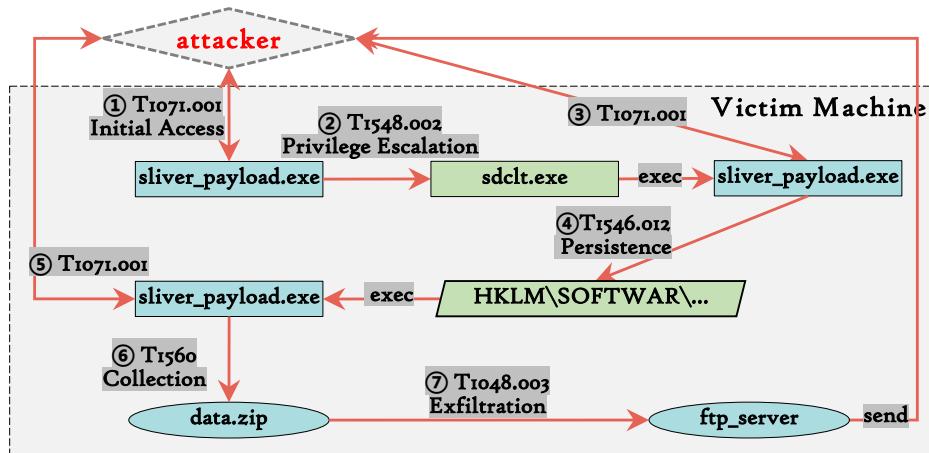
The first challenge is the unstructured nature of cyberattack knowledge. The information needed to construct a realistic attack chain is spread across CTI reports, exploit code, tool documentation, and threat actor profiles, each in heterogeneous formats and varying levels of operational detail. Integrating this knowledge into coherent attack sequences traditionally requires significant human expertise, which is the primary reason existing automated approaches cover only narrow slices of the technique space; the state-of-the-art automated attack planner Chain-Reactor [48], for example, focuses on privilege escalation alone and considers approximately 30 attack actions.

The second challenge is modeling causal dependencies between attack steps. Real adversary behavior is not a sequence of independent actions but a chain in which each step depends on the artifacts and state produced by preceding ones, file handles, credentials, network footholds, process contexts. Prior work [48], [49], [50] has yet to articulate a formal framework that captures these dependencies at a granularity sufficient to drive automated attack composition. As illustrated in Figure 4.1, procedural orchestration tools that sequence atomic actions without modeling such dependencies produce attack chains that read as procedural rather than adversarial.

The third challenge is customizing attacks to specific environments and threat intelligence. A



(a) Each step is executed isolatedly without causality and rely on a central control agent (e.g. Caldera Agent).



(b) More realistic multi-step attacks with each step causality-connected.

Figure 4.1: Causality in emulated attack chains. Without explicit modeling of inter-step dependencies, automated systems produce sequences of actions that do not reflect how real attackers chain operations together.

useful emulation should reflect the behaviors described in a given CTI report while remaining executable under the tools, services, and constraints of a particular target environment. Transforming CTI prose into attack chains that are both faithful to the source report and operationally executable in a concrete environment remains an open problem.

AURORA addresses these three challenges through three corresponding design choices. To handle the unstructured nature of attack knowledge, AURORA uses LLMs to convert heterogeneous third-party attack tools into a standardized, modular representation called *attack actions*. To model causality, it introduces an *Attack Action Linking Model (AALM)* that formalizes inter-step dependencies in terms of preconditions and effects, enabling classical planning over modular actions to produce causality-preserving attack chains. To support customization, AURORA extracts target-actor signatures from CTI reports and uses them as planning constraints, generating attack chains that simultaneously align with the source report and respect the available tools and surfaces of the target environment. The remainder of this section formalizes the attack-action representation and the AALM, describes the system architecture, and evaluates the resulting attack chains against both existing automated approaches and expert-crafted playbooks.

4.1.2 Threat Model and Scope

AURORA is designed as an adversary emulation system, not a penetration testing or red-teaming tool. Whereas penetration testing and red teaming aim to discover vulnerabilities or attack paths in unknown or partially known systems, the goal of adversary emulation is to construct and execute prescribed attack scenarios within a target environment whose configuration is known to the defender. The output of an emulation is therefore not a list of newly discovered vulnerabilities but a multi-stage, causality-preserving attack chain that can be replayed against a defense system under controlled conditions.

Accordingly, AURORA assumes that basic information about the target environment, including operating system versions, installed software, known vulnerabilities, and network topology, is available at emulation time. Such information can be obtained automatically through existing asset inventory and security assessment tools such as OpenSCAP, OpenVAS, and osquery [105],

[106], [107]. AURORA does not consider planning under incomplete information; reasoning about the consequences of partial observability is left to penetration-testing approaches and is outside the scope of this work.

Importantly, this assumption does not restrict AURORA to a single predefined testbed or a fixed environment, as in some prior work [46], [47]. The same AURORA pipeline applies to arbitrary user environments as long as basic system information is supplied: the environment description becomes part of the planning input, and the resulting attack chains are customized accordingly (see § 4.1.5.2). Our overarching goal is for AURORA to produce, from CTI reports and ATT&CK techniques, attack chains that simultaneously satisfy the four properties defined in Section 2.3.1, actionability, causal coherence, customizability, and scalability.

4.1.3 Background: Classical Planning

AURORA frames cyberattack construction as a classical planning problem. At a high level, a planning problem aims to find a sequence of actions that transforms an initial world state into a desired goal state [108], [109]. Formally, a classical planning problem is a tuple $(\mathcal{P}, \mathcal{A}, I, G)$, where $\mathcal{P}$ is a finite set of predicates, $\mathcal{A}$ is a finite set of actions, I is the initial state, and G is the goal state. A predicate $p \in \mathcal{P}$ is a boolean proposition describing a specific condition of the world, and a subset of $\mathcal{P}$ represents a state s at some stage of the plan. An action $a \in \mathcal{A}$ is the basic unit that changes states; each action is characterized by two components: its *preconditions*, the predicates that must hold before the action can be applied, and its *effects*, the predicates that hold after the action is applied. A plan is a sequence of actions that transforms I into G . Classical planning problems are typically expressed in the *Planning Domain Definition Language* (PDDL), a declarative language for specifying predicates, actions, and goal conditions.

Cyberattacks have previously been modeled as classical planning problems, but in forms that

are not yet adequate for high-fidelity emulation. Several prior works either lack clear definitions of actions and states [110], [111], [112], [113] or rely on overly abstract or limited action spaces [49], [50], [114], [115], [116], [117] that do not reflect the breadth of real-world adversary behavior [118]. More fundamentally, most existing approaches do not explicitly model how attack actions are causally linked through preconditions and effects, and some [112], [113] do not specify how single-shot planning generalizes to multi-stage APT campaigns in which each stage has distinct initial and goal states. The absence of (i) a well-defined attack-action space, (ii) a principled linking model, and (iii) a multi-stage planning schema leaves a critical gap in the automated planning of actionable and causally coherent cyberattacks. The remainder of § 4.1.4 introduces AURORA’s design choices that close each of these three gaps.

4.1.4 Cyberattack Planning

This section introduces two core concepts, *Attack Actions* and *Attack Action Linking Model (AALM)*. Attack actions provide modular and structured representations of actionable steps in cyberattacks. The attack action linking model defines the relationships between these actions and connects them into causality-preserved attack chains.

4.1.4.1 Attack Actions

In classical planning, an action is the atomic unit in a plan. Similarly, we define an attack action as an atomic operation in cyberattack emulations.

Definition 1 (Attack Action). *An attack action is the smallest unit that is considered to form an attack plan and an atomic operation to execute in a multi-step attack.*

The granularity of an attack action depends on how attack tools are invoked in practice. For example, an individual test in Atomic Red Team [119], a module in Metasploit [42], or a single

command in post-exploitation frameworks [120], [121] can each be treated as an attack action. As summarized in Table 4.1, each attack action has several associated attributes. *Name*, *source*, and *description* define the basic information of the action. *Tactics and techniques* map the action to the entries in the MITRE ATT&CK matrix. The *execution* field specifies both concrete execution instructions (e.g., commands or scripts) and the required executor, which denotes the execution context or agent. For example, a malicious PowerShell script uses `PowerShell` as its executor, whereas commands provided by Meterpreter [122] require an active Meterpreter session on the target system and therefore designate `Meterpreter` as their executor. The *precondition* and *effect* attributes are essential for linking actions together. Therefore, they are discussed in detail in the next section on the attack action linking model.

Table 4.1: Attributes of attack actions

Field	Description
UUID	A universally unique identifier.
Name	A short name of the attack action.
Description	A brief description of the attack action.
Source	The source tool of the attack action.
Tactics	MITRE ATT&CK tactics.
Technique	MITRE ATT&CK technique.
Execution	The concrete executor and instructions for execution.
Preconditions	Conditions that must be satisfied for action execution.
Effects	Conditions that will be satisfied after action execution.

The concept of attack actions is related to, but distinct from, MITRE ATT&CK procedures [123]. While procedures provide descriptions of how a technique may be implemented in practice, they are primarily intended for knowledge sharing and threat intelligence documentation. As a result, procedures are expressed in unstructured natural language and do not explicitly define execution context, preconditions, or effects. In contrast, attack actions are designed as executable and plannable primitives. Each attack action explicitly specifies its executor, concrete instruc-

tions, preconditions, and effects, which enables attack actions to be automatically composed into causally coherent, actionable attack chains. For example, a MITRE procedure may state that attackers dump credentials from LSASS using tools such as Mimikatz. However, this description does not explicitly encode the prerequisite of administrative privileges or the resulting availability of credential artifacts. In our model, credential dumping is represented as an attack action whose execution requires administrator access and whose effect is the acquisition of credentials.

4.1.4.2 Attack Action Linking Model

As introduced in §4.1.3, preconditions and effects are used to connect actions in classical planning. Preconditions represent the requirements that must be met before executing an attack action. For example, executing Sliver commands requires an active session on the victim machine, and data exfiltration requires the data to be collected. Effects are the resulting conditions after executing an attack action. An action is included in an attack chain only if 1) all its preconditions are fulfilled, and 2) its effects help satisfy the preconditions of the following actions or achieve specific attack goals. However, the term *condition* is broad, subjective, and ambiguous. In this paper, we propose the Attack Action Linking Model (AALM), a standard, universal, and succinct framework, to describe the preconditions and effects from the perspective of cyberattack emulation.

Definition 2 (Attack Action Linking Model). *The attack action linking model is an extensible set of PDDL predicates used to describe the preconditions and effects of an attack action.*

We design the predicate set of AALM from nine dimensions: environments, executors, payloads, files, processes, users, information, data, and techniques. Due to space limits, the full list is available in our open-source repository.

- *Environment* includes the predicates related to the operating systems, vulnerabilities, software, and network topology of the attack target. To standardize the predicate format across numerous

vulnerabilities and software types, we use Common Vulnerabilities and Exposures (CVE) [124] to identify vulnerabilities and Common Platform Enumeration (CPE) [125] to represent software.

- *Executor* includes predicates about the executor type and states. Executor type encompasses common executors derived from popular attack frameworks and C2 tools [42], [43], [119]. Executor states contains predicates about privilege levels, persistence status, and so on.
- *Payload* includes predicates describing payload types, formats, and operations. The payload types and formats are derived from commonly used payload generation tools [126]. Payload operation predicates cover the conditions about loading, executing, and setting handlers.
- *File* includes predicates related to file types, states, and operations. To ensure AALM covers the file types commonly involved in cyberattacks, we crawled documentation of popular open-source attack tools [42], [43], [119] and cybersecurity websites [127], [128] to extract a representative set of file types. It also covers similar predicates for directories and Windows registry entries.
- *Information* includes predicates representing information involved in cyberattacks. Given the diversity of information types in cyberattacks, we organize them into 39 categories based on the technique types defined under the MITRE discovery tactic.
- *Data* includes the predicates representing data types and data states. We categorize ten data types according to the technique taxonomy of the MITRE collection tactics. Data state predicates include the storage, transmission, and relay of data during an attack.
- *User* includes the predicates representing user states, types, username, credentials, and privilege levels of users.

- *Process* includes the predicates representing process states and process information such as process ID, name, and privilege level.
- *Technique* includes the predicates that reflect the intent or consequence of an action. For example, although deleting backup files and deleting log files both result in `(file-deleted)`, their purposes are different. The former aims to inhibit system recovery, while the latter serves to remove indicators of compromise. Therefore, we design technique-level predicates such as `(inhibit-system-recovery)` and `(indicator-removal)` to show the difference of the effects.

Figure 4.2 shows an attack chain connected by predicates in AALM. Given the complexity and evolving nature of cyberattacks, AALM is built with extensibility. New predicates and actions can be incrementally added, similar to the ongoing expansion of the MITRE ATT&CK framework. AALM allows users to define custom actions along with their preconditions and effects using PDDL syntax. AALM follows a monotonic extension principle: newly introduced predicates are incorporated through additional action preconditions and effects. Existing attack chains remain unchanged and valid under the extended model. For example, a user can introduce a predicate `(edr-service-disabled ?h)` as the effect of an evasion action and as the precondition for high-risk payload execution, explicitly modeling the causal relationship between disabling a specific endpoint protection and enabling stealthy execution. As another example, when certain attack actions are only feasible with specific tools, users can introduce tool-specific predicates (e.g., a dedicated executor-type predicate) as the preconditions of those specialized actions. As the first work to propose a standardized, classical linking model for automated attack emulation, we envision its coverage and expressiveness to grow through collaboration with the broader security community.

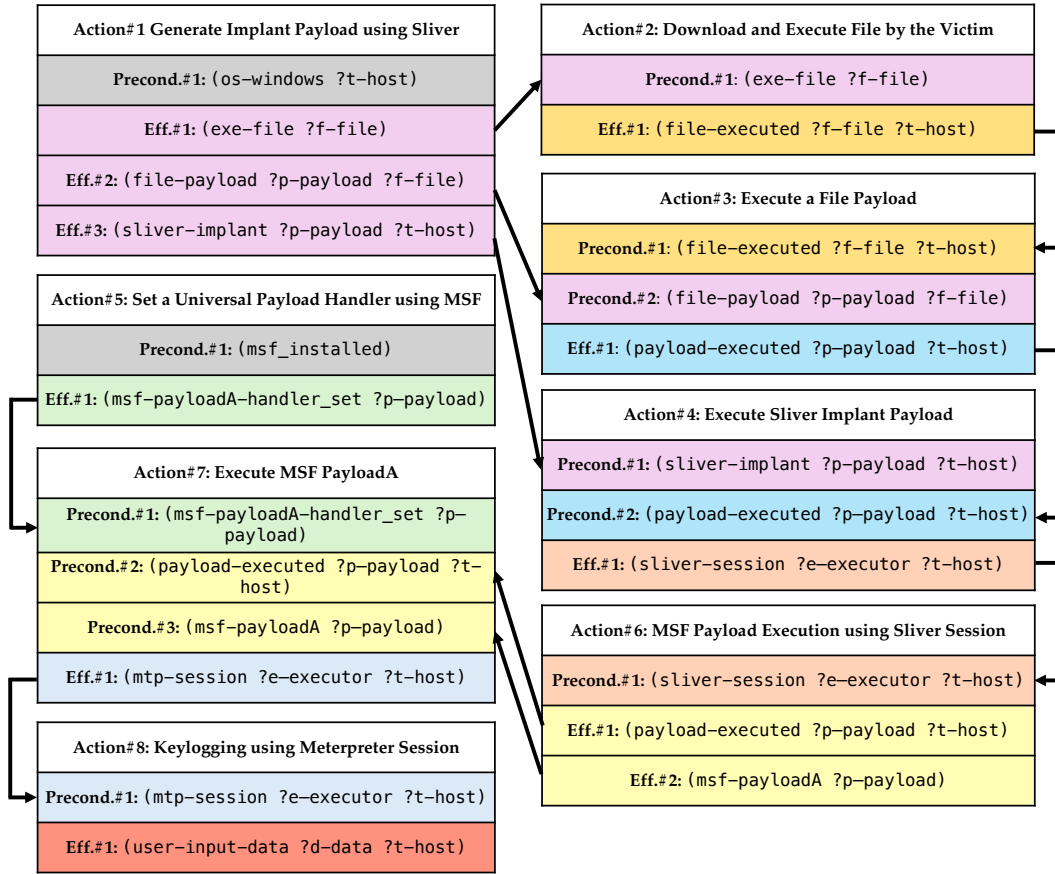


Figure 4.2: An attack chain example connected by AALM.

4.1.5 System Design

Based on the attack actions and AALM, we design AURORA, an automated, customizable, causality-preserving cyberattack emulation system. As shown in Figure 4.3, AURORA encompasses three stages: ①*Attack Action Formalization*, ②*Attack Planning*, and ③*Attack Execution*. AURORA first builds attack actions from existing tools. During attack planning, AURORA considers available attack actions, emulation environments, and CTI reports to formalize the planning problem in PDDL. The classical planner then chains the actions to a plan. Finally, a Python script is generated to execute the attack plan in the emulation environment.

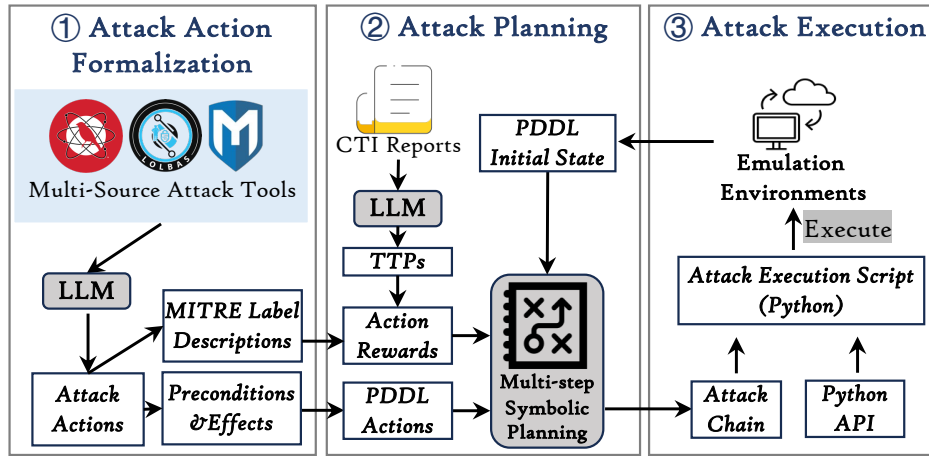


Figure 4.3: System overview.

4.1.5.1 Attack Action Formalization

The emulated attacks leverage existing tools rather than zero-day exploits, ensuring that AURORA evaluates defense systems only against known attack techniques. AURORA reviews documentation of attack tools and converts them into structured attack actions defined in §4.1.4.2. However, analyzing heterogeneous documentation is challenging. Although some simple features, such as name and description, are provided in the documentation and can be extracted with regular expressions, identifying preconditions and effects requires a deeper semantic understanding and substantial domain expertise in cybersecurity.

Recent studies show that LLMs demonstrate strong capabilities in both text comprehension and domain-specific reasoning [129], [130]. AURORA leverages LLMs to formalize third-party tools into attack actions. As shown in Figure 4.4, we begin by categorizing all tools into 14 groups based on their corresponding MITRE tactics. For each tactic, we design a prompt for LLM to analyze the preconditions and effects. The prompt consists of five components: a task description, a step-by-step thinking process (a series of questions), a set of candidate predicates from the AALM,

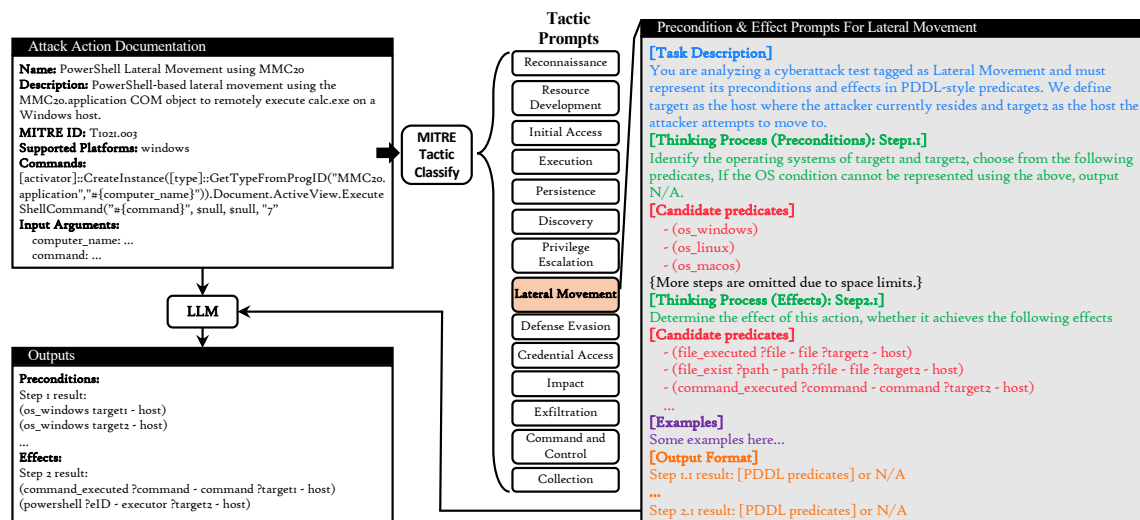


Figure 4.4: An example showing how AURORA analyzes preconditions and effects of an attack action using AALM.

illustrative examples of correct outputs and common mistakes, and formatting requirements for the output. For each question in the thinking process, the LLM is restricted to selecting only from the given predicate set. If no suitable predicate is available in AALM, the LLM is instructed to output *N/A*, and the action is excluded from the attack emulation. This approach minimizes hallucination and ensures that only actions compliant with AALM specifications are retained for subsequent classical planning.

As an example, consider tools used for payload generation, which fall under the *Resource Development* tactic, including widely used tools such as Msfvenom, Sliver, and Empire. The corresponding prompt first outlines the function of these tools and the analysis task. It then details the analysis process. For preconditions, the LLM is asked to identify the target OS and architecture of the payload and selects matching predicates from AALM; if none apply, it returns *N/A*. For effects, the thinking process begins by checking whether the payload type matches any predefined payload types in AALM; if not, *N/A* is returned. Next, it determines the payload format (file, shellcode,

one-liner command, raw packet). If the payload is a file, its file type is also identified. If any of these steps yield N/A, AURORA excludes the action from the action space. In our experiments, AURORA uses GPT-o3 (o3-2025-04-16) as the underlying LLM for predicate extraction. Prompts were iteratively refined during development using a small validation set for each tool category to ensure syntactic correctness. Once finalized, the prompts were fixed and not modified during evaluation.

Experimental results show that AALM and our prompt engineering can cover the majority of attack actions and formalize more than 5000 actions. AALM also addresses the problem of relying on MITRE tactic labels in attack emulation. For instance, certain tests in Atomic Red Team are labeled as *Lateral Movement*, yet their actual effect does not achieve remote file transfer or code execution. Therefore, AURORA ensures these actions are not included in the emulation solely based on their MITRE labels. In §4.1.6.2, we show how this helps avoid invalid attack emulation compared to existing work.

4.1.5.2 Attack Planning

AURORA uses classical planning to chain the attack actions. It represents attack actions and emulation environments in PDDL, which is then consumed by an off-the-shelf classical planner to generate execution plans. Moreover, we designed a reward function mechanism to customize attack emulation based on CTI reports and a multi-step planning process to generate multi-stage attack emulation plans.

Representing Emulation Environments in PDDL The emulation environment, i.e., the target machine(s) of the attack, determines which actions are feasible. Therefore, the starting state I in attack planning should reflect the initial state of the emulation environment. Prior to emulation,

basic information about the target machine(s) is required, including operating systems, known vulnerabilities, installed software, and domain or network configurations. This information can be easily obtained by running automated asset-inventory tools such as OpenSCAP, OpenVAS, and osquery [105], [106], [107]. AURORA supports parsing the outputs of these tools and automatically map them to the environment predicates in AALM. In §4.1.6.4, we will show how AURORA generates customized attacks based on different emulation environments.

Reward Function for CTI-based Customization AURORA aims to emulate customized cyber-attacks tailored to the characteristics of specific threat groups. To achieve this, AURORA leverages CTI reports, which document the behaviors of real attackers. We design the reward function to generate attack chains that closely resemble CTI reports. We first extract TTPs from the reports. Despite various approaches in analyzing CTI reports [40], [131], [132], [133], [134], [135], our experiment in § C.2 shows that LLMs surpass these methods in terms of accuracy. Therefore, we employ off-the-shelf LLMs to extract TTPs from the reports. To mitigate the hallucination issue and standardize the output, we include MITRE ATT&CK TTP definitions and an output template in the prompt. The improvement of CTI report analysis is not the focus of this paper.

AURORA customizes attack emulation based on CTI reports by generating attack chains that best match the TTPs extracted from the report. To accomplish this, we assign a reward to each attack action. Attack actions whose TTPs appear in the report are assigned higher rewards, while others receive lower rewards. By doing so, maximizing the total reward of an attack chain directly corresponds to maximizing its similarity to the CTI report. To compute the reward of an action, we first check whether its MITRE label appears in the TTP set extracted from the report. If not, the reward is set to zero. Otherwise, the reward is computed as the cosine similarity between the embeddings of the action description and that of the corresponding TTP description. Therefore,

actions with more similar semantics will receive higher rewards. For example, consider two privilege escalation actions: bypassing User Account Control (UAC) and process injection. While both actions can elevate the attacker’s privileges, if the report contains some phrases such as *abusing the UAC*, the former action will have a higher reward. If two attack chains are generated in the same emulation environment, the one with the higher reward is selected. Please note that we deliberately chose cosine similarity for its simplicity, efficiency, and ease of integration into planning. More sophisticated semantic alignment functions (e.g., cross-encoders or entailment-based models) can be easily plugged in. The evaluation in §4.1.6.4 shows that the cosine-based rewards yield over 30% improvement in CTI-level customization.

Multi-step Classical Planning Setting the goal states of the planning problem remains challenging for the multi-step attacks. On one hand, setting a single predicate as the goal state results in short chains that achieve only one attack objective. However, if we combine all desired predicates into a single goal state, any inaccessible predicate causes the entire planning process to fail. To address this, we designed a multi-step planning procedure. Specifically, we created 12 pseudo actions, which correspond to 12 high-level attack stages defined by MITRE tactics. The preconditions of a pseudo action are predicates that indicate the success of a given stage, connected by a logical *or*. The effect of each pseudo action is a dummy predicate that serves as the planning goal for that stage. The classical planner then searches for a feasible attack chain that can achieve this goal. When multiple chains exist, AURORA selects the one with the highest reward. Pseudo action provides a concise and clear way to organize attack stages in classical planning. The multi-step planning is then conducted according to the order of the APT lifecycle: initial access, execution, discovery, persistence, privilege escalation, defense evasion, credential access, lateral movement, collection, command and control, exfiltration, and impact. The goal state of each step is the dummy

effect predicate of the pseudo action. The initial state before the first attack stage consists of the emulation environment predicates. After completing a stage, AURORA collects the effect predicates of the newly selected actions and adds them to the initial state. With the updated initial state, AURORA proceeds to plan the attack path for the next stage. For instance, after completing the execution stage, we obtain `(sliver_session?e-executor?t-host)` as one of the effect predicates. This predicate is then added to the initial state for planning the next stage. Note that the attack stage order is customizable. For instance, sabotage-focused attacks may skip data collection and exfiltration stages. Users can also reorder certain stages, as the predefined planning sequence does not constrain the causal dependencies between actions. For example, if persistence requires privilege escalation as a precondition, AURORA will automatically place privilege escalation actions earlier, even if their stage comes later in the original planning sequence. Multi-step planning of AURORA maximizes attack tactic coverage and stays robust when some stages are infeasible.

Handling Execution Failures: AURORA explicitly handles execution-time failures during attack emulation. When an action fails to execute due to unexpected runtime errors, AURORA removes the failed action from the action space and triggers re-planning for the current attack stage. Since attack campaigns are decomposed into 12 stages, execution failures are isolated within the current stage and do not affect previously completed stages. In other words, re-planning is restricted to the stage at which the failure occurs. For example, if a PowerShell-based data collection action fails during execution, AURORA excludes this action from the action space and re-plans the data collection stage using the remaining candidate actions. If no alternative action sequence can achieve the objective of the current stage, AURORA skips the stage and proceeds to plan subsequent stages. This design ensures that execution failures do not halt the entire attack emulation process and allows AURORA to adaptively recover from runtime errors without requiring global rollback.

4.1.5.3 *Attack Execution*

The final step is to execute the generated attacks in the emulation environment. Different from existing work [48], [49], [115], [117], AURORA can generate actionable attack emulation plans since each attack action corresponds to a concrete instruction from a tool. To further enhance automation, we developed a Python function API covering all tools integrated in AURORA, including running Metasploit modules, executing commands in the Sliver console, executing commands on the attacker machine, and simulating simple user actions. Based on the action sequence produced by the planner, AURORA automatically generates a Python script using this API. This allows an entire attack chain to be executed by simply running a script. In §4.1.6.6, we collected system logs when executing emulated attacks and used them to test three advanced intrusion detection systems.

4.1.6 **Evaluation**

We evaluate the performance of AURORA along five key dimensions. Specifically, we assess whether AURORA can (1) successfully emulate attacks, (2) generate causally coherent attack chains, (3) customize attack emulation based on target environments and CTI reports, (4) improve the scale and diversity of attack emulation, and (5) support the evaluation and benchmarking of security defenses. We present a case study to further illustrate how AURORA helps generate a complete, causality-preserved attack chain. More evaluations, including the costs of AURORA and the accuracy of TTP extraction, are presented in Appendix § C.2.

4.1.6.1 *Evaluation Setup*

Attack Tools. AURORA converts existing attack tools to attack actions. In this paper, we utilize Atomic Red Team [91], Metasploit [42], Meterpreter [122], Sliver [120], and MSFvenom [126]. They represent typical cyberattack tooling: Metasploit and MSFvenom cover common pre-exploitation

tasks, Atomic Red Team supplies the post-exploitation library, and Meterpreter and Sliver are mature real-world tools. The above attack tools provide AURORA with 5,555 attack actions, more than many other automated cyberattack systems [48], [136].

CTI Reports. We collected 250 CTI reports from multiple sources [40], [137], [138]. The CTI report dataset used in our evaluation will be open-sourced.

Implementations. We implemented AURORA with 4K LoC in Python. We used the GPT models from OpenAI and leveraged Fast Downward [139] to solve classical planning problems. We set up the attack emulation environment infrastructure using Oracle VirtualBox, creating a small-scale LAN with 15 different hosts. We release the code, datasets, and results to the public to facilitate further research.

Baselines. The baselines can be divided into four categories. The first category is human-crafted cyberattack emulation datasets, including MITRE Evaluation [47], and APT Attack Simulation [140], a popular open-source attack sets. The second category consists of procedural attack orchestration frameworks, including Caldera [44], Attack Range [46], and PurpleSharp [45]. The third category is the single-point testing tools, including Atomic Red Team [119], Metasploit [42], and Viper [127]. The last category is the abstract attack modeling studies [48], [49], [115], [117]. Specifically, when evaluating the success rate and causality preservation of attacks, we compared AURORA against PurpleSharp, Caldera, and the MITRE Evaluation because they provide modularized attacks, which are well-structured and suitable for quantitative analysis.

4.1.6.2 Success Rate of Emulated Attacks

We first evaluated whether the attacks could successfully achieve malicious objectives. We examined the main objectives of cyberattacks [141], [142], including espionage, data theft, sabotage, and gaining a long-term strategic advantage, and mapped them to six MITRE tactics: persistence,

privilege escalation, collection, exfiltration, impact, and lateral movement. Note that the remaining tactics, such as initial access and execution, primarily correspond to intermediate stages of an attack. We then evaluated how many emulated attacks successfully accomplished these tactics. Specifically, for privilege escalation, we assessed whether the emulated attack gained elevated privileges (e.g., root or sudo on Unix systems, or Administrator and System on Windows). For persistence, we evaluated whether the attack established persistent access by maintaining a C2 agent or obtaining valid credentials. For collection, we verified whether the attack acquired user data. For exfiltration, we checked whether the attack transmitted collected data back to the attacker. For lateral movement, we measured whether the attack successfully moved laterally to another machine by executing a file, command, or script on a second host. For impact, we assessed whether the attack caused any destructive effects, including deleting user files, removing access, stopping services, or shutting down the system. The results in Table 4.2 show that AURORA generates almost 10 times more attacks than the baseline and achieves the highest success rate across all tactics. We analyzed the attack chains and identified two main reasons for the superiority of AURORA. First, AURORA uses AALM to link attack actions, rather than relying on MITRE labels to arrange actions. This avoids the invalid step due to the mismatch between MITRE labels and the actual action effects. For example, in MITRE’s Fin7 emulation [143], *Dump SAM via Mimikatz(T1003.002)* was used for privilege escalation. However, executing this step does not directly provide the attacker with administrative privileges. It merely dumps SAM credential data; attackers must take additional steps to extract plaintext credentials or hashes from it and then leverage those to gain high-privilege access. On the contrary, AURORA ensures the high-privilege access after the privilege escalation step because AALM provides more concrete and accurate definitions of effects, closely aligned with actual attack emulation, which enhances the success rate and realism of emulations. Second, multi-step planning allows AURORA to integrate multiple malicious behaviors

into a single attack chain. In contrast, most attack chains from tools like PurpleSharp and Caldera focus on just one tactic. With multi-stage planning, users can freely determine which tactics appear in the chain, enabling broader coverage of malicious behaviors within a single emulation scenario.

Table 4.2: Comparison of success rates(with 95% confidence intervals).

	PurpleSharp	Caldera	MITRE Eval.	AURORA
# Total Chains	23	28	8	250
Persistence	8.7% (1.5,27.0)	0 (0.0,12.3)	87.5% (47.4,99.7)	95.6% (92.3,97.8)
Privilege Esc.	4.4% (0.1,22.0)	3.6% (0.1,18.4)	50.0% (15.7,84.3)	100.0% (98.5,100.0)
Collection	0 (0.0,14.8)	39.3% (21.5,59.4)	62.5% (24.5,91.5)	98.0% (95.4,99.4)
Exfiltration	0 (0.0,14.8)	28.6% (13.2,48.7)	37.5% (8.5,75.5)	100.0% (98.5,100.0)
Impact	0 (0.0,14.8)	7.1% (0.9,23.5)	37.5% (8.5,75.5)	59.2% (52.8,65.4)
Lateral Move.	30.4% (13.2,52.9)	28.6% (13.2,48.7)	87.5% (47.4,99.7)	100.0% (98.5,100.0)

4.1.6.3 Causality in Emulated Attacks

AURORA aims to emulate high-fidelity attacks that are not only successful, but also preserve causal relationships between actions. For example, it is not sufficient to merely emulate successful data collection and exfiltration in isolation. Additionally, what is exfiltrated should be exactly what was collected. Similarly, the process that gains elevated privileges should be the one used in subsequent actions. As illustrated in Figure4.1(a), although all four steps are completed successfully, they are not causally linked. To quantify the causality in an attack chain, we assessed logical connections between steps from three dimensions: 1) *Executor Establishment*: Whether preceding steps establish the executor to run the current step. 2) *Argument Sharing*: Whether the step shares arguments (e.g., hostnames, file paths) with other steps in the chain. 3) *Artifact Interaction*: Whether the step utilizes or affects system artifacts (e.g., files, processes) that are involved in other steps. Correspondingly, we define three metrics: *Executor Traceability Rate*, *Shared Arguments Rate*, and *Shared Artifacts Rate*. Each metric measures the proportion of steps in a chain satisfying

the corresponding criterion. In addition, we evaluated the average length of the attack chains.

We verified attack chains from the baselines and AURORA, with the results summarized in Table 4.3. Our analysis highlights three key findings. First, AURORA and the MITRE Evaluations provide significantly longer attack chains than PurpleSharp and Caldera. Longer chains can better reflect the nature of multi-step APT attacks and offer more opportunities for detection. Please note that AURORA can generate even longer chains if increasing the number of actions per stage during the multi-step planning. Second, AURORA and MITRE Evaluations can construct full-lifecycle attack chains. We define a full-lifecycle attack as one commencing with initial access or execution and including at least one remaining downstream tactic. In contrast, the chains from PurpleSharp and Caldera typically focus on a single tactic. Finally, in terms of causality preserving, AURORA significantly outperforms PurpleSharp and Caldera, achieving comparable performance to the expert-crafted MITRE chains. The executor traceability for both PurpleSharp and Caldera is zero. This is an expected outcome of their common architecture: a sequence of isolated scripts orchestrated by a central control agent, without modeling how that agent is initially established on the victim machine. In addition, the high rates of shared arguments and artifacts in AURORA’s chains indicate strong internal connectivity.

Table 4.3: Comparison of causality preservation.

	Avg. Steps	Full Lifecycle	Causality-preservation		
			Executor Trac.	Shared Args	Shared Artifact
PurpleSharp	3.2	✗	0	0	20.3%
Caldera	5.4	✗	0	15.1%	54.6%
MITRE Eval.	53.6	✓	94.4%	16.7%	93.4%
AURORA	55.2	✓	100%	40.0%	99.4%

To further analyze how AALM helps preserve causality, we examined the types of shared

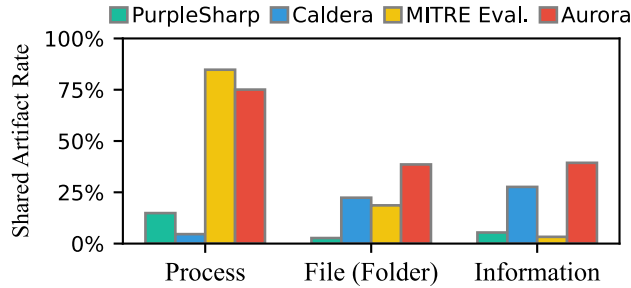


Figure 4.5: Comparison of shared artifact rates.

artifacts that connect steps in the emulated attack chains. As shown in Figure 4.5, chains generated by AURORA exhibit a higher proportion of steps linked via shared files and information, while the MITRE chains show more connections through shared processes. This is because AURORA models pre-foothold stages such as payload generation and initial access, where causal links are typically established via artifacts like files and target information. In contrast, MITRE chains focus more on the stages after a foothold has already been established, where causality is more likely to manifest through process-level interactions. Next, we investigated the contribution of different AALM predicate categories in preserving causality, since the linking relationship expressed by some categories in AALM (e.g., those for operating systems, CVEs, or exploit-payload mappings) has been considered in prior attack planning work [110], [111], [144], [145]. Our findings show that the novel predicate categories in AALM, such as those related to executors, payloads, files, information, users, and data, are crucial: they constitute 70.1% of all predicate connections in the generated attack chains.

4.1.6.4 Customization of Attack Emulation

Customization based on CTI Reports We designed the reward function to prioritize the attack actions mentioned in the CTI reports during planning to achieve CTI-level customization. For this evaluation, we selected 50 CTI reports and generated two attack chains for each: a report-

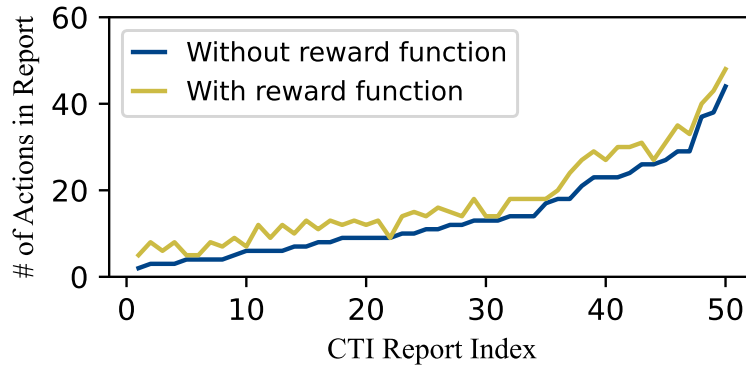


Figure 4.6: Fidelity of emulated attack chains to source CTI reports.

guided chain, using the reward functions during planning, and a report-agnostic chain created from the same action space without CTI guidance. We then compare the number of CTI-mentioned techniques in the generated attacks. Across 50 CTI reports, incorporating the reward function during planning increases the number of CTI-mentioned techniques from 13.02 to 16.98 on average (+30.5%). A paired t-test shows the improvement is statistically significant.

We also present a case study, a CISA report on Phobos Ransomware, to show how the reward function mechanism guides the planner toward report-aligned attacks to achieve report-level customization. In the initial access stage, the report notes that “...*threat actors send spoofed email attachments that are embedded with hidden payloads such as SmokeLoader...*”. Therefore, AURORA selects the action of delivering payloads via email attachments (T1566.001) as malicious files (T1204.002) over other alternatives like using Metasploit exploit modules or supply chain attacks. For the persistence, the report states that “...*Phobos has also been observed using Windows Startup folders and Run Registry Keys to maintain persistence...*”. Accordingly, AURORA favors the action that modifies the user shell folder startup value (T1547.001) over other options, such as creating a new account (T1136) or abusing boot or login initialization scripts (T1037). Please note that AURORA balances CTI fidelity with actionability and causality. When a tool mentioned

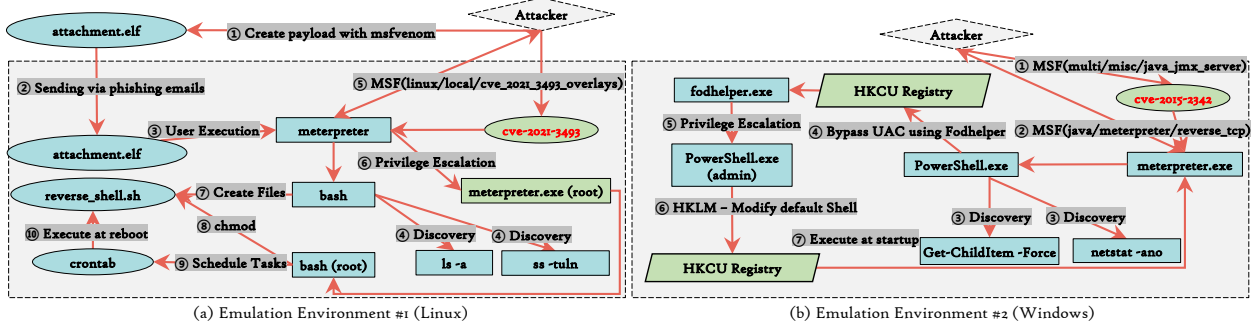


Figure 4.7: Customized attacks for different environments.

in the report, like SmokeLoader, is unavailable, it substitutes a close alternative (e.g., Sliver) and arranges the subsequent actions automatically to preserve the attack chain’s logic.

Customization based on Emulation Environments We demonstrate AURORA’s capability of tailoring attacks to specific user environments through a case study where AURORA generates attack plans for two distinct environments: a modern Linux system (CVE-2021-3493) and an older Windows system (CVE-2015-2342). For both environments, the objective was to create a five-stage attack plan covering initial access, execution, discovery, privilege escalation, and persistence. By simply modifying the initial state predicates in the respective PDDL problem files, AURORA’s classical planner automatically selected compatible attack actions from our linking model to generate a unique chain for each environment. Figure 4.7 illustrates these results, showing two attack chains that achieve the same high-level goals using entirely different, environment-specific implementations. Importantly, AURORA does not fail to generate attack chains due to the absence of environment-specific actions. When certain specialized actions are unavailable, AURORA can always fall back to composing attack chains using more generic and widely applicable actions, ensuring that valid multi-stage attack plans can still be generated for a given environment.

Table 4.4: Comparison of cyberattack dataset scale and TTP diversity.

	Attack Chains	Attack Actions	MITRE Techniques	MITRE Tactics
APT Simulation	13	N/A ¹	N/A ¹	N/A ¹
MITRE Eval.	16	619	152	12
PurpleSharp	25	181	30	8
Caldera	28	1530	309	13
Atomic Red Team	N/A ²	1568	309	13
Viper	N/A ²	97	N/A ³	10
SVED	N/A ²	4000+	262	12
Lore	N/A ³	N/A ³	68	10
ChainReactor	N/A ³	31	N/A ³	N/A ³
AURORA	250	5555	327	14

¹ No modularized steps that can be mapped to MITRE ATT&CK TTPs.

² Cannot construct multi-step cyberattacks.

³ Not clearly reported.

4.1.6.5 Diversity of Emulated Cyberattacks

This section evaluates the scale and diversity of the attack chains generated by AURORA. We assess scale by the total number of generated attack chains and diversity by the number of unique attack actions, MITRE ATT&CK tactics, and techniques covered. Our comparison includes a wide range of baselines. As detailed in Table 4.4, AURORA surpasses all baselines in both scale and diversity. In terms of scale, AURORA generates a significantly higher number of unique attack chains. This is a direct result of our approach, which defines and links modular attack actions, allowing for the creation of numerous valid chains through meaningful permutations. In terms of diversity, AURORA covers the largest number of unique actions, techniques, and tactics. This breadth is achieved through its ability to integrate and leverage a wide array of third-party attack tools from different sources.

Table 4.5: Comparison of detection performance against attacks generated by AURORA (with 95% bootstrap confidence intervals).

	NODOZE	PROVDETECTOR	FLASH
True Positive Rate	18.92% (10.04,31.71)	48.16% (29.61,60.14)	9.67% (2.60,17.99)
False Positive Rate	2.35% (1.19,4.18)	1.72% (0.92,2.58)	0.06% (0.01,0.12)

4.1.6.6 Evaluating Existing Detection Systems

In this section, we used 250 attacks generated by AURORA to evaluate three advanced, open-source attack detectors, namely, NODOZE [29], PROVDETECTOR [146], and FLASH [13]. During these emulations, system traces were collected via ETW and converted into provenance graphs. We established ground truth by labeling system entities involved in the attacks. For training data, we simulated five hours of benign user activity. All three detectors perform anomaly detection on provenance graphs, but use different techniques. We evaluated the true positive rates (TPR) and false positive rates (FPR). For each metric, we first apply the Friedman test [147] to assess overall differences among detectors. When significant, we conduct post-hoc pairwise comparisons using the Wilcoxon signed-rank test with Holm correction ($\alpha = 0.05$) [148], [149].

Table 4.5 shows the detection results. The Friedman test indicates significant overall differences among detectors in both TPR ($p = 0.0046$) and FPR ($p = 0.0011$). For TPR, our testing shows that PROVDETECTOR and NODOZE significantly outperform FLASH. PROVDETECTOR also achieves higher TPR than NODOZE, although the difference is not statistically significant. For FPR, post-hoc analysis shows that FLASH achieves a significantly lower false positive rate than both PROVDETECTOR and NODOZE, while the latter two are statistically indistinguishable. In general, we observed a significant performance drop on our dataset compared to those used in prior work, likely due to the greater sophistication of attacks generated by AURORA. We identified two primary reasons for this decline. First, FLASH’s GNN has only two hidden layers, limiting its

receptive field to two-hop neighbors. This may be sufficient for short attack paths, but inadequate for longer, causality-preserved attack chains from AURORA. Capturing such paths would require a deeper GNN architecture. Similarly, PROVDETECTOR selects the K rarest paths for anomaly detection. However, since AURORA generates longer attack chains with more steps, the original K setting is insufficient to cover all attack-related paths, leading to missed detections. Second, AURORA heavily utilizes living-off-the-land (LotL) techniques, which blend in with benign behavior and may further reduce the effectiveness of anomaly-based detectors, particularly NODOZE, which uses event frequency for detection. Although a comprehensive benchmarking study lies outside the scope of this paper, this experiment demonstrates the necessity of building more frequently updated attack datasets.

4.1.6.7 Case Study

We present a case study on a recent real-world APT group active since 2024, Silver Fox [150], [151], to further illustrate how AURORA and AALM help generate a complete, causality-preserved attack chain. At present, our understanding of the group remains limited. Only a few isolated attack techniques are identified, which are insufficient for constructing attack emulations to support testing or data collection. In collaboration with a red team, we obtained three steps used by the Silver Fox attack, and manually assigned preconditions and effects according to AALM. These three steps focus solely on the tactics of execution, persistence, and impact. Without AALM, it would be hard and time-consuming to integrate them with existing actions to build a complete APT attack emulation.

The attack chain shown in Fig. 4.8 contains the three actions from Silver Fox and nine actions added by AURORA to form a complete, causality-preserved attack. The first action shows Silver Fox is abusing legitimate, digitally signed programs to execute malicious code via techniques like

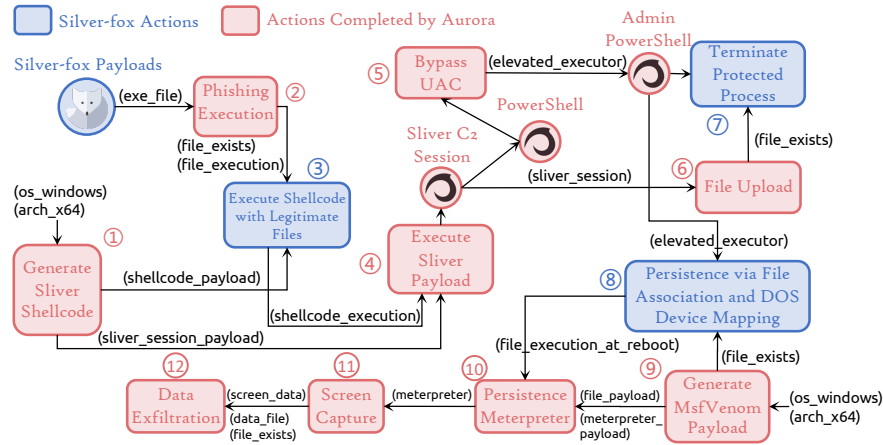


Figure 4.8: The case study showing how AURORA constructs a complete attack chain for three “Silver Fox” attack actions.

DLL hijacking. With `(file_executed)` and `(shellcode_payload)` as preconditions and `(shellcode_executed)` as the effect, AURORA automatically built this action into the attack chain by linking it with phishing attachment execution and Sliver shellcode generation. The second action exploits a known vulnerability in a legitimate, signed driver (`wsftprm.sys`) to gain kernel-level privileges. This action requires the vulnerable driver file and the attacker executable to be placed `(file_exists)`, and a PowerShell with administrative privileges (`powershell`) and `(elevated_executor)`. AURORA integrates it into a complete attack chain that uses Sliver to upload malicious files and perform a UAC bypass to obtain administrative privileges. The third action of Silver Fox establishes stealthy persistence by chaining several evasive maneuvers. The precondition requires an administrative PowerShell environment to execute the script, represented by `(powershell)` and `(elevated_executor)`. The effect of the action is `(file_execution_at_reboot)`. The administrative PowerShell obtained for the previous action is reused. This action is then used to establish a persistent Meterpreter session for subsequent data exfiltration.

4.1.6.8 Ablation Study

We evaluate the effectiveness of our prompt design for predicate extraction by comparing LLM outputs against manually curated ground truth. We perform stratified random sampling of 150 actions across 14 tactic groups. Two authors independently annotate each action’s preconditions and effects based on AALM, resolving disagreements through discussion. On this set, AURORA achieves high accuracy in predicate analysis. Specifically, for 93% of the sampled actions, the LLM correctly identifies preconditions that exactly match the ground truth, and for 86% of the actions, the extracted effects match the ground truth. We further analyze cases where the LLM outputs deviate from human annotations and identify the main sources of errors. First, some errors arise from application-specific dependencies between actions that are not explicitly defined in AALM. For example, certain Meterpreter commands such as Drop Token require a prior Steal Token action, and KeyStrokeDump requires KeyStrokeCapture to be enabled beforehand. Since these dependencies are tool-specific and not encoded in the current AALM predicate set, the LLM cannot reliably infer the correct preconditions and effects. Given the diversity of tools and applications involved in real-world attacks, it is impractical for AALM to predefine predicates for every specific application. Nevertheless, AALM is designed to be extensible: users can introduce new predicates to capture such dependencies, enabling the LLM to recognize and formalize these actions accordingly. Second, inaccuracies occur when a single complex action spans multiple tactics and produces multiple effects. In such cases, the LLM may fail to capture all effects accurately. This limitation is not unique to AALM or LLM-based extraction; similar challenges arise when modeling attack processes using MITRE TTPs, where a single procedure may serve multiple tactical purposes. Importantly, AURORA conservatively excludes actions whose predicates cannot be confidently formalized, preventing such inaccuracies from propagating into downstream classical planning.

We further evaluate the impact of the prompt engineering. In AURORA, we design tactic-specific prompts that explicitly guide the LLM through a reasoning process for identifying preconditions and effects. As an ablation baseline, we remove the prompt and instead provide the LLM with the complete set of predicates in AALM, allowing it to independently decide which predicates apply to a given action. The results show that, compared to the prompt-engineered version, the ablated setting produces inaccurate preconditions and effects for 58% of the evaluated actions, indicating that the LLM fails to exhaustively identify relevant conditions or effects when not explicitly guided by the prompt.

4.1.7 Discussion and Limitations

More Advanced Attack Linking with LLMs. We leveraged LLMs to analyze the preconditions and effects of attack actions based on raw documentation and AALM. We applied prompt engineering, injecting the step-by-step thinking process, illustrative examples, and candidate predicates into the prompt. A key future research direction is enabling LLMs to link actions automatically.

Apply AALM in Automated Penetration Testing. AALM links attack actions to enable classical planning for attack automation; the tools, actions, and predicates in AURORA are reusable for pentesting. However, to realize automated penetration testing with AALM, several additional challenges must be addressed, such as incomplete environment knowledge, non-deterministic effects, unmodeled predicates, and the need to convert observed information into PDDL format.

4.1.8 Section Summary

In this paper, we define attack actions and the attack action linking model to integrate heterogeneous attack tools into causality-preserved cyberattacks. We then introduce AURORA, the first automated, customizable, and causality-preserving cyberattack emulation system with LLM-powered

classical planning. The evaluation demonstrates that AURORA can emulate customized, causality-preserved attack chains with high scalability and diversity.

4.2 Building Next-Generation Benchmark Datasets for Provenance-Based Intrusion Detection

4.2.1 Motivation: The Dataset Gap in PIDS Research

The past decade has witnessed joint efforts on research in provenance-based intrusion detection, which revealed the growing importance of fine-grained system activity logs for understanding complex attack behaviors. However, the work is hindered by the lack of up-to-date and high-quality datasets. Existing benchmarks, such as DARPA Transparent Computing [55] [54] and OpTC [56], ATLAS [23] and ATLASv2 [57], exhibit several shortcomings. First, some datasets are outdated and mainly record attack scenarios that are no longer representative of contemporary threat landscapes. Second, most of them provide limited information about both attack and expected behaviors, and lack fine-grained ground truth labeling. As a result, it is difficult to conduct a comprehensive analysis and unbiased evaluation of provenance-based intrusion detection systems (PIDS) using these benchmarks.

Additionally, recent studies have pointed out that many ML and GNN-based approaches tend to overfit [103], achieving high performance on these restricted benchmarks but failing to generalize to new or more complex attack scenarios. This overfitting is primarily due to limited scale, incomplete attack coverage, and insufficient benign activity representation in existing datasets. These limitations motivate the need for a modern, high-quality, and comprehensive benchmark that captures diverse attack behaviors and extensive benign activity, enabling more reliable training and evaluation of provenance-based intrusion detection methods.

Constructing a high-quality dataset for PIDS involves several challenges. First, an effective

dataset must incorporate up-to-date and realistic attack scenarios, as well as high-fidelity normal user behaviors. This requirement raises two non-trivial issues: how to systematically acquire evolving attack techniques and security knowledge and translate them into attack traces in the dataset, and how to collect or synthesize normal system activities that closely resemble real system behaviors across diverse scenarios. Second, a high-quality dataset should provide sufficient contextual details for both attack and benign activities, along with fine-grained ground truth annotations. Rather than merely enabling the reporting of aggregate performance metrics, such datasets should facilitate in-depth behavioral analysis by offering detailed contextual information at the level of system events or objects. This level of granularity is essential for developers to understand detection outcomes, diagnose failure cases, and iteratively improve PIDS models. Third, practical datasets should support a certain degree of customization to accommodate diverse research objectives. For example, they should allow the inclusion of attacks at different scales and complexities, as well as the adjustment of the intensity and frequency of both malicious and benign behaviors. Such flexibility is crucial for evaluating system robustness under varied and realistic conditions.

This paper presents initial progress toward creating the aforementioned dataset for PIDS. Specifically, we make the following contributions:

- First, we review existing PIDS datasets and summarize their key limitations.
- Second, we propose two new evaluation metrics to provide real and insightful measurements of the performance. We discuss the corresponding requirements on fine-grained ground truth annotations needed to support these metrics.
- Finally, leveraging a state-of-the-art automated attack emulation framework, we construct a small-scale dataset and evaluate eight representative PIDS on this dataset. Through an analysis of the experimental results, we identify several open challenges and outline directions for

future research.

4.2.2 Requirements for Modern APT Benchmark Datasets

4.2.2.1 Time-Weighted Detection Accuracy for Real-Time APT Detection

Existing metrics for intrusion/APT detection commonly quantify performance via unweighted detection rates, e.g., precision/recall computed over labeled detection targets such as nodes/edges in a provenance graph, time windows, or entire attack instances. A key limitation of these metrics is that they implicitly assign identical importance to all targets, regardless of when they occur in the attack. This assumption is misaligned with real-world APT detection and response, especially in the real-time setting: detecting an attack artifact early often enables containment and prevents downstream damage, whereas a correct alert triggered only in late stages may have substantially lower operational value because the harm has largely materialized. Our insight is that evaluation should reflect this asymmetric value by explicitly discounting late-stage detections and prioritizing early-stage detections. To this end, we propose a time-weighted variant of object-level precision and recall.

Specifically, let $\mathcal{I}$ denote the set of detection targets (e.g., nodes or edges), where each target $i \in \mathcal{I}$ has a timestamp t_i , a binary ground-truth label ($y_i \in \{0, 1\}$), and a binary prediction ($\hat{y}_i \in \{0, 1\}$). We define a non-increasing weighting function ($w(t_i) \geq 0$) that assigns larger weights to earlier targets and smaller weights to later ones. The time-weighted recall and time-weighted precision are then:

$$R_w = \frac{\sum_{i \in \mathcal{I}} w(t_i) \mathbb{I}[y_i = 1 \wedge \hat{y}_i = 1]}{\sum_{i \in \mathcal{I}} w(t_i) \mathbb{I}[y_i = 1]} \quad (4.1)$$

$$P_w = \frac{\sum_{i \in \mathcal{I}} w(t_i) \mathbb{I}[y_i = 1 \wedge \hat{y}_i = 1]}{\sum_{i \in \mathcal{I}} w(t_i) \mathbb{I}[\hat{y}_i = 1]} \quad (4.2)$$

In practice, $w(\cdot)$ can be instantiated using simple, interpretable discounting schemes, such as linear discount $w(t) = 1 - \tau(t)$, where $\tau(t) \in [0, 1]$ is a normalized time index, or exponential discount $w(t) = \exp(-\lambda\tau(t))$, which more strongly penalizes late detections and admits a clear “half-life” interpretation for operational response windows. This metric family preserves the familiar semantics of precision/recall while aligning evaluation with the fundamental objective of real-time APT defense: maximizing the utility of early correct detections.

4.2.2.2 *TTP Coverage at a Quality Threshold*

Prior object-based evaluation metrics for PIDS typically report aggregate precision/recall (or detection rate) over all labeled detection targets (e.g., nodes, edges, or time windows). While useful, such metrics provide little insight into which adversary behaviors are being detected reliably. In practice, defenders often prefer detectors with broader detection surface, i.e., systems that accurately cover a wider range of MITRE ATT&CK Tactics, Techniques, and Procedures (TTPs), over systems that perform well only on a narrow subset. Aggregate object-level metrics can mask this limitation: a detector may achieve high overall recall by excelling on a few frequent TTPs while consistently failing on many others, yet this failure remains invisible in a single global score. To explicitly quantify behavioral coverage, we propose *TTP Coverage@threshold*, which measures the fraction of TTPs for which the detector meets a required accuracy criterion. Let $\mathcal{T}$ be the set of ATT&CK techniques or sub-techniques under consideration. For each detection target $i \in \mathcal{I}$, we assume a ground-truth TTP label list $t(i) \in \mathcal{T}$. Let $y_i \in \{0, 1\}$ denote whether i is malicious (positive) and $\hat{y}_i \in \{0, 1\}$ denote the detector’s decision. For each $t \in \mathcal{T}$, we compute a per-TTP

quality score Q_t (e.g., $F\beta$, t , or recall/precision), derived from per-TTP precision and recall:

$$P_t = \frac{\sum_{i \in \mathcal{I}} \mathbf{1}\{t(i) = t \wedge y_i = 1 \wedge \hat{y}_i = 1\}}{\sum_{i \in \mathcal{I}} \mathbf{1}\{t(i) = t \wedge \hat{y}_i = 1\}} \quad (4.3)$$

$$R_t = \frac{\sum_{i \in \mathcal{I}} \mathbf{1}\{t(i) = t \wedge y_i = 1 \wedge \hat{y}_i = 1\}}{\sum_{i \in \mathcal{I}} \mathbf{1}\{t(i) = t \wedge y_i = 1\}} \quad (4.4)$$

Given a user-specified quality threshold $\tau \in [0, 1]$, we define TTP Coverage@threshold as the proportion of TTPs whose per-TTP quality meets the threshold:

$$\text{Coverage@}\tau = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \mathbf{1}\{Q_t \geq \tau\}, \text{ where } Q_t \in \{P_t, R_t\}. \quad (4.5)$$

Optionally, to prevent degenerate cases where high recall is achieved with unacceptably low precision (or vice versa), we also consider a two-constraint coverage definition:

$$\text{Coverage@}(\tau_P, \tau_R) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \mathbf{1}\{P_t \geq \tau_P \wedge R_t \geq \tau_R\}. \quad (4.6)$$

This metric directly captures the detector’s effective behavioral surface under a minimum quality bar, complementing aggregate object-level scores and enabling fine-grained analysis of which ATT&CK behaviors are detected reliably versus those that remain blind spots. Knowledge-driven provenance analysis has also shown that ATT&CK-aligned detection can be operationalized by matching behavior templates/subgraphs against execution provenance, producing interpretable evidence for technique-level detection [152].

4.2.2.3 *Requirements for the Dataset.*

Computing Time-Weighted Detection Accuracy and TTP Coverage@threshold requires more fine-grained dataset labeling. In addition to the object-level ground truth, such as attack-related nodes and edges, for time-weighted evaluation, each object must be associated with a timestamp on the attack timeline. For TTP-aware evaluation, each object must additionally carry a multi-label annotation indicating the corresponding MITRE ATT&CK behavior(s), i.e., a set of labels $\mathcal{T}_i \subseteq \mathcal{T}$ for each object i . In practice, this implies that the dataset must include (i) a structured representation of attack activity as objects (nodes/edges) with stable identifiers, (ii) per-object temporal metadata, and (iii) per-object ATT&CK label sets. Many existing datasets do not satisfy these requirements, which motivate the need for datasets with fine-grained, temporally ordered, and ATT&CK-aligned ground truth to enable meaningful real-time and behavior-centric evaluation of PIDS.

4.2.3 **Dataset Generation Pipeline**

A high-quality dataset for PIDS must satisfy several requirements. First, it should ensure complete and causally consistent event capture. All security-relevant system activities must be recorded without loss, sampling, or filtering, while preserving fine-grained temporal and causal order. Missing or reordered events can break provenance chains, distort graph structures, and significantly impair both detection accuracy and root-cause analysis. Second, a representative dataset must provide realistic, benign workloads and up-to-date attacks. Benign traces should include long-running, heterogeneous system activities, such as user interactions, background services, and application workflows, to reflect natural system variance and avoid overfitting. Meanwhile, attack traces should span multiple stages to better approximate real-world intrusion campaigns. Finally, reliable evaluation requires precise ground truth. Fine-grained labels at the event or entity level are essential for assessing and analyzing detection performance. These requirements together form the

basis for constructing and evaluating robust provenance-based intrusion detection systems.

4.2.3.1 Leveraging Aurora for Attack Emulation

A high-quality PIDS dataset should include up-to-date and diverse APT attack samples. To this end, we employ AURORA, an automated cyberattack emulation system, to systematically generate and execute such attack scenarios. AURORA collects TTP implementations from multiple sources, including the MITRE ATT&CK knowledge base, threat emulation frameworks such as Atomic Red Team [41], and converts them into structured Attack Actions. It then provides a planning-driven orchestration mechanism that composes individual attack TTPs into coherent attack chains. Specifically, it models an attack action as the minimal executable unit in an attack plan. It uses the Attack Action Linking Model, an extensible set of PDDL predicates, to formally describe the preconditions and effects of each action, thereby enabling the automated linking of actions into valid multi-step attack chains [11]. It supports generating customized attack chains for different emulation scenarios and provides the Python scripts to execute the generated attacks.

4.2.3.2 Background Workload and Environment Setup

A frequent limitation of existing provenance datasets is that benign activity is either overly scripted or lacks the multi-process and multi-service interactions that characterize real endpoints, causing detectors to overfit to simplistic normal patterns. Our benign traces are designed to capture a broad spectrum of realistic day-to-day activities. To better mimic real-world complex networks from the perspective of an endpoint provenance graph, we design benign scenarios that induce (i) fragmented network activity initiated by multiple cooperating processes, (ii) diverse short-lived connections to heterogeneous domains/services, and (iii) long causal chains that interleave network, file, and process behaviors across applications. We currently focus on three high-impact

benign scenarios that are common in daily enterprise and personal computing and are known to generate noisy, non-linear provenance structures.

Scenario 1: Office knowledge work with browser-centric network interactions. This scenario captures routine information seeking and document handling workflows such as web searching, downloading, opening documents, and saving/organizing artifacts. A key realism factor is the modern browser architecture: rather than a single monolithic process, browsers use a multi-process design (e.g., browser/UI, renderer, GPU, network/service, utility, and extension processes). Consequently, network connections, downloads, and auxiliary file writes are often performed by non-UI helper processes, producing fragmented yet correlated provenance traces with multiple subjects participating in a single user-visible task. In addition, office applications and document viewers frequently spawn helper processes (e.g., preview/rendering, decompression, plugin components), further increasing process diversity and cross-process data flows. From a network standpoint, benign activity here is dominated by bursts of short connections, multiple DNS resolutions, and accesses to a wide range of domains and content delivery endpoints, which collectively form long and noisy causal chains spanning network objects and file objects.

Scenario 2: General OS usage with installation, updates, and configuration traffic. This scenario reflects day-to-day system maintenance behaviors that routinely occur in real deployments, including software installation, updates, and configuration changes. Such tasks naturally generate process trees rooted at installers/updaters and involve a mixture of file writes (binaries, libraries, caches), registry/configuration edits, and intermittent network downloads. Importantly, these benign traces can be structurally similar to attacker behaviors at the system-call level (e.g., staged downloads, execution of newly written binaries, and extensive configuration modifications), thereby providing a more challenging and realistic background for provenance-based detectors

than repetitive scripted workloads. At the network layer, the activity commonly includes accesses to vendor update servers, package repositories, and telemetry endpoints, often with retries and fallbacks, contributing additional non-determinism and noise in the resulting provenance graphs.

Scenario 3: Developer/power-user workflows with command-line intensive and iterative behavior. This scenario models advanced user activities such as using terminals, scripting, building/testing software, and troubleshooting environment issues. Compared to casual usage, developer workflows exhibit high process turnover (frequent process creation/termination), intensive file system interactions (build artifacts, logs, caches), and more direct network actions (e.g., dependency fetching, remote repository access, and tooling telemetry). A defining feature is the prevalence of failure-repair-retry loops: commands are often executed repeatedly with slight modifications until they succeed, and intermediate failures are common.

We explore two complementary approaches for benign data collection. First, we recruit human volunteers to emulate typical user behaviors in a controlled environment. Beyond common activities such as browsing TikTok, downloading files, visiting webpages, checking emails, and playing video games, participants also perform a variety of everyday user and system workflows, including document editing, software installation and updates, web-based communication, local file management, and media playback. Together, these activities induce diverse process, file, and network interactions, enabling us to construct a representative and robust benign baseline. Second, we leverage a computer-interactive large language model (LLM) agent, referred to as a Computer-Using Agent (CUA) [153], to simulate human-like interactions with the operating system. The CUA autonomously performs routine tasks such as web browsing and office software manipulation, generating benign traces that are consistent yet scalable. This approach complements human-driven traces by providing reproducible and diverse benign behaviors without requiring continuous

and expensive human involvement.

4.2.3.3 Data Collection and Formalization

After collecting raw ETW audit logs, we formalize them into a unified provenance representation to support subsequent graph construction and intrusion detection. Specifically, we convert the raw CSV-form ETW records into a line-delimited JSON stream consisting of node and event objects, where nodes represent system entities (e.g., processes, files, registry keys, and network flows) and events represent typed interactions between entities.

We first load the raw ETW CSV records and retain only the entries that correspond to the structured payload format used by our collector (i.e., records tagged as `ProtoBuf`). For each record, we normalize fields into a canonical tuple (*Time*, *EventType*, *processID*, *threadID*, *Timestamp*, *EventDetails*). The *EventType* is derived by mapping the ETW provider/object name and opcode to a semantic event type using an external `format.json`. This step abstracts away ETW-specific identifiers and yields a compact set of security-relevant event types (e.g., `ProcessStart`, `ProcessEnd`, `FileIoRead`, `FileIoWrite`, `ImageLoad`, and network/registry events).

To build a consistent provenance graph, we assign each observed entity a globally unique node identifier and maintain an in-memory index that maps canonical entity identifiers to node IDs. We use type-specific identifiers to deduplicate entities across events: processes are keyed by PID (e.g., `PID:pid`), files are keyed by full path (e.g., `File:path`), registry keys are keyed by key path (e.g., `Key:path`), and network flows are keyed by a 5-tuple derived from the ETW payload (source/destination IP and ports). Whenever an event references a previously unseen entity, we emit a `NODE` record and update the index; otherwise, we reuse the existing node ID. Each node contains a generated UUID and a typed attribute set (e.g., `PID/PPID/process name/command line`

for processes, path for files/keys, and IP/port fields for netflows).

Each normalized ETW record is translated into a provenance `EVENT` record with the schema $(id, time, type, s, d, d2)$. Here, s denotes the subject entity (typically a process), d denotes the primary object entity (e.g., a file, a netflow, or a registry key), and $d2$ optionally denotes a secondary object when required by the event semantics (e.g., *FileIoRenamePath*, where the event links an old path to a new path). The event time is taken from the ETW timestamp field, and each event is assigned a fresh UUID for traceability. For events indicating that data collection starts on pre-existing entities (e.g., *ProcessDCStart* and *ImageDCStart*), we treat them as graph initialization signals: the former adds a pre-existing process node, while the latter may introduce a pre-existing image file and emits a corresponding *ImageLoad* edge to preserve the implied dependency.

We output the formalized provenance in a streaming, line-delimited JSON representation. Each record is encoded as a *NODE* or *EVENT* object with a corresponding *logData* payload, which provides a compact and append-friendly interface for downstream provenance-graph construction and analysis.

4.2.3.4 Labeling and Ground Truth

A key requirement of provenance benchmarks is precise and reproducible ground truth. While our attack execution is deterministic due to the YAML-based attack chains, mapping each attack step to observed provenance logs still requires careful step-to-log matching. In our current work, ground truth is generated through a manual procedure that produces (i) a malicious entity label set and (ii) optional per-object ATT&CK labels.

Extract step-level anchors from `attack_chain.yml` and auxiliary notes. For each attack chain, we parse the corresponding `attack_chain.yml` and enumerate its ordered `command` steps. Since some operational details are not explicitly present in the YAML or console outputs

(e.g., attacker IP addresses, payload file names), the annotator first records such auxiliary indicators during attack execution. These indicators are later used as high-confidence anchors for matching.

Precise step-to-log matching in provenance.jsonl. Given the extracted anchors, the annotator performs targeted keyword matching over `provenance.jsonl` to retrieve candidate log entries associated with each attack step. For long commands, we use distinctive substrings (avoiding brittle tokens such as escape characters) to reduce false matches, and we may record either the full command, a partial command, or the full JSON line as evidence, as long as it reliably distinguishes attack-related entries. Each matched JSON entry explicitly contains the referenced node identifiers (for NODE records) or $(s, d, d2)$ endpoints (for EVENT records), allowing the annotator to precisely map matched log entries to the corresponding malicious entities. Importantly, matching is performed against structured NODE/EVENT records rather than free-text logs; thus every retrieved entry directly exposes the stable node IDs (and event endpoints) that are labeled.

Construct the malicious entity set and generate ground-truth files. All entities that are directly involved in the execution of attack actions are marked as malicious ground truth, following a conservative strategy that does not label entire causal neighborhoods or transitively affected entities. We materialize the malicious entity set in a label file (`labels.txt`), and then generate the node-level ground truth CSV using a script which retrieves and extracts corresponding contents from `provenance.jsonl` according to `labels.txt`. To facilitate downstream evaluation and alignment with attack steps, we optionally reorder the resulting CSV by node appearance order and then annotate per-object ATT&CK tactic/technique by aligning the ordered objects with the ordered command steps in `attack_chain.yml`.

Non-observable steps and matching boundaries. Not every step in an attack chain is observable on the victim host (e.g., attacker-side payload generation or preparation). For such steps, we do not force a step-to-log correspondence; instead, we only label victim-observable entities and

actions to avoid introducing spurious ground truth.

4.2.4 Analysis of the Generated Data

4.2.4.1 Evaluation Setup.

To evaluate provenance-based intrusion detection under controlled yet realistic multi-stage attack scenarios, we construct two experimental datasets, ProvAttack1 and ProvAttack2, derived from the same benign execution environment but injected with different attack chains.

Datasets. Both datasets are generated on identical system configurations and share the same benign workload traces. The only difference lies in the injected attack behavior. Specifically, ProvAttack1 contains the 11th attack chain defined in the Aurora repository, while ProvAttack2 contains the 21st attack chain. Each attack chain is executed once in isolation to avoid interference between attack instances.

Attack Execution. Attack traces are generated using Aurora’s automated attack orchestration framework. Aurora executes attack chains defined as ordered compositions of attack techniques, specified in YAML-based attack chain descriptions. These descriptions explicitly encode the sequence of attack actions and their execution dependencies, ensuring deterministic and reproducible attack execution.

Ground Truth Annotation. Ground truth labels are derived directly from the corresponding Aurora attack chain YAML files. For each dataset, we identify all system entities (processes, files, registry keys, and network objects) that are directly involved in the execution of attack actions. These entities are marked as malicious ground truth. Importantly, we do not label entire causal neighborhoods or transitively affected entities; only nodes explicitly participating in the attack chain are annotated. This conservative labeling strategy avoids ambiguity and ensures precise evaluation of detection and root-cause identification. To ensure consistent technique seman-

tics across datasets and facilitate behavior-centric evaluation, such labels can be normalized using technique-level knowledge representations derived from threat intelligence, e.g., technique knowledge graphs [40].

Detection Systems. To evaluate our dataset under representative provenance-based intrusion detection pipelines, we adopt the open-source evaluation framework provided by Tristan Bilot et al. [103]. Specifically, we integrate our datasets into PIDSMaker, the unified benchmarking framework open-sourced by the authors, which implements a diverse set of state-of-the-art PIDS with standardized preprocessing, graph construction, and evaluation procedures. Following the original study, we evaluate our data using the same eight detection systems including FLASH [13], KAIROS [14], MAGIC [154], NODLINK [20], ORTHRUS [155], R-CAID [24], THREATTRACE [72] and VELOX [103].

4.2.4.2 *Detection Performance and Analysis*

We perform node-level detection; Table 4.6 shows the results. The evaluation on PROVATTACK datasets reveals detection behaviors highly consistent with prior work on DARPA and OpTC datasets [103]. We observe that zero precision at node-level detection is not an anomaly but a recurring outcome across multiple provenance-based intrusion detection systems. Although some systems achieve near-perfect recall on the PROVATTACK datasets, their alerts are overwhelmingly dominated by false positives, rendering the detection results impractical for real-world forensic analysis. In contrast, ORTHRUS remains one of the few systems capable of producing compact and high-precision alert sets under node-level evaluation on PROVATTACK. Meanwhile, several systems fail to raise any actionable node-level alerts at all, further highlighting the sensitivity of threshold-based detection pipelines to dataset-specific characteristics.

From the perspective of benchmark construction, the observed zero precision and extremely

Table 4.6: Detection results on PROVATTACK datasets

System	Precision(%)	Recall(%)	TP	FP
PROVATTACK1				
FLASH	0.020	55.00	11	55080
KAIROS	0.000	0.000	0	0
MAGIC	0.080	100.0	20	24843
NODLINK	0.011	5.000	1	9219
ORTHRUS	0.000	0.000	0	9
R-CAID	0.000	0.000	0	0
THREATTRACE	0.023	100.0	20	88235
VELOX	0.000	0.000	0	0
PROVATTACK2				
FLASH	0.266	34.09	15	5632
KAIROS	0.000	0.000	0	0
MAGIC	1.985	95.46	42	2074
NODLINK	0.986	29.55	13	1306
ORTHRUS	36.36	9.091	4	7
R-CAID	0.000	0.000	0	0
THREATTRACE	0.332	100.0	44	13202
VELOX	0.000	0.000	0	0

high false-positive rates can be attributed to multiple factors. First, all systems are evaluated using the default hyperparameters provided in the PIDSMaker framework, without extensive dataset-specific tuning. While this choice ensures consistency and reproducibility across systems, it may lead to suboptimal performance for specific models whose behavior is highly sensitive to hyperparameter settings. Recent advances have started to explicitly address this sensitivity by making rule parameters differentiable and tuneable with gradient-based optimization, enabling lightweight adaptation without redesigning detection logic [8]. A more thorough hyperparameter optimization could potentially improve detection quality, although such tuning is non-trivial and often requires access to test data, which may introduce data leakage concerns.

Second, the representativeness of the training data plays a critical role in detection performance. If the training data does not sufficiently cover the diversity of benign system behaviors that appear during testing, models may incorrectly assign high anomaly scores to previously unseen but benign activities, resulting in a large number of false positives. This issue is particularly pronounced in provenance-based intrusion detection, where benign behavior spaces are large, evolving, and difficult to capture exhaustively. This observation aligns with recent arguments that provenance-based detection should be evaluated against a learned baseline of normal system behavior, where distributional gaps between train and test benign traces can dominate false-positive outcomes [156].

This issue highlights the importance of better benign behavior coverage and train–test alignment, which we discuss as a direction for future benchmark extension.

4.2.4.3 Time-Weighted Recall Analysis

Conventional *Recall* only measures how many malicious entities are eventually detected, but does not differentiate between early detections and late detections. However, in real-time APT detection, the timing of detection is critical: detecting a malicious node earlier in the attack chain is typically more valuable for containment and response.

To further evaluate whether the generated data supports real-time APT detection assessment (beyond conventional static metrics), we analyze detector performance using time-weighted recall (R_w) in addition to standard recall. In practice, we select linear discount $w(t) = 1 - \tau(t)$, where $\tau(t) \in [0, 1]$ is a normalized time index, and exponential discount $w(t) = \exp(-\lambda\tau(t))$ with three sensitive levels ($\lambda = 0.5, 1, 3$) represent weak, moderate, and strong emphasis on early detections, respectively.

We randomly sampled three Aurora attack-chain instances: Aurora_100, Aurora_105, Aurora_123, For each attack instance, we evaluate eight detection systems and report: un-

weighted *recall*, $R_{w-linear}$, $R_{w-\lambda=0.5}$, $R_{w-\lambda=1}$, $R_{w-\lambda=3}$.

Table 4.7: Real-time APT Detection recall(%) on generated datasets

System	Unweighted	Linear	$\lambda = 0.5$	$\lambda = 1$	$\lambda = 3$
AURORA_100					
FLASH	0.000	0.000	0.000	0.000	0.000
KAIROS	0.000	0.000	0.000	0.000	0.000
MAGIC	100.000	100.000	100.000	100.000	100.000
NODLINK	10.000	10.055	10.020	10.040	10.119
ORTHRUS	0.000	0.000	0.000	0.000	0.000
R-CAID	0.000	0.000	0.000	0.000	0.000
THREATTRACE	100.000	100.000	100.000	100.000	100.000
VELOX	0.000	0.000	0.000	0.000	0.000
AURORA_105					
FLASH	0.000	0.000	0.000	0.000	0.000
KAIROS	1.176	1.144	1.148	1.152	1.102
MAGIC	25.882	25.551	25.474	25.408	24.438
NODLINK	10.588	10.599	10.616	10.633	10.726
ORTHRUS	0.000	0.000	0.000	0.000	0.000
R-CAID	0.000	0.000	0.000	0.000	0.000
THREATTRACE	100.000	100.000	100.000	100.000	100.000
VELOX	1.176	1.140	1.144	1.149	1.092
AURORA_123					
FLASH	76.000	42.343	29.315	20.938	7.541
KAIROS	0.000	0.000	0.000	0.000	0.000
MAGIC	88.000	88.605	88.098	87.743	91.643
NODLINK	17.333	59.326	66.995	73.598	91.603
ORTHRUS	0.000	0.000	0.000	0.000	0.000
R-CAID	0.000	0.000	0.000	0.000	0.000
THREATTRACE	97.403	93.246	91.824	90.728	91.648
VELOX	4.000	1.866	1.115	0.689	0.001

Table 4.7 summarizes the results on the three sampled attack instances. Overall, R_w provides information that standard recall cannot capture:

When recall is partial, R_w reveals whether detected malicious nodes are concentrated early or late in the attack timeline. This can lead to meaningful score changes under stronger time

weighting (larger λ). Besides, the effect is attack-dependent. `Aurora_100` shows minimal temporal differentiation, `Aurora_105` shows mild differentiation, and `Aurora_123` shows strong differentiation.

We highlight several representative cases.

High recall but weak early-detection utility On `Aurora_123`, `flash` achieves a relatively high unweighted recall (76%), but its time-weighted performance drops sharply to 42.343% with linear discount. And its exponential-time-weighted recall encounters a significant decline from 29.315% ($\lambda = 0.5$) to 7.541% ($\lambda = 3.0$). This indicates that many malicious nodes are detected only in later stages of the attack window. In other words, standard recall overestimates its practical real-time utility in this case.

Low recall but strong early-detection utility In contrast, `nodlink` has a low unweighted recall (17.333%) on `Aurora_123`, but much higher time-weighted scores: $R_{w-linear} = 59.326\%$ and $R_{w-\lambda=1.0} = 73.598\%$. This means the relatively small subset of malicious nodes it detects tends to occur very early in the attack chain. From a real-time defense perspective, these detections may be disproportionately valuable despite modest overall recall.

Strong detectors can still differ in temporal quality Both `magic` and `threatrace` achieve strong performance on `Aurora_123`, but R_w still helps differentiate their temporal detection behavior. `magic` tends to improve under stronger time weighting, suggesting earlier detections on average; `threatrace` remains high but shows a different sensitivity pattern, indicating a broader temporal spread of detected malicious nodes. This shows that R_w is useful not only for distinguishing weak or strong detectors, but also for comparing temporal detection quality among strong detectors.

Mild temporal differentiation in Aurora_105. `Aurora_105` exhibits a more moderate effect: `magic` slightly decreases as λ increases (detections relatively later), `nodlink` slightly increases (detections relatively earlier), `kairos` and `velox` remain close to unweighted recall due to very few true positives. This suggests that not all attack instances are equally sensitive to temporal weighting, which further motivates evaluation across multiple attack chains.

These results demonstrate that the generated data supports a time-aware evaluation protocol that exposes important differences in real-time detection behavior that are invisible under standard recall alone. In particular, R_w distinguishes detectors that detect many but late from those that detect fewer but earlier, a distinction that is especially relevant in APT response scenarios.

4.2.5 Discussion and Limitations

4.2.5.1 Experimental Design and Evaluation Sensitivity

The current study is subject to limitations related to experimental design and evaluation scope. While this work provides an initial and systematic evaluation of provenance-based intrusion detection systems on the PROVATTACK benchmark, several evaluation dimensions outlined earlier in the paper have not yet been fully realized. In particular, the experiments corresponding to § 4.2.2.1 and § 4.2.2.2 remain to be designed and executed. As a result, the present evaluation primarily focuses on static, post-hoc detection outcomes, without explicitly assessing detection timeliness or technique-level coverage under practical alerting constraints.

In addition, detection performance can be sensitive to experimental configurations such as hyperparameter settings. In the current evaluation, all systems are assessed using default hyperparameters provided by the PIDSMaker framework, without extensive dataset-specific tuning. While this choice ensures consistency and avoids data leakage, it may prevent certain systems from operating at their optimal detection points.

Together, these factors indicate that some observed detection behaviors may reflect limitations of the current experimental setup rather than fundamental weaknesses of the underlying detection approaches. At the same time, they highlight the challenges of designing comprehensive and fair experimental evaluations for provenance-based intrusion detection systems.

4.2.5.2 Scale and Diversity of Attack Traces

Another limitation of the current benchmark lies in the scale and diversity of attack traces. Both PROVATTACK1 and PROVATTACK2 contain a single attack chain generated using the Aurora framework, and each attack unfolds over a relatively short execution window. This design allows for controlled and interpretable evaluation, but it restricts the scope of attack scenarios represented in the benchmark.

As a result, the current evaluation primarily reflects system behavior under isolated, short-lived attack conditions. It does not assess detection performance in more complex scenarios involving multiple concurrent or temporally overlapping attack chains, nor does it capture long-running or stealthy attacks that evolve gradually over extended periods. Such scenarios are common in real-world APT campaigns and may pose additional challenges for provenance-based detection systems.

4.2.5.3 Benign Behavior Coverage and Data Editing

A further limitation of the current benchmark lies in the representativeness of benign training data. Due to the session-based nature of provenance capture, benign behaviors are often fragmented across isolated execution windows. As a result, training data may not sufficiently cover the full range of normal system behaviors that appear during testing, even when both are collected from the same host. This mismatch can lead detection systems to flag previously unseen but benign

activities as anomalous incorrectly.

This issue is inherent to provenance-based datasets, where entity identifiers are session-scoped, and causal chains are frequently truncated by capture boundaries such as reboots or deployment constraints. Simply aggregating benign logs in temporal order does not resolve this problem, as semantic continuity across sessions remains broken.

4.2.5.4 *Future Work*

To address above limitations, we will extend PROVATTACK along key experimental dimensions as future work. We plan to (i) complete the evaluation of time-weighted detection accuracy and technique-level coverage, and conduct systematic hyperparameter sensitivity analyses using training/validation data only; (ii) incorporate longer and more realistic traces, including multi-stage attack chains and their variants, to assess robustness and scalability beyond isolated instances; and (iii) improve benign behavior coverage via data editing and expansion, such as canonicalizing session-specific entity identifiers into stable identities and selectively merging benign logs from the same host, so that learned normality better reflects long-running cross-session execution and reduces spurious false positives.

4.2.6 **Section Summary**

This section argued that the public benchmark datasets on which the PIDS literature has long relied no longer reflect the threats they purport to evaluate, and that this gap can be closed by an automated pipeline rather than by another round of manual data collection. Building on AURORA for attack emulation and a structured benign-workload generator, the pipeline produces large-scale, fine-grained, and continuously updatable attack traces with event-level ground truth. Evaluating eight representative PIDS on the resulting ProvAttack datasets revealed systematic generalization

failures that are entirely invisible on prior benchmarks, demonstrating that stale evaluation infrastructure conceals real weaknesses in the field.

4.3 Chapter Summary

This chapter addressed the second operational gap identified in Chapter 1: the lack of realistic infrastructure for testing and evaluating APT defense systems. AURORA closed the first half of the gap by automatically generating actionable, causally coherent, customizable, and scalable attack chains from CTI reports and MITRE ATT&CK techniques, four properties that no prior emulation system satisfies simultaneously. The next-generation datasets pipeline closed the second half by reusing AURORA as its attack engine and pairing it with structured benign workloads, producing benchmark datasets whose evaluation results expose blind spots that the field's standard benchmarks have been masking. Together, the two contributions form an end-to-end pipeline in which attack emulation feeds dataset generation, and dataset generation in turn drives credible PIDS evaluation, providing the testing and evaluation infrastructure that the next generation of operational APT defense research requires.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Summary of Contributions

This dissertation has argued that the gap between academic APT defense research and operational practice is not a single problem but a system of two interconnected deficiencies, in detection efficiency and in the infrastructure for realistic testing and evaluation, and that closing both simultaneously is a prerequisite for APT defense research to reach operational maturity. The five research projects presented in this dissertation collectively address these deficiencies through machine learning, generative AI, and symbolic reasoning, each designed with real-world deployment constraints as a first-order concern.

Gap I: Efficiency in Provenance-Based Intrusion Detection.

CAPTAIN addressed the core deployability failure of rule-based PIDS: the reliance on static, manually configured propagation parameters that cannot adapt to the behavioral diversity of real enterprise environments. By reformulating rule configuration as a differentiable optimization problem and applying gradient descent on benign training data, CAPTAIN reduced false-positive rates by over $11\times$ compared to prior rule-based systems while preserving their lightweight, real-time, and interpretable properties. CAPTAIN+ extended this contribution with an online learning module that continuously updates detection parameters during inference using real-time feedback, enabling the system to adapt to concept drift without periodic retraining and making it viable for long-running production deployments. GraphFaaS addressed a complementary bottleneck that had received little prior attention: the bursty, irregular nature of detection workloads in produc-

tion environments, which renders statically provisioned GNN inference pipelines either wasteful during quiet periods or insufficient during spikes. By re-architecting GNN inference on a serverless platform, GraphFaaS reduced average detection latency by $6.7\times$ and latency variance by 64% compared to a statically provisioned baseline, without sacrificing detection accuracy.

Gap II: Realistic Testing and Evaluation Infrastructure.

AURORA addressed the fundamental challenge of generating high-fidelity attack emulation data at scale. By formalizing four properties that any practical emulation system must satisfy, actionable, causally coherent, customizable, and scalable, and demonstrating that no existing system simultaneously achieves all four, AURORA used LLM-powered symbolic planning to automatically generate and execute multi-stage attack chains from CTI reports and MITRE ATT&CK techniques. The resulting dataset is $15\times$ larger than the leading expert-crafted benchmark and was generated from 250 threat intelligence reports with no manual engineering per attack chain.

The next-generation datasets work documented three critical limitations of existing PIDS benchmarks, outdated attack techniques, coarse-grained labeling, and overfitting-induced performance inflation, and demonstrated that these limitations are not merely measurement inconveniences but actively inflate reported performance and slow real progress. By constructing a dataset generation pipeline built on AURORA, the work produced large-scale, fine-grained, and continuously updatable attack traces for PIDS evaluation. Evaluating eight representative PIDS on this dataset revealed systematic generalization failures that are entirely invisible on prior benchmarks, underscoring how stale evaluation infrastructure conceals real weaknesses in the field.

The Coherence of the Two Thrusts.

Taken individually, each of these contributions addresses a well-defined technical problem. Taken together, they form a self-reinforcing research system that the dissertation’s operational-gap framing was designed to surface. AURORA’s attack emulation capability directly enables the

dataset pipeline: without the ability to generate causally coherent, replayable attack chains at scale, constructing a comprehensive and current evaluation corpus is infeasible. The dataset pipeline, in turn, reveals the generalization failures that motivate the adaptive detection work: it is only because the evaluation infrastructure is finally capable of presenting distribution-shifted attack behavior that the inadequacy of static detection parameters becomes measurable. And the adaptive, deployable detection systems, CAPTAIN, CAPTAIN +, GraphFaaS, are the ultimate consumers of this improved infrastructure, providing the operational endpoint toward which the other contributions build. The broader lesson is that realistic attack emulation, rigorous evaluation infrastructure, and operationally deployable detection are not independent research problems; they are stages of a single pipeline, and progress on each is necessary but not sufficient without the others.

5.2 Reflections on the Operational-Gap Framing

The framing of this dissertation around *operational gaps*, rather than around algorithmic novelty or benchmark performance, reflects a deliberate methodological stance. Security research has historically measured progress in terms of detection accuracy on controlled datasets or attack success rates in simplified environments. These metrics matter, but they are insufficient: a system that achieves state-of-the-art F1 on the DARPA TC dataset while failing to deploy in a real SOC, adapt to new attack patterns, or generalize beyond its training distribution has not actually advanced operational practice.

The operational-gap framing forced each project in this dissertation to confront deployment constraints, latency, memory, false-positive tolerance, workload variability, environment heterogeneity, as first-order design requirements rather than post-hoc considerations. CAPTAIN is designed to be lightweight enough to run on existing SOC infrastructure without GPU resources. GraphFaaS is designed to absorb workload spikes without over-provisioning. AURORA is de-

signed to produce replayable, causally coherent chains rather than static log snapshots that cannot be reproduced or extended. The next-generation datasets pipeline is designed to keep pace with evolving threat landscapes rather than freezing the field’s evaluation benchmarks at a single point in time. The insistence on operational deployability as a design criterion is, this dissertation argues, the correct orientation for security systems research at a time when the gap between academic results and operational practice remains wide and consequential.

5.3 Limitations

Despite the contributions described above, each project in this dissertation carries limitations that bound its current applicability and point toward future work.

CAPTAIN and CAPTAIN + require a representative sample of benign system activity for gradient-based parameter tuning. In environments where benign behavior is itself highly variable or where clean training data is difficult to obtain, the quality of the tuned parameters may degrade. The online adaptation mechanism in CAPTAIN + also introduces a risk of catastrophic forgetting under sustained concept drift, though the system’s modular design allows the learning rate and window size to be adjusted as a mitigation.

GraphFaaS inherits the cold-start latency characteristic of serverless platforms, which introduces a fixed overhead when new function instances are initialized in response to detection workload spikes. In threat scenarios where the adversary acts very rapidly within a single burst event, this cold-start penalty may create a detection window that a sufficiently fast attacker could exploit. The system also assumes access to a public cloud or private serverless infrastructure, which may not be available in air-gapped or highly restricted deployment environments.

AURORA’s attack chain quality depends on the quality and specificity of the CTI reports from which attack procedures are extracted. Vague or incomplete threat intelligence produces corre-

spondingly incomplete attack chains. The system also inherits the hallucination tendencies of the underlying LLMs, which can produce syntactically plausible but semantically incorrect PDDL action definitions; the current pipeline includes validation steps to catch these errors, but not all failures are detectable automatically.

The next-generation dataset pipeline produces attack traces that, while more diverse and up-to-date than prior benchmarks, are still generated in controlled virtual environments. Behavioral differences between virtualized and bare-metal hosts, between containerized and native services, and between simulated and genuine attacker tradecraft remain a source of distributional gap that no synthetic dataset can fully eliminate.

5.4 Future Research Directions

The limitations described above, together with the broader research agenda motivated by the operational-gap framing, suggest several concrete directions for future work.

Knowledge-Driven Provenance Analysis. A limitation to both rule-based and embedding-based PIDS is that they treat detection as a purely data-driven problem, without incorporating the structured domain knowledge available in threat intelligence reports, attack taxonomies, and known adversary profiles. We aim to address this by integrating knowledge graphs derived from CTI sources with provenance graph analysis, enabling detection that is simultaneously grounded in observed system behavior and informed by symbolic threat knowledge.

Automated Cyberrange Construction. A complementary need in the testing and evaluation infrastructure developed in Chapter 4 is the ability to spin up diverse target environments against which attack chains can be replayed. Toward this end, the author has developed Cyberrange-Sphere¹, an early-stage prototype that uses LLMs to generate Ansible and Vagrant configurations

¹<https://github.com/LexusWang/cyberrange-sphere>

for automatically provisioning vulnerable, attack-surface-rich environments. The current implementation establishes feasibility, but a systematic methodology, a taxonomy of supported attack surfaces, and a quantitative evaluation of the diversity and realism of generated environments remain open directions for future work.

Toward a Unified Attack-Defense Pipeline. At a broader level, the contributions in this dissertation suggest the outlines of a unified operational pipeline in which attack emulation, dataset generation, and detection are tightly integrated. In such a pipeline, AURORA would continuously generate up-to-date attack traces as new CTI reports emerge; the dataset pipeline would use these traces to maintain a current evaluation corpus; and adaptive PIDS systems would be retrained or re-tuned against this corpus. Realizing this vision requires research advances in automated pipeline orchestration, continual learning under distribution shift, and multi-tenant provenance analysis at cloud scale, all of which remain open problems.

Scaling to Cloud and Container Environments. The provenance graph abstraction was originally designed for host-based audit logs on individual machines. Modern enterprise workloads increasingly run in containerized, microservice, and serverless environments where system boundaries are fluid, audit telemetry is fragmented across many ephemeral processes, and the notion of a single “host” is poorly defined. Extending the provenance graph paradigm to these environments, with appropriate graph construction, entity resolution, and detection algorithms, is a necessary prerequisite for PIDS to remain relevant as infrastructure continues to evolve.

REFERENCES

- [1] U.S. Senate, Committee on Commerce, Science, and Transportation, “A “Kill Chain” analysis of the 2013 Target data breach,” U.S. Senate, Committee on Commerce, Science, and Transportation, Tech. Rep., Mar. 2014, Majority Staff Report for Chairman Rockefeller.
- [2] U.S. House of Representatives, Committee on Oversight and Government Reform, “The Equifax data breach,” U.S. House of Representatives, Committee on Oversight and Government Reform, Tech. Rep., Dec. 2018.
- [3] Microsoft, “Analyzing solorigate, the compromise of solarwinds orion platform,” Microsoft Security Response Center, Tech. Rep., 2021, Available at <https://www.microsoft.com/en-us/security/blog/2020/12/18/analyzing-solorigate-the-compromised-dll-file-that-started-a-sophisticated-cyberattack-and-how-microsoft-defender-helps-protect/>.
- [4] FireEye, “Highly evasive attacker leverages solarwinds supply chain to compromise multiple global victims with sunburst backdoor,” FireEye Threat Research, Tech. Rep., 2020, Available at <https://cloud.google.com/blog/topics/threat-intelligence/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor/>.
- [5] Cybersecurity and Infrastructure Security Agency, National Security Agency, and Federal Bureau of Investigation, “People’s republic of china state-sponsored cyber actor living off the land to evade detection,” Joint Cybersecurity Advisory AA23-144A, Tech. Rep., May 2023, Available at <https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-144a>.
- [6] X. Shen, Z. Li, G. Burleigh, L. Wang, and Y. Chen, “Decoding the mitre engenuity att&ck enterprise evaluation: An analysis of edr performance in real-world environments,” in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 96–111.
- [7] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, “MITRE ATT&CK: Design and philosophy,” The MITRE Corporation, Tech. Rep. PR-18-0944-11, 2018.

- [8] L. Wang et al., “Incorporating gradients to rules: Towards lightweight, adaptive provenance-based intrusion detection,” *arXiv preprint arXiv:2404.14720*, 2024.
- [9] Z. Li et al., “Incorporating gradients to rules: Towards online, adaptive provenance-based intrusion detection,” *IEEE Transactions on Dependable and Secure Computing*, 2025.
- [10] L. Wang, V. Yegneswaran, X. Shi, Z. Li, A. Gehani, and Y. Chen, “GraphFaaS: Serverless GNN inference for burst-resilient, real-time intrusion detection,” in *Workshop on ML for Systems at Neural Information Processing Systems*, 2025.
- [11] L. Wang et al., “From sands to mansions: Towards automated cyberattack emulation with classical planning and large language models,” *arXiv preprint arXiv:2407.16928*, 2024.
- [12] Q. Cai, L. Wang, Y. Zhu, Z. Chen, X. Shen, and Z. Li, “Building next-generation datasets for provenance-based intrusion detection,” in *Workshop on Attack Provenance, Reasoning, and Investigation for Security in the Monitored Environment (PRISM), NDSS Symposium*, 2026.
- [13] M. U. Rehman, H. Ahmadi, and W. U. Hassan, “Flash: A comprehensive approach to intrusion detection via provenance graph representation learning,” in *2024 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2024, pp. 3552–3570.
- [14] Z. Cheng et al., “Kairos: Practical intrusion detection and investigation using whole-system provenance,” in *2024 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2024, pp. 3533–3551.
- [15] J. Zeng et al., “Shadewatcher: Recommendation-guided cyber threat analysis using system audit records,” in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 489–506.
- [16] M. N. Hossain et al., “*SLEUTH*: Real-time attack scenario reconstruction from *COTS* audit data,” in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 487–504.
- [17] M. N. Hossain, S. Sheikhi, and R. Sekar, “Combating dependence explosion in forensic analysis using alternative tag propagation semantics,” in *2020 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2020, pp. 1139–1155.
- [18] M. Zipperle, F. Gottwalt, E. Chang, and T. Dillon, “Provenance-based intrusion detection systems: A survey,” *ACM Comput. Surv.*, vol. 55, no. 7, Dec. 2022.

- [19] Z. Li, Q. A. Chen, R. Yang, Y. Chen, and W. Ruan, "Threat detection and investigation with system-level provenance graphs: A survey," *Computers & Security*, vol. 106, p. 102 282, 2021.
- [20] S. Li et al., "Nodlink: An online system for fine-grained apt attack detection and investigation," *arXiv preprint arXiv:2311.02331*, 2023.
- [21] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," *arXiv preprint arXiv:2001.01525*, 2020.
- [22] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics.," in *NDSS*, 2021.
- [23] A. Alsaheel et al., "{Atlas}: A sequence-based learning approach for attack investigation," in *30th USENIX security symposium (USENIX security 21)*, 2021, pp. 3005–3022.
- [24] 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, 2024, pp. 3515–3532.
- [25] F. Yang, J. Xu, C. Xiong, Z. Li, and K. Zhang, "{Prographer}: An anomaly detection system based on provenance graph embedding," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4355–4372.
- [26] Q. Wang et al., "You are what you do: Hunting stealthy malware via data provenance analysis.," in *NDSS*, 2020.
- [27] C. Xiong et al., "Conan: A practical real-time apt detection system with high accuracy and efficiency," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 551–565, 2022.
- [28] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: Real-time apt detection through correlation of suspicious information flows," in *IEEE Symposium on Security and Privacy (SP)*.
- [29] W. U. Hassan et al., "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *26th Annual Network and Distributed System Security Symposium, NDSS*, The Internet Society, 2019.

- [30] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, “Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting,” *ACM SIGSAC Conference on Computer and Communications Security*, 2019.
- [31] *Tiber-eu framework: How to implement the european framework for threat intelligence-based ethical red teaming*, https://www.ecb.europa.eu/pub/pdf/other/ecb.tiber_eu_framework_2025~b32eff9a10.en.pdf.
- [32] *Official website of cymulate*, <https://cymulate.com/>.
- [33] *Welcome to pentera: Don't assume. validate. because 'pretty certain' doesn't mean secure*, <https://pentera.io/>.
- [34] *Are you vulnerable or exploitable? official website of horizon3*, <https://horizon3.ai/>.
- [35] *Auromated breach and attack simulation market*, <https://www.marketsandmarkets.com/Market-Reports/automated-breach-attack-simulation-market-43164821.html>.
- [36] *Darpa engagement data*, <https://drive.google.com/drive/folders/1okt4AYElyBohW4XiOBqmsvjwXsnUjLVf>.
- [37] 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*, 2016, pp. 1035–1044.
- [38] Y. Takahashi, S. Shima, R. Tanabe, and K. Yoshioka, “{Aptgen}: An approach towards generating practical dataset labelled with targeted attack sequences,” in *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*, 2020.
- [39] S. Choi, J.-H. Yun, and B.-G. Min, “Probabilistic attack sequence generation and execution based on mitre att&ck for ics datasets,” in *Cyber Security Experimentation and Test Workshop*, 2021, pp. 41–48.
- [40] Z. Li, J. Zeng, Y. Chen, and Z. Liang, “Attackg: Constructing technique knowledge graph from cyber threat intelligence reports,” in *European Symposium on Research in Computer Security*, Springer, 2022, pp. 589–609.

- [41] Red Canary, *Atomic red team test repository*, <https://github.com/redcanaryco/atomic-red-team>, Accessed: 2025-12-10.
- [42] Online, *Metasploit: The world's most used penetration testing framework*, <https://www.metasploit.com/>, 2025.
- [43] *Living off the land binaries, scripts and libraries* <https://lolbas-project.github.io/>.
- [44] *Mitre caldera*, <https://github.com/mitre/caldera>.
- [45] *Purple sharp*, <https://detectionlab.network/usage/purplesharp/>.
- [46] S. T. R. Team, *Splunk attack range*, Accessed: 2024-07-01, 2024.
- [47] M. Engenuity, *Cybersecurity: Att&ck® evaluations*, Accessed: 2025-06-01, 2024.
- [48] G. D. Pasquale et al., “ChainReactor: Automated privilege escalation chain discovery via AI planning,” in *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA: USENIX Association, Aug. 2024, pp. 5913–5929, ISBN: 978-1-939133-44-1.
- [49] A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wolf, “Intelligent, automated red team emulation,” in *Proceedings of the 32nd annual conference on computer security applications*, 2016, pp. 363–373.
- [50] J. Hoffmann, “Simulated penetration testing: From" dijkstra" to" turing test++",” in *Proceedings of the international conference on automated planning and scheduling*, vol. 25, 2015, pp. 364–372.
- [51] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, “Provenance-aware storage systems,” in *Unix annual technical conference, general track*, 2006, pp. 43–56.
- [52] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, “Hi-fi: Collecting high-fidelity whole-system provenance,” in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 259–268.
- [53] A. Gehani and D. Tariq, “Spade: Support for provenance auditing in distributed environments,” in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, Springer, 2012, pp. 101–120.
- [54] D. T. program, *Transparent Computing Engagement 3 Data Release*, 2020.

- [55] *Transparent-Computing*, <https://github.com/darpa-i2o/Transparent-Computing>, 2023.
- [56] D. T. program, *Operationally transparent cyber (optc) data release*, 2020.
- [57] A. Riddle, K. Westfall, and A. Bates, “Atlasv2: Atlas attack engagements, version 2,” *arXiv preprint arXiv:2401.01341*, 2023.
- [58] J. Liu, M. A. Inam, A. Goyal, A. Riddle, K. Westfall, and A. Bates, “What we talk about when we talk about logs: Understanding the effects of dataset quality on endpoint threat detection research,” in *2025 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2025, pp. 112–129.
- [59] Y. Huang, W. U. Hassan, Y. Guo, X. Chen, and D. Li, “Provsyn: Synthesizing provenance graphs for data augmentation in intrusion detection systems,” *arXiv preprint arXiv:2506.06226*, 2025.
- [60] 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,” in *30th Network and Distributed System Security Symposium (NDSS 23)*, 2023.
- [61] K. Mukherjee et al., “Evading {provenance-based}{ml} detectors with adversarial system actions,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1199–1216.
- [62] 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, 2019.
- [63] *Missed alarms and 40 million stolen credit card numbers: How target blew it*, <https://hbr.org/2014/03/could-target-have-prevented-its-security-breach>.
- [64] *Equifax says cyberattack may have affected 143 million the u.s.* <https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html>.
- [65] F. Dong et al., “Are we there yet? an industrial viewpoint on provenance-based endpoint detection and response tools,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2396–2410.

- [66] G. P. Spathoulas and S. K. Katsikas, “Using a fuzzy inference system to reduce false positives in intrusion detection,” in *2009 16th International Conference on Systems, Signals and Image Processing*, 2009, pp. 1–4.
- [67] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Comput. Surv.*, vol. 41, no. 3, Jul. 2009.
- [68] C. Dietrich, K. Krombholz, K. Borgolte, and T. Fiebig, “Investigating system operators’ perspective on security misconfigurations,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’18, Toronto, Canada: Association for Computing Machinery, 2018, pp. 1272–1289, ISBN: 9781450356930.
- [69] S. C. Sundaramurthy et al., “A human capital model for mitigating security analyst burnout,” in *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, 2015, pp. 347–359.
- [70] B. A. Alahmadi, L. Axon, and I. Martinovic, “99% false positives: A qualitative study of SOC analysts’ perspectives on security alarms,” in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, ISBN: 978-1-939133-31-1.
- [71] S. C. Committee, *A “kill chain” analysis of the 2013 target data breach*, 2014.
- [72] S. Wang et al., “Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning,” *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3972–3987, 2022.
- [73] F. B. Kokulu et al., “Matched and mismatched socs: A qualitative study on security operations center issues,” in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1955–1970.
- [74] K. Bandla, *Apt notes*, 2019.
- [75] J. Newsome and D. X. Song, “Dynamic taint analysis for automatic detection, analysis, and signaturegeneration of exploits on commodity software.,” in *NDSS*, Citeseer, vol. 5, 2005, pp. 3–4.
- [76] W. Xu, S. Bhatkar, and R. Sekar, “Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks.,” in *USENIX Security Symposium*, 2006, pp. 121–136.

- [77] S. Arzt et al., “Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps,” *ACM sigplan notices*, vol. 49, no. 6, pp. 259–269, 2014.
- [78] S. Ma, X. Zhang, and D. Xu, “Protracer: Towards practical provenance tracing by alternating between logging and tainting,” in *23rd Annual Network And Distributed System Security Symposium (NDSS 2016)*, Internet Soc, 2016.
- [79] K. H. Lee, X. Zhang, and D. Xu, “High accuracy attack provenance via binary-based execution partition,” in *NDSS*, vol. 16, 2013.
- [80] J. Park, D. Nguyen, and R. Sandhu, “A provenance-based access control model,” in *2012 tenth annual international conference on privacy, security and trust*, IEEE, 2012, pp. 137–144.
- [81] T. Pasquier et al., “Practical whole-system provenance capture,” in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 405–418.
- [82] S. S. Khan and M. G. Madden, “One-class classification: Taxonomy of study and review of techniques,” *The Knowledge Engineering Review*, vol. 29, no. 3, pp. 345–374, 2014.
- [83] D. M. Tax and R. P. Duin, “Data domain description using support vectors,” in *ESANN*, vol. 99, 1999, pp. 251–256.
- [84] D. M. Tax and R. P. Duin, “Support vector domain description,” *Pattern recognition letters*, vol. 20, no. 11-13, pp. 1191–1199, 1999.
- [85] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
- [86] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *Advances in neural information processing systems*, vol. 30, 2017.
- [87] Y. Chen, Y. Chi, J. Fan, and C. Ma, “Gradient descent with random initialization: Fast global convergence for nonconvex phase retrieval,” *Mathematical Programming*, vol. 176, pp. 5–37, 2019.
- [88] DARPA, *Transparent computing engagement 3 data release*, Accessed: 2024-09-01, 2018.
- [89] DARPA, *Transparent computing engagement 5 data release*, Accessed: 2024-09-01, 2019.

- [90] CISA, *North korean advanced persistent threat focus: Kimsuky*, 2020.
- [91] R. Canary, *Explore atomic red team*, 2023.
- [92] T. R. Lampert, *CHAOS: a PoC that allow generate payloads and control remote operating systems. Features like persistency and others can be achieved with a program updating and using the Dnsmasq service, which is also a feature of chaos*. <https://github.com/tiagorlampert/CHAOS>.
- [93] HackTricks, *D-bus enumeration & command injection privilege escalation*, Accessed: 2024-08-01, 2024.
- [94] Y. Nesterov, “A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$,” in *Dokl. Akad. Nauk. SSSR*, vol. 269, 1983, p. 543.
- [95] M. D. Zeiler, “Adadelata: An adaptive learning rate method,” *arXiv preprint arXiv:1212.5701*, 2012.
- [96] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization.,” *Journal of machine learning research*, vol. 12, no. 7, 2011.
- [97] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [98] Y. Wang et al., “{Gnnadvisor}: An adaptive and efficient runtime system for {gnn} acceleration on {gpu},” in *15th USENIX symposium on operating systems design and implementation (OSDI 21)*, 2021, pp. 515–531.
- [99] H. Hu, F. Liu, Q. Pei, Y. Yuan, Z. Xu, and L. Wang, “λGrapher: A resource-efficient serverless system for gnn serving through graph sharing,” in *Proceedings of the ACM Web Conference 2024*, 2024, pp. 2826–2835.
- [100] J. Thorpe et al., “Dorylus: Affordable, scalable, and accurate {gnn} training with distributed {cpu} servers and serverless threads,” in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 495–514.
- [101] L. Zeng, P. Huang, K. Luo, X. Zhang, Z. Zhou, and X. Chen, “Fograph: Enabling real-time deep graph inference with fog computing,” in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1774–1784.

- [102] L. Toader, A. Uta, A. Musaafr, and A. Iosup, “Graphless: Toward serverless graph processing,” in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, IEEE, 2019, pp. 66–73.
- [103] T. Bilot et al., “Sometimes simpler is better: A comprehensive analysis of {state-of-the-art}{provenance-based} intrusion detection systems,” in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 7193–7212.
- [104] F. Dong et al., “{Distdet}: A {cost-effective} distributed cyber threat detection system,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6575–6592.
- [105] OpenScap, *Open source security compliance solution*, <https://www.open-scap.org/>.
- [106] OpenVAS, *Openvas - open vulnerability assessment scanner*, <https://www.openvas.org>.
- [107] osquery, *Osquery: A sql powered operating system instrumentation, monitoring, and analytics framework*. <https://github.com/osquery/osquery>.
- [108] G. Chen et al., “Language-augmented symbolic planner for open-world task planning,” *arXiv preprint arXiv:2407.09792*, 2024.
- [109] Y. Ding et al., “Integrating action knowledge and llms for task planning and situation handling in open worlds,” *Autonomous Robots*, vol. 47, no. 8, pp. 981–997, 2023.
- [110] C. Phillips and L. P. Swiler, “A graph-based system for network-vulnerability analysis,” in *Proceedings of the 1998 workshop on New security paradigms*, 1998, pp. 71–79.
- [111] Z. Hu, R. Beuran, and Y. Tan, “Automated penetration testing using deep reinforcement learning,” in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2020, pp. 2–10.
- [112] J. F. Loevenich and R. R. F. Lopes, “Agentic generative ai for automation of cyber security attack chains in tactical manets,” in *2025 IEEE 50th Conference on Local Computer Networks (LCN)*, IEEE, 2025, pp. 1–7.
- [113] J. F. Loevenich, E. Adler, T. Hürten, F. Spelter, D. Roncevic, and R. R. F. Lopes, “Automating cyber threat intelligence and attack chain generation using cyber security knowledge

- graphs and large language models,” in *2025 International Conference on Military Communication and Information Systems (ICMCIS)*, IEEE, 2025, pp. 1–10.
- [114] C. Sarraute, O. Buffet, and J. Hoffmann, “Pomdps make better hackers: Accounting for uncertainty in penetration testing,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, 2012, pp. 1816–1824.
 - [115] D. Miller, R. Alford, A. Applebaum, H. Foster, C. Little, and B. Strom, “Automated adversary emulation: A case for planning and acting with unknowns,” MITRE CORP MCLEAN VA MCLEAN, Tech. Rep., 2018.
 - [116] Q. Li, R. Wang, D. Li, F. Shi, M. Zhang, and A. Chattopadhyay, “Dynpen: Automated penetration testing in dynamic network scenarios using deep reinforcement learning,” *IEEE Transactions on Information Forensics and Security*, 2024.
 - [117] H. Holm, “Lore a red team emulation tool,” *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1596–1608, 2022.
 - [118] *Att&ck matrix for enterprise*, <https://attack.mitre.org/>.
 - [119] Online, *Explore atomic red team*, <https://atomicredteam.io/>, 2025.
 - [120] BishopFox, *Sliver*, Accessed: 2025-04-01, 2024.
 - [121] Fortra, *Software for adversary simulations and red team operations*, Accessed: 2025-04-01, 2024.
 - [122] Rapid7, *Metasploit documentation of meterpreter*, Accessed: 2025-04-01, 2024.
 - [123] *Ttp-based hunting*, <https://www.mitre.org/sites/default/files/2021-11/prs-19-3892-ttp-based-hunting.pdf>.
 - [124] T. M. Corporation, *Cve program mission*, <https://www.cve.org/>.
 - [125] N. V. Database, *Official common platform enumeration (cpe) dictionary*, <https://nvd.nist.gov/products/cpe>.
 - [126] *Msfvenom - metasploit unleashed*, <https://www.offsec.com/metasploit-unleashed/msfvenom/>.

- [127] FunnyWolf, *Viper*, Accessed: 2024-11-01, 2024.
- [128] *Turla (2023) overview*, <https://attacker.vals.mitre-engenuity.org/enterprise/turla/>.
- [129] J. Wei et al., “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
- [130] D. Hendrycks et al., “Measuring massive multitask language understanding,” *arXiv preprint arXiv:2009.03300*, 2020.
- [131] G. Husari, E. Al-Shaer, M. Ahmed, B. Chu, and X. Niu, “Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources,” in *Proceedings of the 33rd annual computer security applications conference*, 2017, pp. 103–115.
- [132] K. Satvat, R. Gjomemo, and V. Venkatakrishnan, “Extractor: Extracting attack behavior from threat reports,” in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021, pp. 598–615.
- [133] M. T. Alam, D. Bhusal, Y. Park, and N. Rastogi, “Looking beyond iocs: Automatically extracting attack patterns from external cti,” in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023, pp. 92–108.
- [134] M. R. Rahman, R. M. Hezaveh, and L. Williams, “What are the attackers doing now? automating cyberthreat intelligence extraction from text on pace with the changing threat landscape: A survey,” *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–36, 2023.
- [135] U. Kumarasinghe, A. Lekssays, H. T. Sencar, S. Boughorbel, C. Elvitigala, and P. Nakov, “Semantic ranking for automated adversarial technique annotation in security text,” in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 49–62.
- [136] G. Deng et al., “Pentestgpt: An llm-empowered automatic penetration testing tool,” *arXiv preprint arXiv:2308.06782*, 2023.
- [137] CISA, *Cybersecurity alerts advisories*, Accessed: 2024-07-01, 2024.
- [138] Symantec, *Symantec enterprise blogs threat intelligence*, Accessed: 2024-07-01, 2024.
- [139] A. I. G. -. U. of Basel, *Fast downwards*, Accessed: 2024-11-01, 2024.

- [140] S3N4T0R-0X0, *Apt attack simulation*, Accessed: 2025-03-01, 2024.
- [141] G. Lindemulder, *What are advanced persistent threats?* <https://www.ibm.com/think/topics/advanced-persistent-threats?regionCode=US&languageCode=en&contactmodule=true&cm-history=US-en>.
- [142] P. Network, *What is an advanced persistent threat?* https://www.paloaltonetworks.com/cyberpedia/what-is-advanced-persistent-threat-apt?utm_source=chatgpt.com.
- [143] A. Evaluations, *Machine-readable fn7 emulation plan*, https://github.com/attacker/ael/tree/main/Enterprise/fin7/Emulation_Plan/yaml.
- [144] J. Chen, S. Hu, H. Zheng, C. Xing, and G. Zhang, “Gail-pt: An intelligent penetration testing framework with generative adversarial imitation learning,” *Computers & Security*, vol. 126, p. 103 055, 2023.
- [145] M. C. Ghanem, T. M. Chen, and E. G. Nepomuceno, “Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks,” *Journal of Intelligent Information Systems*, vol. 60, no. 2, pp. 281–303, 2023.
- [146] Q. Wang et al., “You are what you do: Hunting stealthy malware via data provenance analysis,” Jan. 2020.
- [147] M. Friedman, “The use of ranks to avoid the assumption of normality implicit in the analysis of variance,” *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.
- [148] F. Wilcoxon, “Individual comparisons by ranking methods,” in *Breakthroughs in statistics: Methodology and distribution*, Springer, 1992, pp. 196–202.
- [149] S. Holm, “A simple sequentially rejective multiple test procedure,” *Scandinavian journal of statistics*, pp. 65–70, 1979.
- [150] S. Ö. Hacıoğlu, *Silver fox apt targets public sector via trojanized medical software*, <https://www.picussecurity.com/resource/blog/silver-fox-apt-targets-public-sector-via-trojanized-medical-software>.

- [151] C. Leigh, *Silver fox apt attack taiwan*, <https://westoahu.hawaii.edu/cyber/global-weekly-exec-summary/silver-fox-apt-attack-taiwan/>.
- [152] Z. Li et al., “Marlin: Knowledge-driven analysis of provenance graphs for efficient and robust detection of cyber attacks,” *arXiv preprint arXiv:2403.12541*, 2024.
- [153] X. Wang et al., “Opencua: Open foundations for computer-use agents,” 2025. arXiv: 2508.09123 [cs.AI].
- [154] Z. Jia, Y. Xiong, Y. Nan, Y. Zhang, J. Zhao, and M. Wen, “{Magic}: Detecting advanced persistent threats via masked graph representation learning,” in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5197–5214.
- [155] B. Jiang et al., “{Orthrus}: Achieving high quality of attribution in provenance-based intrusion detection systems,” in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 7173–7192.
- [156] Y. Zhu, L. Zhenyuan, Y. Wei, and S. Ji, “The case for learned provenance-based system behavior baseline,” in *Forty-second International Conference on Machine Learning*.

**BRIDGING GAPS IN OPERATIONAL CYBER DEFENSE: ADAPTIVE INTRUSION
DETECTION AND REALISTIC EVALUATION**

Approved by:

Yan Chen
Department of Computer Science
Northwestern University

Han Liu
Department of Computer Science
Northwestern University

Xinyu Xing
Department of Computer Science
Northwestern University

Zhenkai Liang
Department of Computer Science
National University of Singapore

Date Approved: May 22, 2026

APPENDIX A

CAPTAIN SUPPLEMENTARY MATERIALS

This appendix provides supplementary materials for CAPTAIN (Section 3.1), including additional dataset details, the tag-propagation and alarm-generation policies inherited from MORSE, mathematical proofs of the gradient derivations, and additional experimental results.

A.1 Attack Scenarios in Datasets

For benign scenarios, the SOC collaborating with us provided three detailed operation environments in the real world: Streaming simulates a data store and streaming server; Dev simulates a development server; Cloud simulates a cloud server. The simulated APT attack chains are generated using the Atomic Red Team [91]. We designed three types of simulated attacks: fully randomized, partially randomized, and deterministic attacks. Each step in a fully randomized attack is randomly selected from the Atomic Red Team library. A partially randomized attack has determined steps at some stages but randomly chooses steps for the rest of the steps. A deterministic attack has defined steps at every attack stage. We generated one deterministic and three partially randomized attacks in our evaluation. We conduct the experiments following the attack campaign pattern in DARPA Engagements to make it as realistic as possible. Table A.1 specifies the apps or processes running in each simulated environment.

Table A.1: Simulated Scenario Summary

Attack	Attack Description
ReverseShell(RS)	Connect to the victim's host, collect system information, and install multiple applications
WebShell(WS)	Connect to the victim's host, collect system information and modify system configurations
AttackChain(AC)	A randomly generated attack chain using Atomic Red Team [91]
Kimsuky(Kim)	A simulated North Korean APT Kimsuky
Chaos	A malicious payload that allows remote control
Benign	App or Process Involved
Streaming	Kafka, Mysql, Nginx, Redis, Zookeeper
Developing	Iptables, Zabbix, Gitlab, VSCode, Influxdb, Ixextend, redis-server, Qingtengyun, Baota, docker
Cloud	finalshell, postgres, web.py, Apache Struts 2, saltstack, Cloud Workload Protection Platforms

A.2 CAPTAIN Policies

We adopt tag propagation rules, detailed in Table A.2, from MORSE [17] for unbiased comparison. $dtag$ refers to the data tag and $ptag$ refers to the code tag. Please refer to the original paper for formal and detailed descriptions of the hyperparameters such as T_{qb} , T_{qe} , d_b , and d_e . All hyperparameters are set according to the recommendation in [17]. We also adopt the alarm generation rules from MORSE [17], detailed in Table A.3. $incl_exec(p)$ means p contains the execution permission. $socket(o)$ holds when o refers to a socket.

Table A.2: Tag propagation policies

Event	Tag to update	New tag value for different subject types		
		benign	suspect	suspect environment
$create(s, x)$	$x.dtag$	$s.dtag$	$s.dtag$	$s.dtag$
$read(s, x)$	$s.dtag$	$\min(s.dtag, x.dtag)$	$\min(s.dtag, x.dtag)$	$\min(s.dtag, x.dtag)$
$write(s, x)$	$x.dtag$	$\min(s.dtag + a_b, x.dtag)$	$\min(s.dtag, x.dtag)$	$\min(s.dtag + a_e, x.dtag)$
$load(s, x)$	$s.ptag$		$\min(s.ptag, x.itag)$	
	$s.dtag$		$\min(s.dtag, x.dtag)$	
$exec(s, x)$	$s.ptag$	$x.itag$	$\min(x.itag, susp_env)$	$x.itag$
	$s.dtag$	$(1.0, 1.0)$	$\min(s.dtag, x.dtag)$	$\min(s.dtag, x.dtag)$
$inject(s, s')$	$s'.dtag$		$\min(s'.dtag, s.itag)$	
	$s'.ptag$		$\min(s.dtag, s'.dtag)$	
$periodically:$	$s.dtag$	$\max(s.dtag, d_b * s.dtag + (1 - d_b) * T_{qb})$	no change	$\max(s.dtag, d_e * s.dtag + (1 - d_e) * T_{qe})$

Table A.3: Alarm generation policies

Name	Description	Operation(s)	Data integrity condition	Other conditions
MemExec	Prepare binary code for execution	$mmap(s, p), mprotect(s, p)$	$s.itag < 0.5$	$incl_exec(p)$
FileExec	Execute file-based malware	$exec(s, o), load(s, o)$	$s.itag < 0.5$	$s.ptag > 0.5$
Inject	Process injection	$inject(s, s')$	$s.itag < 0.5$	$s'.ptag > 0.5$
ChPerm	Prepare malware file for execution	$chmod(s, o, p)$	$s.itag < 0.5$	$incl_exec(p)$
Corrupt	Corrupt files	$write(s, o), mv(s, o), rm(s, o)$	$s.itag < 0.5$	-
Escalate	Privilege escalation	$any(s)$	$s.itag < 0.5$	changed userid
DataLeak	Confidential data leak	$write(s, o)$	$s.itag < 0.5$	$s.ctag < 0.5, socket(o)$
MalFileCreation	Ingress Tool Transfer	$create(s, o)$	$s.itag < 0.5$	$File(o)$

A.3 Mathematical Proof of the Optimization

In this section, we prove the equations of gradients of A , G , and T to the loss, i.e., Eq. 3.7, Eq. 3.9, and Eq. 3.10 in §3.1.4. According to the loss function Eq.3.4 and the chain rule in calculus, we have:

$$\begin{aligned}
\frac{\partial Loss}{\partial a_n} &= \sum_{e \in E} \frac{\partial \mathcal{L}(e)}{\partial f} \cdot \frac{\partial f}{\partial a_n} + \alpha \cdot (a_n - a_0) \\
\frac{\partial Loss}{\partial g_e} &= \sum_{e \in E} \frac{\partial \mathcal{L}(e)}{\partial f} \cdot \frac{\partial f}{\partial g_e} + \gamma \cdot (g_e - g_0) \\
\frac{\partial Loss}{\partial thr_e} &= \sum_{e \in E} \frac{\partial \mathcal{L}(e)}{\partial f} \cdot \frac{\partial f}{\partial thr_e} + \tau \cdot (thr_e - thr_0)
\end{aligned} \tag{A.1}$$

where $\frac{\partial \mathcal{L}}{\partial f}$ is identical for all events according to Eq. 3.3 and α , γ , and τ are pre-defined hyperparameters. We have

$$\begin{aligned}
f(e) &= tag_{dest}^{new} - thr_e \\
&= g_e \cdot tag_{rule} + (1 - g_e) \cdot tag_{dest} - thr_e \\
\frac{\partial f}{\partial thr_e} &= -1 \\
\frac{\partial f}{\partial a_n} &= g_e \cdot \frac{\partial tag_{rule}}{\partial a_n} + (1 - g_e) \cdot \frac{\partial tag_{dest}}{\partial a_n}
\end{aligned}$$

$f(e)$ is influenced by the propagation rate of the current event (denoted by g_e) and all previous events (denoted by $g_{e'}$). Consequently, every time when tag_{dest}^{new} is updated, we recalculate $\frac{\partial f}{\partial g_e}$ and $\frac{\partial f}{\partial g_{e'}}$.

$$\begin{aligned}
& \text{Since } \frac{\partial g_e}{\partial g_{e'}} = 0, \\
& \frac{\partial f}{\partial g_{e'}} = g_e \cdot \frac{\partial \text{tag}_{rule}}{\partial g_{e'}} + (1 - g_e) \cdot \frac{\partial \text{tag}_{dest}}{\partial g_{e'}}
\end{aligned} \tag{A.2}$$

The calculation of $\frac{\partial f}{\partial g_e}$ involves the product of two functions. According to the Product Rule:

$$\text{if } h(x) = f(x)g(x), \text{ then } h'(x) = f'(x)g(x) + f(x)g'(x)$$

we have

$$\frac{\partial f}{\partial g_e} = g_e \frac{\partial \text{tag}_{rule}}{\partial g_e} + \text{tag}_{rule} + (1 - g_e) \frac{\partial \text{tag}_{dest}}{\partial g_e} - \text{tag}_{dest} \tag{A.3}$$

A.4 Mathematical Proof of the Heuristic Hyperparameter Setting

According to Eq.A.1, to let $\frac{\partial \text{Loss}}{\partial a_n}$, $\frac{\partial \text{Loss}}{\partial g_e}$, and $\frac{\partial \text{Loss}}{\partial \text{thr}_e}$ be zero, we have

$$\begin{aligned}
\alpha &= - \sum_{e \in E} \frac{\partial \mathcal{L}(e)}{\partial f} \cdot \frac{\partial f}{\partial a_n} \cdot (a_n - a_0)^{-1} \\
\gamma &= - \sum_{e \in E} \frac{\partial \mathcal{L}(e)}{\partial f} \cdot \frac{\partial f}{\partial g_e} \cdot (g_e - g_0)^{-1} \\
\tau &= - \sum_{e \in E} \frac{\partial \mathcal{L}(e)}{\partial f} \cdot \frac{\partial f}{\partial \text{thr}_e} \cdot (\text{thr}_e - \text{thr}_0)^{-1}
\end{aligned} \tag{A.4}$$

Since $\mathcal{L}(e) = \max(0, (1 - f(e))^2 - 1)$ and $-1 < f(e) < 0$,

$$- \sum_{e \in E} \frac{\partial \mathcal{L}(e)}{\partial f} = \sum_{e \in E} 2(1 - f(e))$$

$\frac{\partial f}{\partial \text{thr}_e} = -1$, $0 < \frac{\partial f}{\partial a_n} < 1$, $-1 < \frac{\partial f}{\partial g_e} < 0$. As mentioned in §3.1.3.2, a_0 is usually 0 while g_0 is

1. We can make the following estimations:

$$\begin{aligned}
f(e) &\approx -0.5, \partial f / \partial a_n \approx 0.5, \partial f / \partial g_e \approx -0.5 \\
a_n - a_0 &\approx 0.5, g_e - g_0 \approx -0.5, \text{thr}_e - \text{thr}_0 \approx -0.25
\end{aligned} \tag{A.5}$$

Let $N(e)$ be the number of the allowed false alarms related to event e and $N(n)$ be the number

of the allowed false alarms related to node n .

$$\begin{aligned}
\alpha &= \sum_{e \in E} 2(1 - f(e)) \cdot \frac{\partial f}{\partial a_n} \cdot (a_n - a_0)^{-1} \approx 3N(n) \\
\gamma &= \sum_{e \in E} 2(1 - f(e)) \cdot \frac{\partial f}{\partial g_e} \cdot (g_e - g_0)^{-1} \approx 3N(e) \\
\tau &= \sum_{e \in E} 2(1 - f(e)) \cdot \frac{\partial f}{\partial thr_e} \cdot (thr_e - thr_0)^{-1} \approx 12N(e)
\end{aligned} \tag{A.6}$$

A.5 Additional Experiment Results

A.5.1 Gradients Storage During Training

We maintain two dictionaries for each node n to save the gradients of tag_n , $\nabla_n^A : \{n' : \frac{\partial tag_n}{\partial a_{n'}}, n' \in N\}$ and $\nabla_n^G : \{e : \frac{\partial tag_n}{\partial g_e}, e \in E\}$. Theoretically, the worst case for saving ∇_n^A is $O(|N|^2)$, and $O(|N||E|)$ for saving ∇_n^G . Please note that we only store non-zero gradients. Every time after the gradients are calculated, we add the non-zero values into ∇_n^A and ∇_n^G and discard the too-small gradients (e.g., $< 10^{-5}$). In practice, most gradients would always be zero for three reasons: 1) In most cases, the provenance graph is relatively sparse. For example, a specific process would only read a small set of files on the system; 2) The gradients only get updated when the event changes the tags. Therefore, most benign nodes and edges would not be added to ∇_n^A and ∇_n^G ; 3) As g_e is smaller than 1, some gradients would become near-zero after many iterations. Fig. A.1 clearly illustrates the long-tailed distribution of the non-zero gradients on the DARPA dataset, which means we only need to store a tiny number of non-zero gradients for most nodes. The average numbers are less than 5 for all those four datasets. Hence, the additional overhead is reasonable during training.

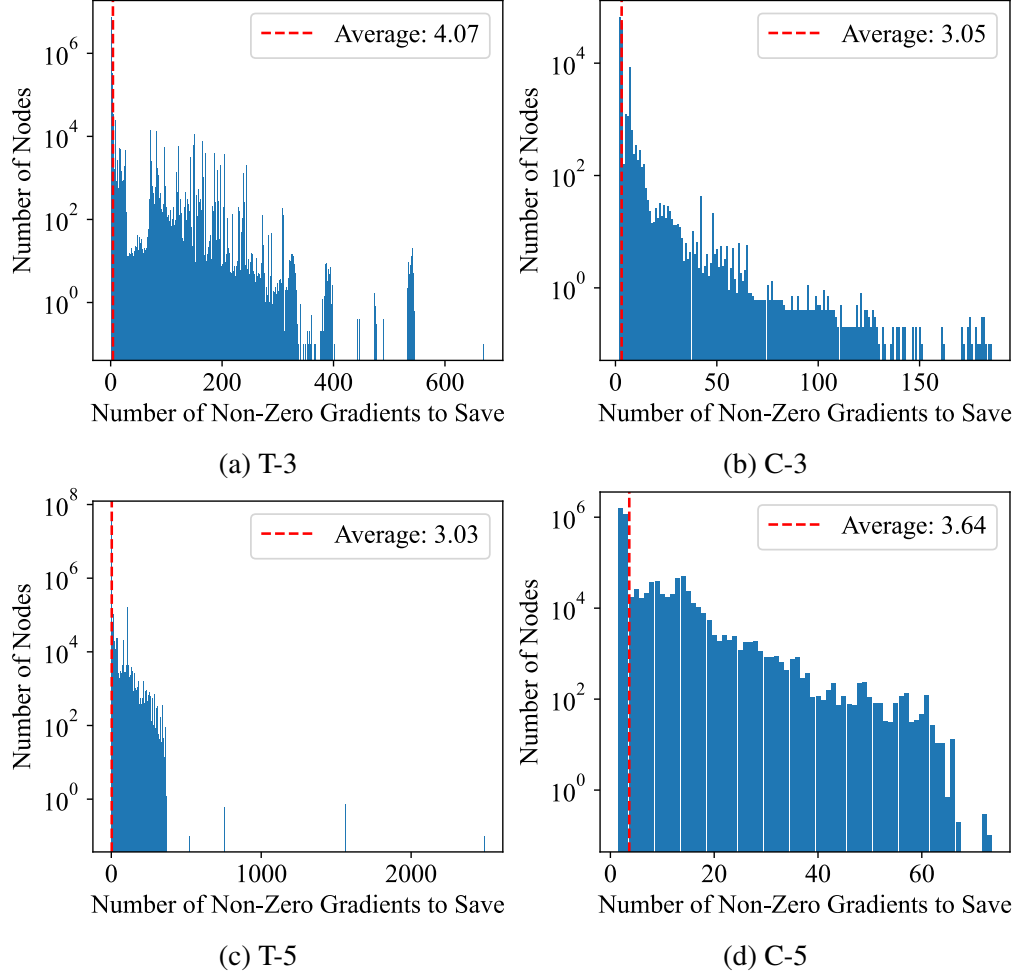


Figure A.1: The distribution of the numbers of non-zero gradients to A and G that each nodes need to save during tag propagation.

A.5.2 Mimicry Attack Experiment

We insert events into the dataset to mimic benign behaviors involving malicious entities. we used the E3 CADETS for this experiment. In the attack, `/tmp/test` file is downloaded and executed, serving as a command and control malware to carry out the rest of the attack. We identified a benign file `/dev/tty` to serve as the target to mimic and replicated interactions with `/dev/tty` on `/tmp/test` to let `/tmp/test` mimic normal behaviors.

A.5.3 Ablation Study on Learning Rate

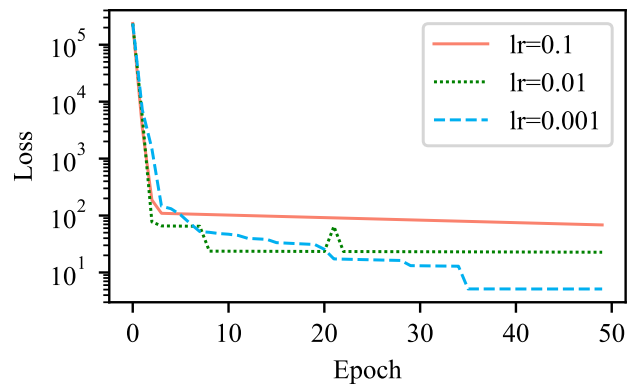


Figure A.2: Training loss using different learning rates.

Fig. A.2 shows the training loss with different learning rates on CADETS from DARPA Engagement 3. From the figure, if the training is limited to 20 epochs (due to time and resource issues), a learning rate of 0.01 is advisable; for longer training durations, 0.001 is likely a more preferred choice.

APPENDIX B

GRAPHFAAS SUPPLEMENTARY MATERIALS

This appendix provides supplementary materials for GraphFaaS (Section 3.3), including the subgraph partition and packing algorithm used to bound the size of each subgraph’s K -hop neighborhood while minimizing the total number of subgraphs sent to serverless workers.

B.1 Algorithm for Subgraph Partition & Packing

The goal of subgraph partitioning and packing is to divide a large graph into several smaller subgraphs such that the size of the K -hop neighborhood of each subgraph remains within a predefined threshold. At the same time, we aim to avoid excessive fragmentation by merging K -hop neighborhoods that share overlapping regions. In other words, we seek a partitioning of the original graph where each subgraph’s K -hop neighborhood is bounded in size, while also minimizing the total number of subgraphs. For each cluster k , we maintain its current node set $U_k = N_2(X_k)$, represented as a bitset, and its edge count $f_k = |E_G[U_k]|$. When a new vertex v is considered for insertion, we compute the increase in edge count as:

$$\Delta_k = |E_G[U_k \cup B_2(v)]| - f_k.$$

If Δ_k does not exceed the remaining capacity of cluster k , the vertex is inserted into that cluster. Otherwise, a new cluster is created.

To minimize the total number of clusters used, we adopt the *Best-Fit* strategy, which attempts to pack the vertex into the cluster where it fits most tightly. A simpler alternative is the *First-Fit*

strategy.

Algorithm 1: BinPack_FFD (First-Fit Decreasing Bin Packing) for Graph Partitioning

Input: List of neighborhoods *neighbors*, capacity *capacity*, strict flag *strict*

Output: List of bins and remaining capacities

```

1 Sort neighbors by number of edges in descending order;
2 Initialize bins as empty list;
3 Initialize remain as empty list;
4 Initialize subgraph as empty list of edge sets;
5 foreach (idx, neighborhood) in sorted items do
6   placed  $\leftarrow$  False;
7   edge_set_to_add  $\leftarrow$  set of edges in neighborhood;
8   for b  $\leftarrow$  0 to  $|bins| - 1$  do
9     margin  $\leftarrow$  size of edge_set_to_add minus overlapping edges with
       subgraph[b];
10    if strict is True then
11      if margin < remain[b] then
12        Add idx to bins[b];
13        Update subgraph[b] and remain[b];
14        placed  $\leftarrow$  True; break;
15      end
16    else
17      if margin  $\leq$  remain[b] then
18        Add idx to bins[b];
19        Update subgraph[b] and remain[b];
20        placed  $\leftarrow$  True; break;
21      end
22    end
23  end
24  if placed is False then
25    Create new bin with idx;
26    Add edge_set_to_add to subgraph;
27    Append capacity -  $|edge\_set\_to\_add|$  to remain;
28  end
29 end
30 return bins, remain;

```

APPENDIX C

AURORA SUPPLEMENTARY MATERIALS

This appendix provides supplementary materials for AURORA (Section 4.1), including details of the multi-plan search procedure used by the symbolic planner and additional experimental results on system costs and CTI report analysis.

C.1 Searching for Multiple Attack Chains using Symbolic Planners

To find as many attack chains as possible, we leverage the cost mechanism in symbolic planning. Initially, all actions have a cost of zero, and the algorithm searches for the plan with the lowest total cost in this action space (since all plans have a total cost of 0 at this stage, the search returns the first attack plan it discovers). After discovering a plan, we increase the costs of all actions within that plan by a fixed amount and update the domain. Then we search for the lowest-cost plan in the updated domain. Because the actions from the first plan now have higher costs, the planner favors creating new plans using previously unused actions. This iterative process continues until either no new plans are generated for several consecutive rounds or we reach our target number of generated plans.

C.2 Supplementary Experimental Results

Costs of AURORA: We assessed the time and monetary costs of AURORA. The results are shown in Table C.1. With LLMs, extracting attack actions from documentation and analyzing their pre-conditions and effects takes less than 10 seconds on average. The time required to analyze one

Table C.1: Average Time and Monetary Costs of AURORA

	Time	LLM Costs	
		Tokens	Cost (USD)
Attack Tool Analyzing ¹	9.8s	10.8K	0.029
CTI Report Analyzing ²	26.1s	16.8K	0.055
Reward Embedding ²	14.9s	95.945K	0.010
Attack Planning ³	0.5-2h	-	-
Attack Execution ³	9.3m	-	-

¹ per attack action. ² per report. ³ per attack chain (50 actions on avg.).

Table C.2: The Performance Comparison of Attack Technique Extraction between LADDER and AURORA

id	Recall		Precision		Correctness	
	Base	Ours	Base	Ours	Base	Ours
1	61.5%	100.0%	42.1%	100.0%	52.6%	84.6%
2	28.6%	71.4%	17.4%	100.0%	30.4%	100.0%
3	20.0%	100.0%	5.6%	28.6%	22.2%	80.0%
4	28.9%	88.1%	43.3%	97.2%	43.3%	100.0%
5	33.3%	93.3%	41.7%	100.0%	46.2%	100.0%
6	58.3%	62.5%	35.9%	93.8%	43.6%	100.0%
7	17.1%	38.3%	25.0%	78.6%	42.9%	80.0%
8	19.0%	76.2%	25.0%	88.9%	37.5%	94.4%
9	25.0%	100.0%	11.8%	53.8%	35.3%	84.6%
10	46.2%	84.6%	22.2%	84.6%	33.3%	100.0%

CTI report for attack customization is approximately 40 seconds. Planning time varies with the length of the attack chain, taking approximately 0.5 to 2 hours for a 50-step chain. Executing an emulated attack takes ten minutes on average. We also evaluate the cost of using LLM. The result shows that AURORA is economical in terms of both time and money.

Extracting TTPs from CTI Reports: We evaluate the accuracy of extracting TTPs from CTI using reports from CISA [137] as the ground truth. We employ LADDER [133] as the baseline. As shown in Table C.2, the LLM-based report analyzer of AURORA achieves higher precision than

the baseline, nearing or exceeding 80% for almost all reports. And the average recall is 69%. The result also shows that AURORA provides a more accurate description for each individual technique than LADDER. After examining the results, we identified two key advantages of using LLM. First, LLMs have better capabilities in understanding long sentences. Conversely, LADDER tends to be distracted by irrelevant words in filenames or positions. Second, LLM provides more comprehensible descriptions for the extracted TTPs. In contrast, LADDER sometimes yields incomplete sentences.

We also analyze the reasons for failure cases. First, LLM makes mistakes when some descriptions are ambiguous. For instance, a report mentions that *affiliates communicate with victims via TOR, Tox, email, or encrypted applications*. However, the LLM incorrectly identified this sentence as evidence of a command and control technique during the attack. Second, LLMs sometimes fail to enumerate all TTPs exhaustively. For instance, a report mentions that the attacker *impersonates company IT and/or helpdesk staff to gain trust and obtain credentials*. While the LLM successfully identifies Impersonation (T1656), it fails to recognize that this also demonstrates the attacker is gathering victim identity information (T1589).

VITA

Lingzhi Wang is a Ph.D. candidate in Computer Science at Northwestern University, advised by Prof. Yan Chen. His research lies at the intersection of artificial intelligence and real-world cybersecurity, with a focus on AI-driven systems for intrusion detection, attack emulation, and the construction of realistic security benchmarks. During his Ph.D., he held research internships at SRI International, the University of Southern California's Information Sciences Institute, and Obsidian Security. He has published at venues including NDSS, IEEE TDSC, AsiaCCS, ACNS, and NeurIPS workshops. He obtained his Bachelor's degree from Tsinghua University in 2020.