UNIVERSITÉ DE RENNES

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

H D R T H E S I S

to obtain the title of

HDR of Science

of the Université de Rennes

Specialty : Computer Science

Defended by

Pierre-François Gimenez

Contributions to Anomaly-based Intrusion Detection Systems Through Data Representation, Data Quality and Data Generation

Jury :

Reviewers :	Isabelle Chrisment	-	Université de Lorraine
	Hervé Debar	-	Télécom SudParis
	Jilles Vreeken	-	CISPA
Examinators :	Marc Dacier	-	KAUST
	Philippe Owezarski	-	LAAS-CNRS

Acknowledgments

I would like to thank my reviewers, Isabelle Chrisment, Hervé Debar and Jilles Vreeken, for eagerly accepting to review my work. I also thank the other members of the jury, namely Marc Dacier and Philippe Owezarski.

Research is a social endeavor, and the work presented in this dissertation is a team effort. Therefore, I would like to thank the Ph.D. students I co-supervised: Adrien, Antoine, Fanny, Grégor, Maxime, Solène, and Vincent, and my co-authors: Alexandre, Aliénor, Barbara, Éric, Frédéric, Gilles, Grégory, Hélène, Jean-François, Jérôme, Jonathan, Joscha, Laurent, Lénaïg, Ludo, Malcolm, Michel, Mohamed, Thomas, Tomás, Valérie, Vincent, and Yufei, and more broadly everyone from the CIDRE and PIRAT teams. I learned a lot from working with you, both scientifically and personally. I would like to thank again Ludo and Valérie who dedicated a lot of their time and energy to making me the researcher I am today. I grew a lot thanks to your advice and support. I would also like to especially thank Frédéric and Grégory for our discussions on IDS evaluation and dataset quality. They deeply influenced my research project.

But there is more to life than just work, so I would like to also thank my friends for all the good time fencing on stage, enjoying various role-playing games, and creating weird things together. This dissertation is dedicated to the ones who will actually come to my defense (checkmate!).

Finally, I would like to thank my family for their support during all the years, as well as my partner Manon for all our precious moments together and your continuous encouragement in everything I undertake.

No AI assistant has been involved in the writing or editing of this dissertation. Errors are my own.

Contents

Chapter 1

Introduction

This document is a synthetic description of my scientific contributions since 2018. At that point, I defended my Ph.D. [67] on PAC (probably approximately correct) preference learning and recommendation using statistical models, and I decided to pivot to cybersecurity[1] for more applied research. In 2018, I joined the LAAS-CNRS laboratory in Toulouse, France, as a post-doctoral researcher in the TSF[2] team that specialized in dependability and cybersecurity. In 2020, after two years of postdoc, I joined Inria's CIDRE research team in Rennes as an assistant professor. In 2024, CIDRE split into two smaller teams, SUSHI and PIRAT, and I was recruited as an Inria's researcher in the PIRAT team.

Since the start of my Ph.D. in 2015, I also taught at various universities and engineering schools: at Université Paul Sabatier, UPSSITECH and INSA Toulouse from 2015 to 2020, and at CentraleSupélec since 2020, for a total of about 880 hours.

As a postdoc researcher specialized in machine learning, I worked mostly on intrusion detection, a research field that heavily benefited from the surge of deep learning. The TSF team focuses on embedded systems security, i.e., the security of small appliances with limited computing power such as airplane entertainment systems or "smart devices" deployed in the Internet of Things (IoT). My research on intrusion detection broadened to the security of IT networks when I joined CIDRE in 2020, covering alert explainability, dataset quality and synthetic dataset generation.

1.1 Context

In the ANSSI "Cyber threat overview" from 2025[3], we learn that there was in France 3,586 security events for which ANSSI incident response team were mobilized, and that the number of attacks is expected to increase in next years due to the surge of generative AI that can be used to identify vulnerabilities in software and websites, and to exploit them automatically at a large scale.

The goal of cybersecurity is to protect the confidentiality, integrity and availability of data against malicious attacks. The NIST, in its Cybersecurity Framework[4], proposes five functions of cybersecurity: 1) identify assets and vulnerabilities, 2) protect with policies to prevent attacks, 3) detect attacks that were not prevented, 4) respond to limit the impact of an attack, and 5) recover to restore any capabilities that were impacted by the attack.

My work focuses on the "Detect" function. More precisely, we can decompose this function into three main steps: i) data collection, ii) detection engine, and iii) alert investigation. Due to the success of machine learning and deep learning techniques in detection engines, the scientific

[1] Then simply called "computer security" before, for some reason, the prefix "cyber" became fashionable again.
[2] Now part of the TRUST team.
[3] https://cert.ssi.gouv.fr/cti/CERTFR-2026-CTI-003/
[4] https://www.nist.gov/cyberframework/csf-11-five-functions

literature generally focuses on improving this step. In the following, I will mostly refer to these systems as Intrusion Detection Systems (IDS)[5].

There exist two main categories of IDS: misuse-based (that identify malicious behavior) and anomaly-based (that model legitimate behavior). Misuse-based detectors generally rely on hand-crafted signatures or supervised machine learning models that have been fitted on benign and attack examples. While still very popular in the industry, they have a significant drawback: they cannot (by design) detect undocumented attacks, which are more and more common with LLM-powered vulnerability search. Anomaly-based detectors aim at detecting such undocumented attacks. By modeling the legitimate behavior, they can detect attacks as deviations from the expected behavior. They have a few drawbacks as well: legitimate but rare events can also produce alerts, and their alerts are more difficult to investigate because the expert does not know what the malicious behavior is, only that something is not normal. These drawbacks severely limit the applications of anomaly-based intrusion detection in the industry.

Alert explainability has been a topic of my research since my postdoc, as it can alleviate false alerts which are a huge problem in operational settings. This explainability also means that we could more precisely understand when our systems produce false positives and why. This research led to the realization that the quality of an intrusion detection system could be only as good as the quality of the training and testing data, and that public cybersecurity datasets suffer from many errors and biases. Therefore, I extended my research to the question of synthetic dataset generation, i.e., how to use machine learning to generate datasets of (hopefully) good quality. This journey will be the focus of this dissertation.

1.2 Scientific Responsibilities

Ph.D. Students Supervision In the past years, I was the co-supervisor of several Ph.D. students:

- Adrien Schoen on network flow generation (directed by Ludovic Mé, and with Yufei Han, Frédéric Majorczyk and Éric Totel), who defended in 2024,
- Maxime Lanvin on unsupervised network intrusion detection (directed by Ludovic Mé, and with Grégory Blanc, Yufei Han and Frédéric Majorczyk), who defended in 2024,
- Vincent Raulin on dynamic malware analysis (directed by Valérie Viet Triem Tong, and with Yufei Han), who defended in 2025,
- Grégor Quetel on intrusion detection system by semantics inference based on instrumented parsers (directed by Laurent Pautet, and with Thomas Robert), whose defense is planned for 2026,
- Fanny Dijoud on system intrusion detection (directed by Michel Hurfin, and with Frédéric Majorczyk and Barbara Pilastre), whose defense is planned for 2026,

[5]The literature traditionally distinguishes IDS (responsible for raising elementary alerts) and SIEM (for alert correlation), but in recent years the term IDS encompasses more and more the functions generally devolved to SIEM. In this dissertation, I use this extended definition of IDS.

- Solène Delourme on automatic detection and response in SOC with agentic AI (directed by Jean-François Lalande, and with Tony Quertier), who started in 2025, and
- Antoine Cellier on synthetic network and system data generation (directed by Ludovic Mé, and with Gilles Guette and Frédéric Majorczyk), who started in 2025.

Starting in Fall 2026, I will also co-supervise:

- Atieh Atieh (with Corentin Larroche, Barbara Pilastre and Michel Hurfin) on the application of graph foundation model to cybersecurity, with a funding from ANSSI, and
- Rayen Doudech (with Valérie Viet Triem Tong and Romain Deveaud) on spam detection, as a CIFRE Ph.D. student with OVHcloud.

Project Management From 2022 to 2025, I led the associate team SecGen between Inria and CISPA, focused on generating synthetic security data for intrusion detection system assessment. I am also a member of French national projects on supervision (Superviz), malware analysis (DefMal) and vulnerability search (REV).

In 2026, I obtained a scientific chair called "SYNTHETIC" at the SequoIA Cluster[6]. This project aims to enhance synthetic network traffic generation using transfer learning to address limitations in current methods, which struggle to adapt to different network descriptions. By leveraging unsupervised learning for benign traffic and style transfer for malicious traffic, SYNTHETIC enables the creation of parametrized datasets that can simulate gradual or sudden distribution shifts, improving the evaluation of intrusion detection systems (IDS) and their robustness over time. Additionally, this approach enhances the realism of cyber ranges and honeypots by generating adaptable background traffic, making training scenarios more effective and transferable to real-world cybersecurity challenges.

Scientific Events Organization Since 2025, I co-organize the ANUBIS[7] workshop co-located with ESORICS. This workshop is sponsored by the Superviz project of PEPR Cybersécurité. In the face of the humongous volume of publications in the field of intrusion detection and response, coupled with the lack of rigorous evaluation methodology of these (increasingly AI-based) methods, reproducibility is close to impossible. To remediate to that issue, ANUBIS offers the opportunity for researchers from different domains and communities to bring and discuss their evaluation methodology practices.

Since 2020, I have co-organized seminars every two weeks for the CIDRE[8] and PIRAT[9] teams. In 2021, I organized the event "Hands-on Machine Learning for Security"[10] that was labeled by the GDR Sécurité Informatique.

Collective Responsabilities I participated in several cybersecurity conference program committees (ISSRE, EICC, DSN, ESORICS), artificial intelligence conference program committees

[6] https://cluster-sequoia.univ-rennes.fr/
[7] https://superviz.inria.fr/anubis25/, https://superviz.inria.fr/anubis26/
[8] https://team.inria.fr/cidre/research/seminars/
[9] https://team.inria.fr/pirat/pirat-biweekly-seminars/
[10] https://team.inria.fr/cidre/hands-on-machine-learning-for-security/

(ECAI, AAAI, IJCAI) and national events program committees (THCon, RESSI), and I reviewed for established journals such as IEEE Network, IEEE Transactions on Information Forensics and Security, or for Computers & Security. I am a member of the Technical Committee of the PTCC ("Programme de Transfert au Campus Cyber"), and I reviewed projects for the PTCC, the ANR ("Agence Nationale de la Recherche") and the ANRT ("Association Nationale de la Recherche et de la Technologie").

I was an examiner for several Ph.D. defenses: Léa Kenmogne (2026, Inria Grenoble), Gabin Noblet (2026, LAAS-CNRS), Hamdi Friji (2024, Télécom SudParis), Léo Lavaur (2024, IMT Atlantique), Mohamed El Bouazzati (2023, Université Bretagne Sud), and Malcolm Bourdon (2021, LAAS-CNRS).

Dissemination Since 2024, I disseminated my work and my Ph.D. students' work 76 times at various events, ranging from conferences to seminars[11]. From 2021 to 2024, I created and managed the CIDRE team's YouTube channel. It contains 48 videos and has cumulated 715 hours of views.

Software Development I published several open-source software and libraries, and I actively develop the software Fos-R[12] mentioned in Chapter 4, including the reimplementation of work initially developed by Ph.D. students.

1.3 Research Objectives

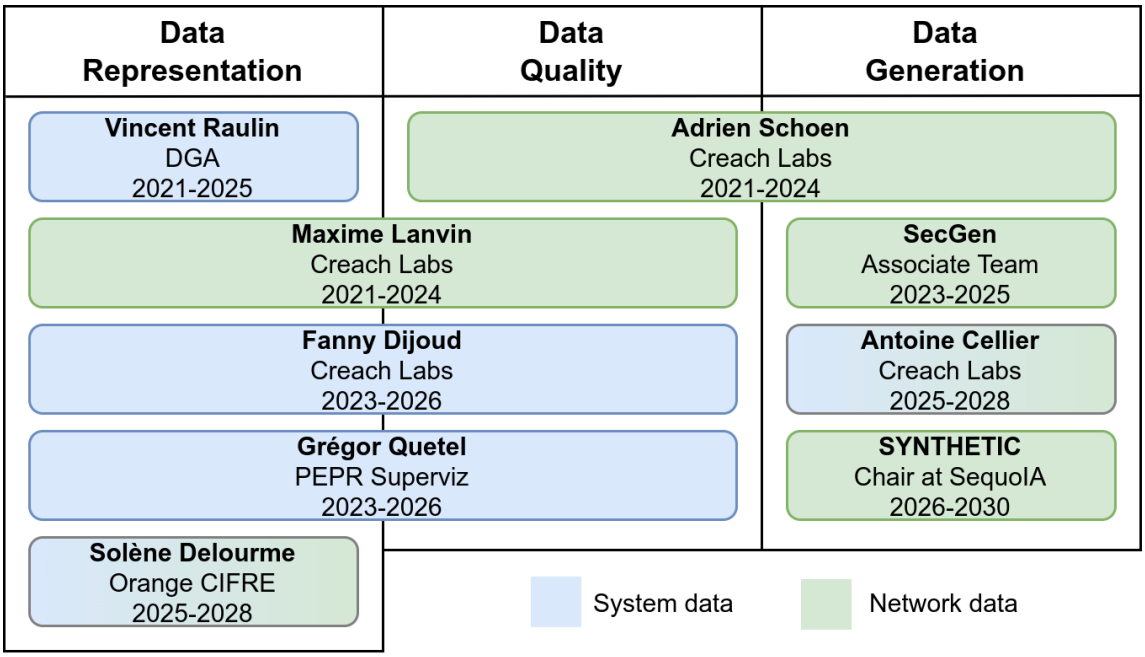


Figure 1.1: Topics of the Ph.D. students I co-supervise and of my projects

[11] Complete list on https://pfgimenez.fr/talks/
[12] https://gitlab.inria.fr/pirat-public/Fos-R

My overall research objective is to address the fundamental issue of **the explainability of anomaly-based detection** and to make these detectors **more applicable to real-world scenarios**. My research objectives can be summarized as follow.

RO1. Enhance explainability and detection accuracy with data representation.

RO2. Ensure dataset quality and fair intrusion detection system evaluation.

RO3. Generate high-quality synthetic security data for better IDS evaluation.

So, in short, these research objectives can be summarized as data representation, data quality and data generation, be it network or system data. The Ph.D. students and the projects described earlier are mapped to these research objectives in Fig. 1.1.

1.4 Other Scientific Contributions

Several contributions won't be presented in the manuscript. First, my collaboration with Éric Alata on a theoretical framework on injection-based vulnerabilities based on formal languages. This collaboration led to three articles: one published in the Theoretical Computer Science journal [7] and two at the LangSec workshop at IEEE Security & Privacy [8, 1].

Second, I also have a few contributions on the security of AI [41, 22, 19].

Finally, I published several articles with my Ph.D. supervisors on the subject of preference learning, during my Ph.D. and more recently, notably at ECAI [37], AAAI [65] and IJCAI [30], as well as a few other venues [69, 66, 60]. While this work is not related to security, it offers an opportunity to remain in the statistical AI academic community, which is still very relevant to my current work in data generation. For example, I learned about the minimum description length principle during my stay at CISPA in 2022, and it was through contributions [37, 30] on preference learning that I learned to work with it. This principle was later reapplied in cybersecurity-oriented publications [25, 10].

1.5 Structure of the document

This document is structured in a mostly chronological manner and follows the previous research objectives. Chapter 2 presents some of my contributions on data representation for intrusion detections systems. Chapter 3 focuses on explainability for intrusion detection, dataset quality and dataset creation. Chapter 4 details several work on synthetic network traffic generation. Conclusion and future works are listed in Chapter 5.

Three representative articles [3, 38, 10] mentioned in this dissertation are available in the appendix.

Chapter 2

Data Representation for Intrusion Detection

2.1 Introduction

All detection relies on observations of the system to protect. Such observations are made with sensors[1] that can be broadly categorized as network and host sensors. Network sensors monitor communications between hosts. Their advantage is that the vast majority of attacks are made remotely and involve network activity. However, communications are generally encrypted (about 95% of Web communications are encrypted[2]) so the actual content of the communication cannot be observed. Host sensors are located within a host (a computer, a smartphone, etc.) and range from monitoring the whole system to a single application. Their advantage is that they can monitor plain (i.e., non-encrypted) data. However, their implementation varies across OS (Linux, Windows, Android, etc.) and not every system can afford to host a sensor (such as low-cost embedded systems used in the Internet of Things).

Since its inception in 1980 [81], the field of intrusion detection has expanded significantly. As mentioned in the introduction, two main categories of detectors exist: misuse-based detectors, that directly model attacks, and anomaly detectors, that model the nominal operation of a system and define attacks as a deviation from such operation. Both categories are complementary: misuse-based detectors are very effective at detecting documented attacks, while anomaly detectors are necessary to detect undocumented attacks performed by resourceful attackers.

Anomaly-based detectors can be split into two subcategories. Specification-based anomaly detectors rely on a formal specification of how the system should behave, and behavioral-based anomaly detectors approximate the nominal operation from its typical operation, i.e., from empirical observations. Specification-based anomaly detectors are rarely used due to the ever-increasing complexity of IT systems. In the last decades, behavioral-based anomaly detectors significantly benefited from artificial intelligence techniques[3], notably deep learning.

Machine learning can be broadly described as the learning (or training) of a model from observations, such that it can be used afterward to solve a task. The training part is performed on a set of data called the train set[4]. The evaluation of the learned model is performed with another, disjoint set of data called the test set. For anomaly-based detectors, the train set only contains observations of legitimate behavior, while the test set includes both legitimate and

[1] To my knowledge, the vocabulary is not fully standardized. In this dissertation, I differentiate the "probe" (that contains the observation analyzer) from the "sensor" (that only measures), as in RFC 4766. Other documents, like RFC 4765, may call "sensor" the analyzer.

[2] https://transparencyreport.google.com/https/

[3] In this dissertation, "artificial intelligence" denotes the whole domain, not the restricted, LLM-focused, meaning it can have in 2026.

[4] Sometimes, a "validation set" is extracted from the train set, but it is outside the scope of this dissertation.

malicious behavior. In the following, the term "dataset" will denote a pair of train and test sets (typically collected from the same environment with an assumption of stationarity).

A popular deep learning model in anomaly detection is the autoencoder [71]. Because the rest of my work rely on this class of models, we will introduce them briefly[5].

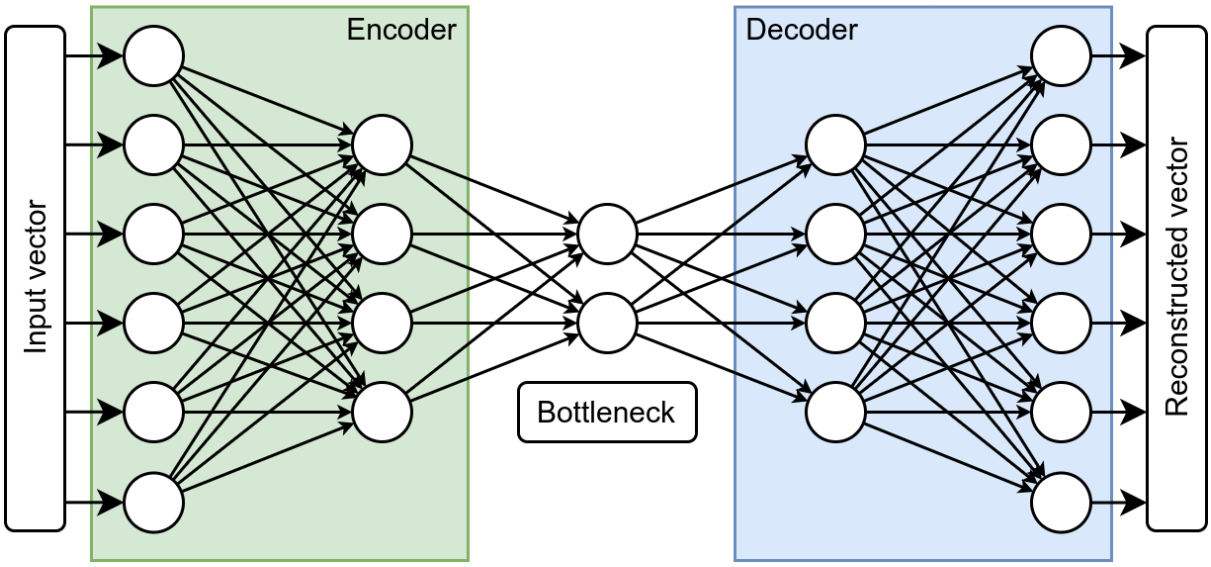


Figure 2.1: Schematic of an autoencoder

The architecture of an autoencoder, as shown in Fig. 2.1, can be split into two parts: the encoder, which processes the input vector and outputs into a vector space with a smaller dimension, called the *latent space*, and the decoder, which processes vectors from the latent space and outputs into the feature space. During training, the autoencoder is fed vectors from the train set and is optimized to minimize the distance between the input and the output vectors. In that regard, a perfect autoencoder would simply represent the identity function. The trick is that the latent space is smaller than the feature space, so the encoder must compress the input vector with minimal loss and the detector must decompress it. To achieve this compression, the model approximates the manifold on which the train set is located by taking advantage of the structure of the data, such as correlations and more complex, non-linear relations.

The idea behind using autoencoders for anomaly detection is that attacks probably do not live on the same manifold as the legitimate traffic. In that case, the assumptions about the data that have been implicitly learned during the compression and decompression steps won't be verified, and the distance between the input and the output of the autoencoder should be larger than it is for legitimate data. Whether this intuition is theoretically grounded or not, it works empirically.

The first research objective concerns the *representation of data*, i.e., the transformation of observations to vectors that can be processed by autoencoders. This chapter will present three contributions on this topic, on three different kinds of data: system logs, parse trees and radio waves.

[5] Autoencoders have many variations that are outside the scope of this dissertation and won't be presented.

2.2 Provenance Graph Representation with GRAAL

In the context of the Ph.D. of Fanny Dijoud, we proposed GRAAL, an anomaly-based IDS that relies on heterogeneous provenance graphs. GRAAL offers a multi-level monitoring of system activity leveraging transfer learning. This work has been published at EuroS&P in 2026 [3] and is reproduced in the appendix on page 50.

This work builds upon the provenance graph representation [77] of system logs, where each node represents a system resource (e.g., a process, a file, a registry key) and each edge represents an action (e.g., process creation, file reading, registry key addition), as shown in Fig. 2.2.

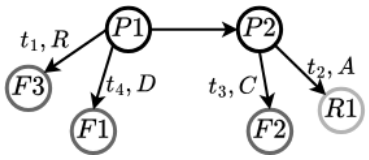


Figure 2.2: A provenance graph, with 3 different node types: process (P), files (F) and registry (R); and with 4 different action types: create (C), read (R), delete (D) and add (A). Each action is associated to a timestamp t_i. Figure from [3].

The methods proposed in the literature [26, 24, 29, 15, 9] have several limitations. They generally have poor scalability and limited explainability, which can severely limit their applicability outside the lab. To tackle these issues, we chose to rely on a simple autoencoder that has better scalability[6] and to rely on the representation of the data for achieving good performances. A novelty of the approach is its use of two models: a *global* model that monitors all resources at once and a *local* model that monitors each process independently.

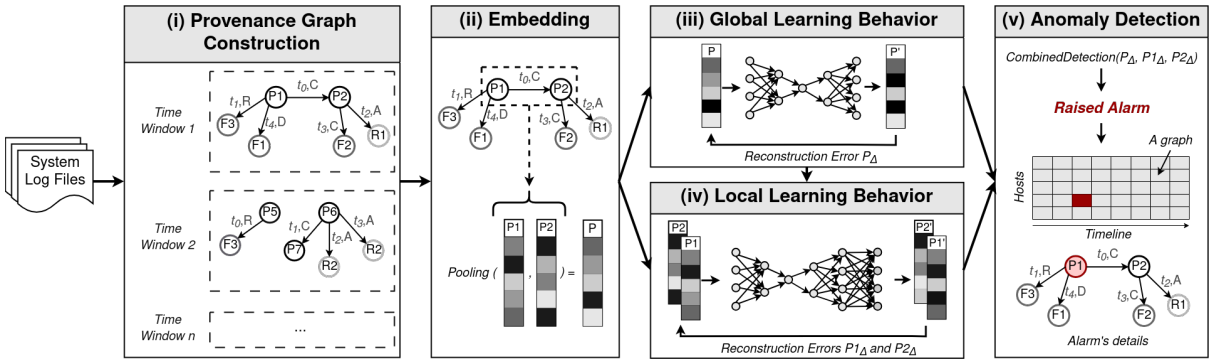


Figure 2.3: Overview of the GRAAL framework. Figure from [3].

The resulting framework, called GRAAL, is presented in Fig. 2.3. It consists of five steps. Step (i) is the creation of the provenance graph from the system log files, similarly to the rest of the literature. It creates graphs using 15-minute windows to coincide with the literature (this duration has little impact on the performances). In step (ii), a set of vectors is extracted from each graph. This step is further explained below. In step (iii), GRAAL uses a *global* autoencoder that produces a reconstruction error for an entire graph. In step (iv), GRAAL uses

[6] Small autoencoders can typically be run on CPU and do not require expensive equipment.

a *local* autoencoder that produces a reconstruction error per process node present in the graph. All these reconstruction errors are taken into account in step (v) that raises an alert if the global model and at least one local model raise an alert.

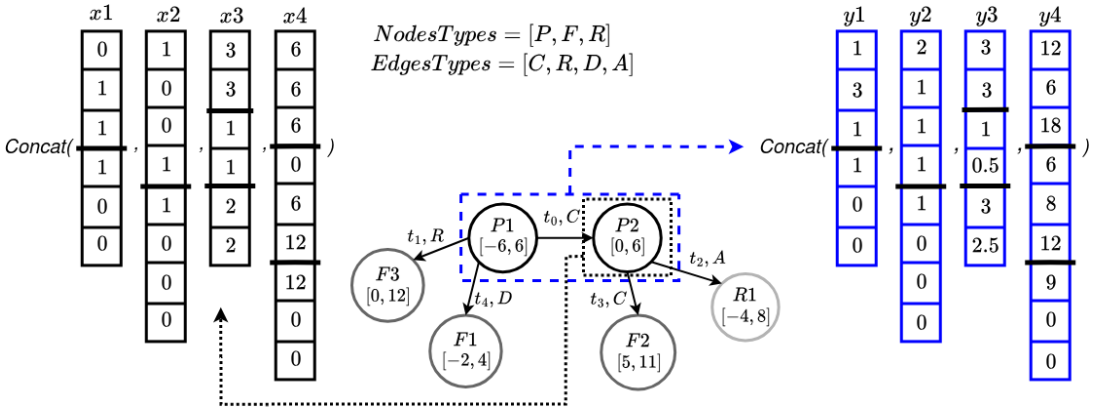


Figure 2.4: Example of the encoding of the structural part of the vectors. The encoding for the graph vector is framed in blue $(y1, y2, y3, y4)$. The encoding for the entity vector is framed in black $(x1, x2, x3, x4)$. Figure from [3].

For brevity's sake, let us focus on step (ii). Each vector has two parts: the "structural" part, which entails the local topological neighborhood of the nodes, and the "attribute" part, which entails the attributes associated to specific nodes (such as the command path of a process). The attribute part is created with classical embedding methods. The structural part is presented in Fig. 2.4. Let us first focus on the embedding of individual nodes, on the left part of the figure, and more precisely of the Process nodes that are the actors of all actions on the system. The embedding of a Process node consists of the concatenation of four vectors, denoted $x1$ to $x4$.

x1. The vector *x1* contains information about the entity type of the nodes present in the one-hop neighborhood of the considered node. In Figure 2.4, the graph is composed of nodes related to three different entity types: process (P), file (F) and registry (R). In the first half of the vector, we keep the number of neighbors of each type that are the targets of an outgoing edge. Symmetrically, the second half contains the number of neighbors of each type that are the sources of an incoming edge.

x2. The vector *x2* is structured in the same way as *x1* but contains the cardinality of outgoing and incoming edge types connected to the node. In the example of Figure 2.4, the graph is composed of four edge types: create (C), read (R), delete (D) and add (A). The first half of the vector focuses on edges where the entity is an actor and the second half focuses on edges where the entity is an object.

x3. The vector *x3* is split in three and contains information on degrees of the process node: degree, inbound degree and outbound degree. In an entity vector, the three values are duplicated. As explained afterward, this is not the case for a global vector.

x4. The vector *x4* contains information on the lifetime of the entity and the lifetime of its one-hop neighborhood, i.e., how for long they exist. For brevity's sake, I will not detail this vector as it has limited impact on the performances.

To compute the structural part of the global vector of a graph, one simply averages the structural parts of all the Process nodes of this graph. Using an average ensures that the

ranges of values are similar between the local and global vectors and allows the transfer learning between the local and global models. Indeed, once the global model is learned, the weights of its autoencoder are frozen and reused in the local model, which simply trains another layer of neurons after the output of the global model.

In conclusion, GRAAL shows that it is possible to have high detection performances using a lightweight model: see Table 2.1 for the global model and Table 2.2 for the local model. GRAAL was the most scalable model we evaluated: it could be deployed on each computer to detect attacks with minimal latency and overhead. It also means that the models could be tailored to specific hosts: for example, the system activity of a graphist or a programmer might differ.

Dataset	IDS	Precision	Recall	F1-score	AUC
Stream-spot	GAE*	0.63	1.00	0.77±0.00	**0.40**±0.00
	Flash*	1.00	0.95	0.97±0.00	**0.96**±0.00
	Magic*	1.00	1.00	1.00±0.00	**1.00**±0.00
	GRAAL	1.00	0.95	0.97±0.00	**1.00**±0.00
Wget	GAE*	0.50	1.00	0.67±0.00	**0.56**±0.03
	Flash*	0.28	0.36	0.32±0.35	**0.43**±0.31
	Magic*	0.94	0.94	0.94±0.01	**0.97**±0.02
	GRAAL	0.90	0.94	0.92±0.01	**0.96**±0.02
OpTC-6hosts	GAE*	0.11	0.85	0.19	**0.53**
	Kairos*	0.40	0.31	0.35	**0.63**
	Magic*	0.12	0.55	0.19	**0.55**
	GRAAL	0.98	0.40	0.57	**0.74**

Table 2.1: Global-level detection comparison (*thresholds are biased, leading to optimistic performances) (± correspond to the 95% confidence interval). Table from [3].

Host	IDS	TP	FP	TN	FN	Precision	ADP
201	**Orthrus**	1	0	18.6k	519	**1.00**	**0.99**
	Velox	2	7	18.6k	518	0.22	**0.39**
	Flash*	4	143	18.5k	516	0.03	**0.32**
	GRAAL	2	0	18.6k	518	**0.99**	**0.99**
501	Orthrus	0	1	20.7k	97	0.00	**0.06**
	Velox	0	1	20.7k	97	0.00	**0.02**
	Flash*	5	884	19.9k	92	0.01	**0.25**
	GRAAL	16	13	20.8k	81	**0.55**	**1.00**
051	Orthrus	0	3	16.1k	151	0.00	**0.25**
	Velox	0	0	16.1k	151	0.00	**0.12**
	Flash*	4	140	16.1k	147	0.03	**0.25**
	GRAAL	34	11	16.2k	117	**0.76**	**1.00**

Table 2.2: Local-level detection comparison on three hosts of the DARPA OpTC Dataset (*threshold is biased, leading to optimistic performances). Table from [3].

2.3 Parse Tree Representation with Gaur

In his Ph.D., Grégor Quetel tackles an important issue of application-specific sensors: they are not standardized and cannot be easily adapted to other applications. We propose to instrument a software component most applications share: the parser. Indeed, many communications protocols are text-based (such as HTTP, FTP, JSON, or SQL) and they must be parsed by the application to be transformed into machine-usable objects. While parsers can be written manually, they are often created with parser generators such as yacc/bison, ANTLR, Menhir, etc. By modifying a parser generator, we can ensure that generated parsers are instrumented automatically without requiring any modification to the software project. This instrumentation is used to produce a parse tree of every parsed string that can be used for detection and that is, by design, application-independent.

Semantic Model	**Semantic Tags**
Expert	*actions*: Create, Delete, Modify, Execute, Read *objects*: Tablespace, Table, Index, View, User, Procedure, Database, Function, Instance, Logfile, Server, Trigger
Mistral	Data definition, Data import export, Data manipulation, Data query, Database management, Locking concurrency, Miscellaneous operations, Replication clustering, Resource management, Security privileges, Statement control, Stored procedures functions, System information, System maintenance, System variables, Temporary objects, Transaction control, Triggers events, User management, Views

Table 2.3: Examples of semantic model instantiations abstracting the semantics of MySQL user-inputs obtained during step 1. Table from [4].

The originality of this work is that we capture three levels of information that can be extracted from parse trees [32]. *Lexical information* captures surface properties of tokens derived by the lexer (e.g., values, casing, symbol usage). *Syntactic information* describes their organization according to the grammar. *Semantic information* characterizes the effect of an input on the application's behavior. Yet, unlike lexical or syntactic information, semantic information is substantially harder to define, collect, and efficiently provide to decision engines. For this reason, we compare the propositions of an expert and of five Large Language Models (LLM) to propose semantic categories associated with each rule of the grammar.

To verify this idea empirically, our proposition has been implemented with Gaur, a modification of the bison parser generator, that has been used to instrument an SQL database. The SQL grammar has been analyzed: the outputs of the expert and of one LLM are included in Table 2.3. The expert identified 5 different actions on 12 different objects, while Mistral identified 20 categories. From these categories, Gaur produces a parse tree augmented with semantic tags, as shown in Fig. 2.5. We coupled our feature extraction with an autoencoder and evaluated this method on the Superviz25-SQL dataset. We obtained high performances, as shown in Table 2.4. In particular, these results confirm that using LLM to add semantic tags is competitive with human experts, and that this process could be automated for any software project using a parser. Because the LLM is only used once to parametrize the instrumentation, the overhead

during detection is minimal (less than 1ms). This work has been published at IFIP SEC in 2026 [4]. Ongoing work suggests that this method is largely better than the state of the art for transfer learning as well, thanks to the automatic extraction of transferable semantic features.

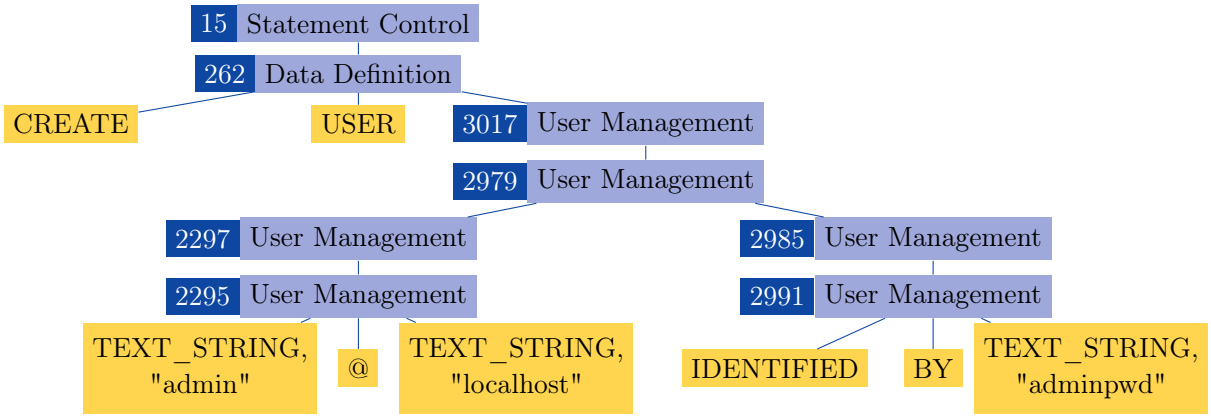


Figure 2.5: Simplified tree structure produced by Mistral for the query `CREATE USER "admin"@"localhost" IDENTIFIED BY "adminpwd"`. Colors indicate the type of information captured: lexical, syntactic and semantic. Figure from [4].

Metric	Expert	ChatGPT	Claude	Llama	Mistral	OSS-GPT
AUPRC	92.76±0.03	**95.57±0.02**	92.91±0.03	94.04±0.03	93.66±0.03	94.13±0.03
AUROC	96.50±0.02	**98.06±0.01**	97.34±0.02	97.89±0.02	97.74±0.02	97.61±0.02

Table 2.4: AUPRC and AUROC (%) for SQL attack detection with 95% confidence intervals of Gaur using different semantic models. Table from [4].

2.4 Wireless Communication Representation with RIDS

Finally, I also worked on wireless communication representation during my postdoc at LAAS-CNRS. In this work, published at the ACM TCPS journal in 2021 [56], we focus on wireless communication intrusion detection. In contrast to wired communications, where only physically connected devices can communicate, wireless communications are intrinsically more vulnerable to attacks. Besides, wired communications can be easily monitored with a network tap (i.e., a physical device that receives all bits transiting through a wire), wireless communications are much more difficult to monitor due to the large available spectrum. To monitor this spectrum, we used a Software-Defined Radio (SDR) that can locally measure the radio wave (more precisely its "IQ", in-phase and quadrature components). Because the sampling rate is finite, there is a natural trade-off between temporal resolution (the time between two measures at the same frequency) and spectral resolution (the spectral width between two measurements). A classical, dedicated device receiver can focus on a few frequencies, allowing for a high temporal resolution. However, to monitor a wide range of frequencies, we need to lower the temporal resolution.

The issue is that the famous Nyquist-Shannon sampling theorem states that it is typically required to have a high enough temporal resolution to capture the data. The corollary is that to

monitor widely the spectrum, one cannot access the bits of the communication. Therefore, this method operates on the physical layer of the OSI model. Even if we cannot access higher levels of abstraction, this is not necessarily an issue for wireless communication, as many wireless protocols used in the Internet of Things are proprietary and undocumented anyway. More precisely, our method is protocol-agnostic.

A typical representation of these measurements is the "waterfall", which plots the signal intensity across frequencies against time. A preprocessed waterfall is presented in Fig. 2.6.

The IDS we propose relies on an autoencoder whose encoders contains convolutional layers that are adapted at analyzing images. One originality of this work is that we use a 1D convolution layer because we allow time-translation invariance but not frequency-translation invariance. In Fig. 2.6, the center picture shows the reconstructed spectrogram and the right picture shows the difference between the original and the reconstructed error, revealing an attack.

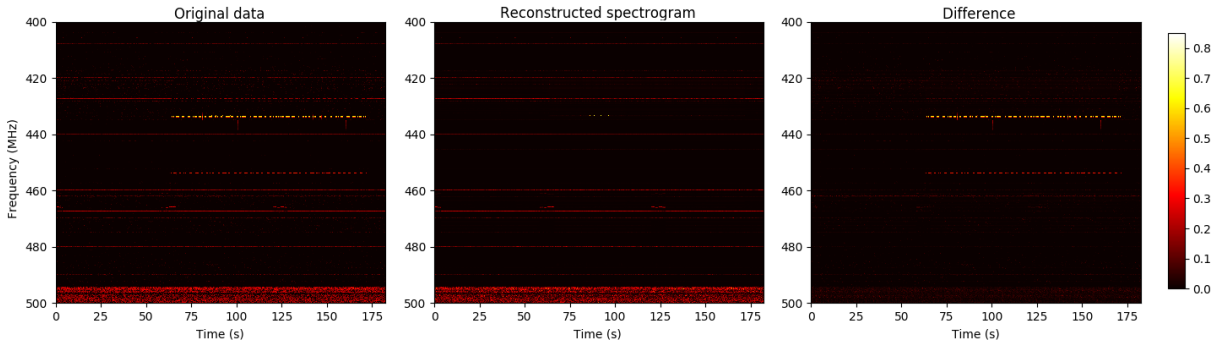


Figure 2.6: Original, preprocessed spectrogram on the left, the reconstructed spectogram (i.e., the output of the autoencoder) in the middle, and their difference on the right. Figure from [56].

To evaluate this work, we had to create our own dataset. We set up a testbed in the lab, at the place regularly used for coffee breaks, and near workstations. Part of the legitimate behavior stems from users[7] and from automated actions. Three probes were deployed, and all of them monitored three bands typically used in IT and IoT. The collection lasted for 20 days. Our experiments showed that we could accurately detect attacks in the 400–500 MHz band and the 800–900 MHz band, but it was more difficult to detect in the 2.4–2.5 GHz band that is more crowded with communications.

Name	Probe 1	Probe 2	Probe 3
400–500 MHz			
scan433-17	52.59%	97.61%	0%
DoS433-27	100%	100%	4.00%
DoS433-40	100%	100%	100%
TV-spoofing	100%	100%	100%
bruijn	92.31%	100%	100%

Name	Probe 1	Probe 2	Probe 3
800–900 MHz			
scan868	65.82%	76.00%	65.09%
DoS868	83.75%	95.00%	65%
2.4–2.5 GHz			
ID 13–18	$\leq 1\%$	$\leq 1\%$	$\leq 1\%$

Table 2.5: Proportion of detected anomalies. Table from [56].

During the evaluation of the methods, we had some unexpected anomalies. For example, there was a 15-minute anomaly detected by the model. After analyzing the signal and inter-

[7]Because private communications were also captured, this dataset is not public.

viewing the users of the lab, we identified the signal as a wireless phone call. This event is rare nowadays, and did not occur during the 9-day training period. We consider this anomaly to be a false positive: it is a rare but legitimate event.

We also found several unexpected anomalies related to possible malicious behaviors. We chose to leave these anomalies in the datasets as they are true anomalies and should be identified as such by anomaly-based IDS, rather than just being rare events. First, some probes failed. During this failure, they did not halt but only returned noise with no signal on two bands (software-defined radios were not very mature in 2019, when this experiment was performed). Second, an attack performed on the 433 MHz frequency produced a strong 866 MHz harmonic that was visible in the second band. And finally, we detected a strong signal on two close frequencies, 462 MHz and 467 MHz, for 33 hours that did not match any known device nor technology.

To find the origin of the 33-hour anomaly, we had to go all over the laboratory with a radio receiver. Since it was intermittent, it was no easy task. Our initial assumptions included the nearby radio laboratory (what if their Faraday cage leaked?) and some actual malicious behavior (IoT devices can easily be part of a botnet without the user noticing, and covert radio communication is not unheard of[8]).

2.5 Other Contributions to Data Representation

In the Ph.D. of Vincent Raulin, we proposed a new representation of sequences of system calls [57, 49, 39] based on an ontology similar to but more complex than provenance graphs. During my post-doc, I also published a few articles on intrusion detection for embedded systems, including avionics systems [59] and smart industrial devices [58]. I also recently contributed to an alert correlation mechanism deployed in the hospital *Hospices Civils de Lyon* [12].

2.6 Conclusion

Some articles on intrusion detection thrive on newer and more complex deep learning models, even if it means more opaque and less scalable methods. In this chapter, I explored a second, complementary path: how to represent data in a way that is both relevant according to cybersecurity expertise and adapted to machine learning's constraints. This way, we can propose more explainable and more scalable methods.

Explainability, in particular, is in my opinion a necessary property for unsupervised IDS, as shown in the hunt for the mysterious radio signal. This event profoundly affected my perception of anomaly detectors, their operational utility and their evaluation. The following chapter tackles this question.

So, what about the 33-hours anomaly? It was a monitor brought by an intern: it leaked whenever it was powered on. Due to the flex office policy for interns, it was used intermittently. Not an attack, in the end.

[8] https://en.wikipedia.org/wiki/The_Thing_(listening_device)

Chapter 3

Anomaly Explainability and Dataset Quality

3.1 Introduction

Explainable AI, or XAI for short, is a set of techniques to produce an explanation for the prediction of a machine learning model. Similar to the intersection between AI and cybersecurity, there are several fields of research in XAI, namely: 1) the security of XAI, 2) XAI to enhance attacks, and 3) XAI to enhance defense against attacks. The first field is the most studied, and includes questions such as "how to protect against manipulations targeting the fairness, confidentiality or the integrity of explanations?". My research focuses on the third field: using XAI to increase transparency and explain alerts. The interested reader can learn more about how XAI and cybersecurity interact in a survey [41] I co-authored in 2022.

However, the vast majority of XAI techniques applicable to cybersecurity focus on explaining supervised machine learning models (i.e., learning from both benign and malicious observations). A few methods [52] have been proposed to adapt SHAP to autoencoders but they have some drawbacks.

3.2 Temporal, Frequential and Spatial Diagnostic for Radio IDS

In the previous chapter, I mentioned the detection techniques used in RIDS [56] for analyzing the radio communications. A second contribution of this work is the temporal, frequential and spatial diagnostics that are associated with each alert. The temporal diagnostic outputs start and end times for each anomaly using two thresholds. From the time period, the reconstruction errors of the various probes are extracted. The frequential diagnostic outputs the main frequency of the attack by identifying the frequency with the largest reconstruction error. Finally, the original signal strength is extracted from the measurements identified by the temporal and frequential diagnostics. The spatial location is estimated with a weighted centroid algorithm. I will not further detail the temporal and spatial diagnostics for brevity's sake.

The frequential diagnostic is very effective in our experiment, as shown in Table 3.1[1]. But even if it helped us identify the source of the 33-hour anomaly, it still has clear limitations. First, it assumes that there is only one frequency involved in the attacks, or at least a few close frequencies, which may not be suited to certain multi-frequency attacks. Second, it can sometimes fail. Let us analyze the large error observed for Probe 3 on the DoS443-27 attack. In the 400-500 MHz band, there is some constant signal of DVB-T[2]. This signal can be considered

[1] This work was published before the SHAP-AE method [52].

[2] More specifically, the "Télévision Numérique Terrestre" (TNT) in France

random at the physical layer, making it difficult for the autoencoder to compress and decompress it effectively. When the signal strength of the attack is low enough, the frequential diagnostic component may select the DVB-T frequency instead of the attack frequency. This is exactly what happened in this case. This limitation is lifted by the next contribution.

Name	Frequency	Probe 1	Probe 2	Probe 3
400–500 MHz				
scan433-17	433 MHz	0.1 MHz	0.1 MHz	×
DoS433-27	433 MHz	0.1 MHz	0.1 MHz	63.4 MHz
DoS433-40	433 MHz	0.1 MHz	0 MHz	0 MHz
TV-spoofing	485–499 MHz	0 MHz	0 MHz	0 MHz
bruijn	433.8 MHz	0 MHz	0 MHz	0 MHz
anomaly462	462 MHz	×	0.1 MHz	×
anomaly467	467 MHz	0 MHz	0 MHz	×
800–900 MHz				
scan868	868 MHz	0.1 MHz	0.1 MHz	0.1 MHz
DoS868	868 MHz	0.1 MHz	0.1 MHz	0.2 MHz
harmo433	867.7 MHz	0 MHz	0 MHz	0 MHz

Table 3.1: Median frequency error. Table from [56].

3.3 Feature Importance for Reconstruction-Based Anomaly Detection

A few years later, during the Ph.D. of Maxime Lanvin, we encountered an issue with our anomaly detector while evaluating it on the CICIDS2017 dataset. This dataset contains 5 days of traffic: Monday does not contain any attack and is reserved for learning, while Tuesday to Friday contain various kinds of attacks. While the false positive rate of most days of the test set is between 0.2% and 0.3%, it is 22% on Wednesday. This is surprising because the false positive rate only depends on the benign traffic and the distribution of the benign traffic in this dataset (which was captured on a testbed without real users) is stationary and should remain the same across days. While benign traffic can be affected by attacks (e.g., if an attack makes a service unavailable), a hundredfold increase is worth investigating. The following work has been published at RAID in 2023 [38] and is reproduced in the appendix on page 72.

The detector relies on an anomaly-based network intrusion detection system previously published by the team. This method, proposed by [61] and reimplemented by Maxime [47], transforms network packet capture files into a heterogeneous "security objects graph" (see the example in Fig. 3.1). Each edge of this graph is then analyzed with an autoencoder. More precisely, a vector is the concatenation of several features: the edge type, the source node type and its associated features, and the destination node type and its associated features (for example, the DNS query type of a DNS node, or the duration of a network connection node). Besides, numerical features are discretized with a Gaussian mixture model, and all features are encoded in a one-hot manner, so each dimension of this vector is associated with one value of a feature. In the following, I will use the term "dimension" to designate each binary element of the vector (for example, the dimension "proto=TCP") and the term "feature" to designate the features of

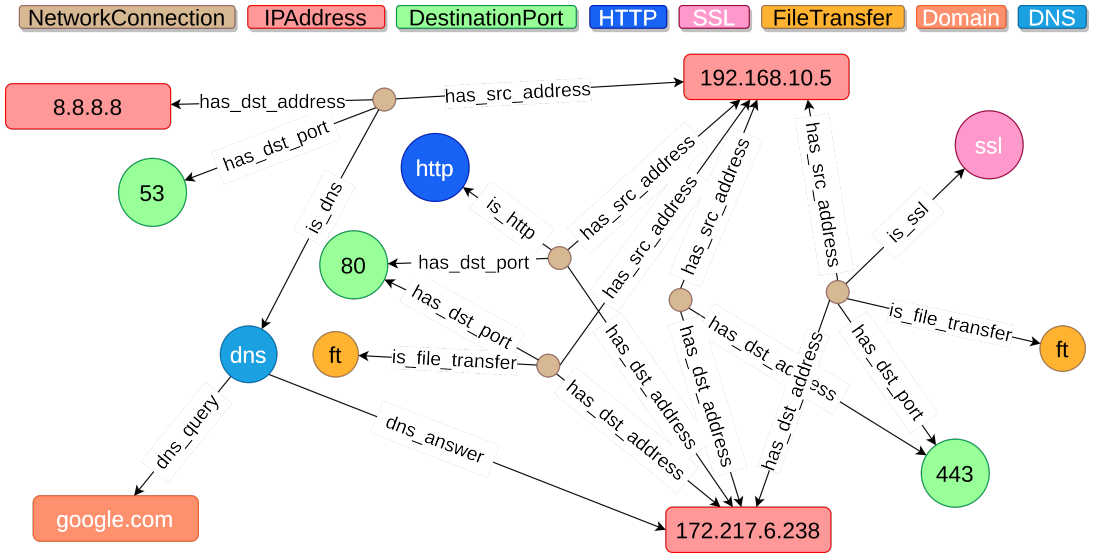


Figure 3.1: Example of a security objects graph. Figure from [38].

the node (for example, the feature "proto").

Our research question is: *for a given anomaly, how to order the importance of each dimension?*. The approach proposed in the previous section, i.e., ordering the dimension by their reconstruction error, yields poor results. Indeed, similar to the DVB-T signal that is more difficult to reconstruct, the dimensions of this vector exhibit widely different reconstruction errors, ranging from quasi-null (for dimensions that are rarely if ever set to 1) to much higher (for dimensions with higher variance and with minimal mutual information with other dimensions).

As an example, check Fig. 3.2 that plots the reconstruction error on benign traffic for two dimensions, A and B. As shown by the box plots, the reconstruction error is typically smaller for dimension A than it is for dimension B. This means that some reconstruction errors, such as 0.1, are less remarkable for dimension B than they are for dimension A. We capture this idea with a measure of the p-value: for any dimension, its importance score is the probability of measuring a reconstruction error smaller than what is actually observed.

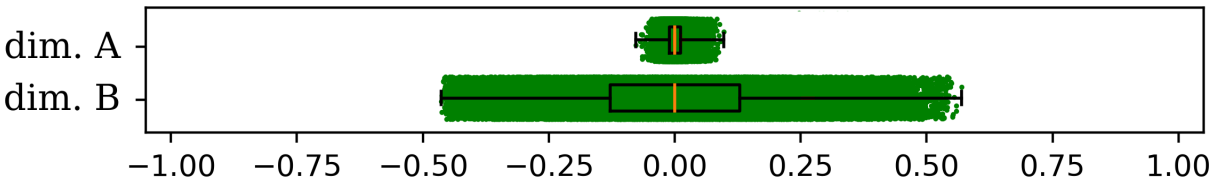


Figure 3.2: Example of empirical reconstruction error distributions.

For example, assume we measure a reconstruction error of 0.05 for dimension A and 0.2 for dimension B. The empirical reconstruction error distribution of dimension A indicates that the probability of measuring an error less than 0.05 is 95%, while the probability of measuring an error less than 0.2 for dimension B is 75%. In that case, even if the absolute value of the reconstruction error of dimension B is the highest, the most unusual (and therefore important) reconstruction error is the one of dimension A, which fits the intuition. This approach, which we

call "AE-pvalues", is independent of the model and can be applied to any reconstruction-based method.

A difficulty of this approach is estimating the p-value. Even though the empirical distributions tend to be unimodal, we prefer not to assume a Gaussian distribution. So, instead of estimating the parameters of a family of distributions, we rely solely on the empirical distribution[3]. We estimate the p-value with $p_i = \frac{\#\{r_i \geq e_i\}}{\#\{r_i\}}$, where i is a dimension of the autoencoder, r_i are the empirical reconstruction errors observed in the train set, and e_i is the observed error for the vector we need an explanation for. So, to estimate the p-value of e_i, we empirically verify the proportion of empirical errors from the training set that are higher.

However, by its nature, the training data contain few outliers, so it may be possible that the observed error e_i is higher than any errors observed in the training set. When it happens for several dimensions, their estimated p-values are zeroes and we cannot rank them. So, when two p-values are equal, we use a secondary criterion to rank the associated dimension: $\alpha_i = \frac{e_i - m_i}{p_i^{99} - p_i^1}$. This criterion relies on the distance between the measured error e_i and the median of the empirical reconstruction errors m_i, normalized by the difference between the 1st and 99th percentiles of the empirical reconstruction errors. This method seeks to capture how far from the distribution the value e_i is, while taking into account that not all distributions have the same support length.

To evaluate this approach, we purposefully included anomalies in vectors and verifies whether the explanations match these anomalies. We compare AE-pvalues to the method proposed in RIDS, called "AE-abs", and with the state-of-the-art SHAP_AE and a random baseline. The mean rank of the perturbed features or dimension is lower for AE-pvalues than it is for SHAP_AE. Besides, AE-pvalues is four orders of magnitude cheaper to compute (1.9ms per prediction) than the state of the art SHAP_AE (28s per prediction). We also validate the predictions on the CICIDS2017 datasets by investigating true and false positives.

Explanation method	Mean rank of the **perturbed dimension** ↓	Mean rank of the **network feature** ↓
AE-pvalues	**4.61**	**1.39**
AE-abs	5.78	1.49
SHAP_AE	18.96	2.15
Random	26.93	7.8

Table 3.2: Mean ranks of the perturbation per explanation method. Table from [38].

Fig. 3.3 shows the most common explanations per attack type in the CICIDS2017 dataset. For example, the SSH-patator attack is correctly identified using SSH-related features, and scanning attacks are correctly identified using the IP addresses and destination port values. We also identified some curious explanations: for example, web attacks and the botnet are identified with the user-agent (denoted "ua_os" and "ua_browser" in Fig. 3.3). The user-agent is a string sent with HTTP requests that identifies the OS and the browser of the client. This is certainly shortcut learning, i.e., the model relies on an experimental artifact: the creators of the dataset probably used audit software to launch these attacks. Such software generally indicate clearly in their user-agent that they are legitimate audit tools. Real attackers certainly remove such indicators for their malicious actions.

[3]The empirical distribution is estimated on the validation set, not on the training set.

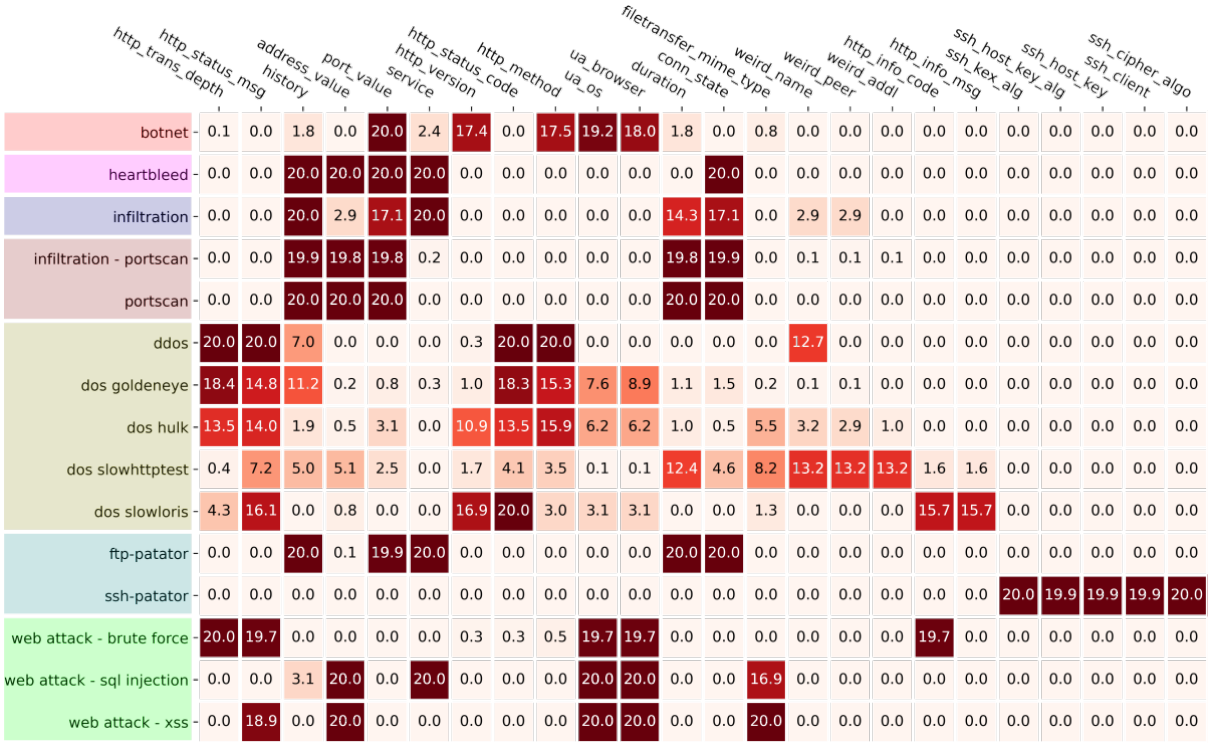


Figure 3.3: Visualization of the explanations per attack type. The scores are the percentages indicating how often the features occur in the top-5 explanations for the respective attack type. Figure from [38].

But it was not the only problem we found in this dataset. Thanks to AE-pvalues, we could investigate the curious 22% false positive rate on Wednesday. This investigation led to a full contribution that I describe in the next section.

3.4 Errors in the CICIDS2017 dataset

Thanks to AE-pvalues, we discovered that a scan attack was launched on Wednesday but was not documented anywhere in the dataset. So, they were not false positives, but in fact true positives. This discovery surprised us: CICIDS2017 is a widely used dataset and we wondered why this issue had not been discovered previously. We therefore continued our analysis, which taught me an important lesson: if the quality of the dataset cannot be trusted, then the evaluation results cannot either. The data quality[4] is therefore the focus of the rest of this chapter. In the end, we did find several issues in the dataset and published our results at the CRiSIS conference in 2022 [48]. Other articles pointing out other problems in this dataset have also been published [53, 55]. Here are the main problems we found.

First issue: the preprocessing tool (called CICFlowMeter) did not behave correctly when the packets were not properly temporally sorted, which was the case in CICIDS2017.

[4]Data *characterization* is a better term than data *quality* because quality is always relative to a certain use case. In this dissertation, I will use the term "quality" in terms of "adapted for IDS evaluation".

Second issue: some timestamps were incoherent, i.e., some packets were temporally inverted (for example, a SYN-ACK packet was received before the SYN packet was sent). Such an inversion can impact the preprocessing step: the source and destination IP of a conversation are typically derived from its initial packet. Inverting the two initial packets can lead to inverting the source and destination IP addresses, which can have a strong impact on the intrusion detection system. This issue was probably due to the sensor, but we could not find its exact source.

Third issue: a significant issue was data duplication. Normally, network sensors are located at routers. Routers are not typically the destination of packets but serve as interfaces between subnetworks. It means that a packet typically enters a router from one subnetwork and exits to either the same subnetwork or another. Depending on where the port mirroring is located, the sensor can capture different packets. An hypothesis consistent with the observations is that the sensor is configured to capture all packets incoming from the company subnetwork *and* outgoing to the company subnetwork, meaning that packets whose origin and destination are in the company subnetwork would be observed twice. These duplicated observations can later impact statistics (for example, the number of packets observed in a conversation).

Fourth issue: as mentioned previously, one attack is neither documented nor labeled properly. We identified a port scan attack performed by an internal host. We counted about 4,060 port scans for all six Ubuntu machines, about 5,300 for the Windows 7 machine, about 6,900 for the Mac machine, about 8,000 for the two Windows 10 machines and about 12,000 for the Windows 8.1 machine. These large numbers explain why it leads to an inflated false positive rate.

Then, why was this issue not found earlier? We verified the performances of supervised machine learning on this dataset. These models use a different train/test sets split, so they can train with attacks as well. There is another port scan in the training set, so we expect the model to correctly find the port scan attack we discovered. However, the new port scan attack is an *internal* port scan, so it is affected by the third issue (data duplication). So, even if the actual attack resembles the known port scan from the train set very much, two bugs are canceling each other: the internal port scan has wrong features due to the duplicated data, making it more difficult to identify, and it was labeled as benign, hiding the issue with the lack of detection. So, we found this issue thanks to both our unsupervised approach and our new explainability method.

We published a corrected version of the dataset alongside a new version of the labels.

3.5 Identify Experimental Biases in Provenance Graph-based Detectors

A trustworthy dataset is not a sufficient condition for trustworthy results: it must also be properly used. In the article we published during Fanny Dijoud's Ph.D. [3], we searched for bias in the literature [31] of provenance graph-based detectors [26, 24, 29, 15, 9]. Starting from 50 biases collected from the literature [40, 35, 73, 6, 9], our methodology presented in Fig. 3.4 found 12 in other detectors, which we consolidated into 7 biases split in two categories: biases related to the training and detection methodology (denoted "A") and biases related the reproducibility of the results (denoted "B"). All detectors from the literature had at least two biases, as indicated in Table 3.3. When possible, we removed biases in the detectors and published their fixed versions for fairer comparisons.

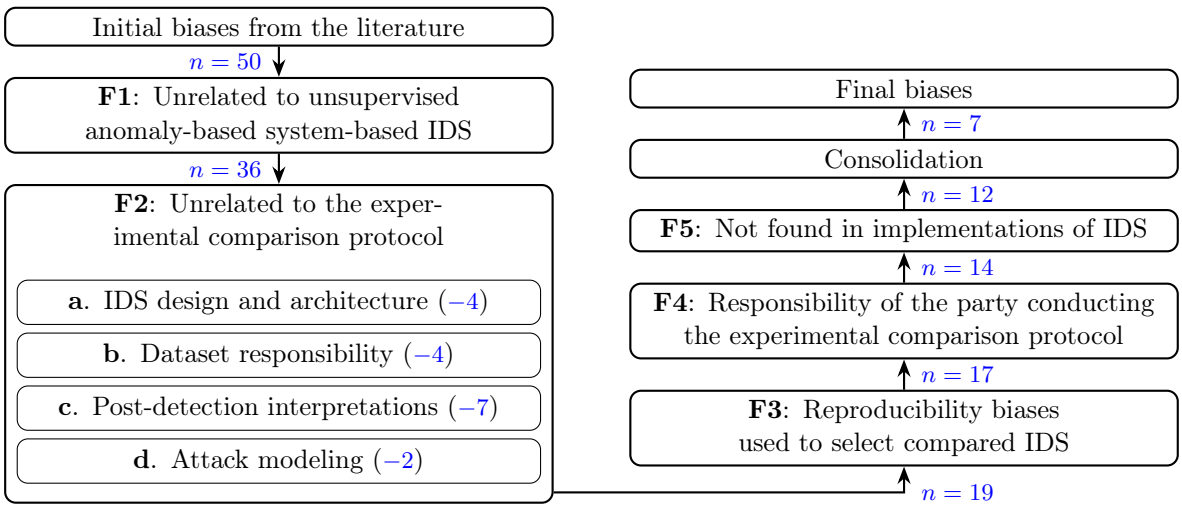


Figure 3.4: Methodological review of the biases. In blue, the number of biases after each step. Figure from [3].

Bias	Description	Affected IDS
A1	**The division of the dataset** into training, validation, and test sets must be **clearly defined and well documented**. In the case of temporal data, the split must preserve chronological order: timestamps cannot be shuffled across sets, i.e., each set should correspond to consecutive time periods.	Magic, Kairos
A2	**Test data** must not be used during the **training phase**.	Magic
A3	A model must be evaluated on the **entire test set**, without leveraging any prior knowledge of the attacks it contains.	Kairos, Flash, Orthrus, Velox
A4	**Ground truth** must not be used during the **detection phase**, but only to analyze the accuracy of the detection results.	Flash
A5	All **parameters** of a solution must be clearly defined and documented. The method used to determine their values must be explicitly stated and **must never rely on knowledge of the test data**. Among the parameters, threshold value requires special attention as it significantly impacts the detection performance. Therefore, the process for setting its value must be fair, rigorous and justifiable.	Magic, Kairos, Flash
B6	Each dataset used, along with its associated ground truth (i.e., dataset labeling), must be **available**.	Kairos
B7	All relevant **artifacts** must be shared to enable others to reproduce the detection results.	Kairos, Flash, Orthrus, Velox

Table 3.3: Description of the 7 consolidated biases and the affected detectors. Table from [3].

3.6 CasinoLimit Exercice and Dataset

A common limitation with datasets that are created in testbeds is that the attacks are not very realistic: they are automatically run with audit tools (which are less stealthy than actual attackers' tools) and they follow an idealized scenario of an attacker who knows exactly what vulnerability can be exploited and who does not commit any errors. While this assumption could be reasonably made for high-profile, state-sponsored, attackers who seek to steal industrial secrets or manipulate public opinion, the vast majority of attacks target opportunities and stealth is secondary to immediate, financial gain.

To better understand this second class of attackers, the PIRAT team proposed in 2024, 2025 and 2026 a cybersecurity challenge to the BreizhCTF event, which is the largest in-person cybersecurity competition in France. The challenge we propose consists of finding and executing a series of attacks to compromise a host, move inside a network, and finally obtain the "flag" (a short message) that proves the challenge has been succeeded. I contributed to the 2024 and 2025 challenges and led the design team for the 2026 challenge. As an example, the steps required to complete the 2026 challenge are presented in Fig. 3.5. In this challenge called *SKY_FALL[M]*, the participants must infiltrate a company network in order to get access to a satellite controlled by an evil LLM. Access to the LLM was simulated by a Minitel[5], where the participants had to convince the LLM to stop its malicious behavior. Of the 120 teams present at the BreizhCTF this year, 44 started the challenge and 8 completed it entirely.

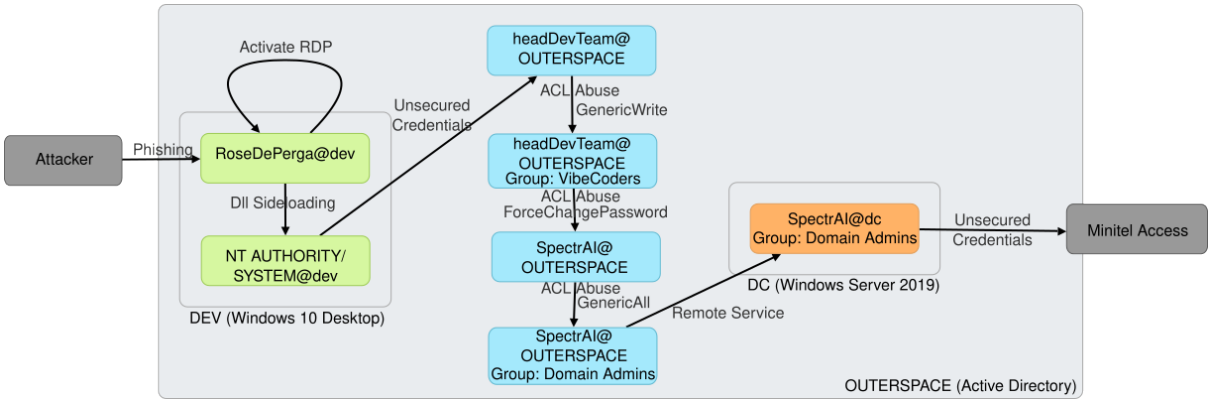


Figure 3.5: Steps required to complete the challenge for BreizhCTF 2026.

There are several objectives in creating such an exercise: besides contributing to the PIRAT's and Inria's visibility, it allows software developed in the team to be tested outside the lab, and we collect the data[6] for research purposes. This led to the creation of a dataset by Ph.D. student Sébastian Kilian (who I do not supervise), published at RAID in 2025 [16]. This dataset includes system logs and network traffic and is labeled with MITRE ATT&CK techniques.

[5] A nationally famous terminal for a discontinued French technology from the 80's.
[6] This process is vetted by Inria's legal team.

3.7 Other Contributions to Data Quality

For brevity's sake, I'll mention a few more works. I contributed to an article published in TIFS in 2022 [45] on the representativity of datasets and proposed a new algorithm to reduce class inconsistencies across datasets. The debiased datasets have been publicly released [54].

During the Ph.D. of Grégor Quetel, we proposed two new datasets. Superviz25-SQL [23], published at ANUBIS in 2025, is specifically dedicated to evaluating unsupervised SQL attack detectors. The two existing datasets have many issues that we sought to avoid by relying on the literature on dataset bias. To ensure the quality of this dataset, we open-source the experimental testbeds, characterize the diversity of the datasets using state-of-the-art metrics (see Table 3.4), and propose a reference experimental protocol with baseline results for easy comparisons.

Dataset	Sample Type	Number of Samples	Vocabulary Size	Unique Parse Trees	Semantic Space Dispersion
Superviz25-SQL	Normal	3,352,696	**459,215**	**58,891**	**0.01197**
WAF-A-MoLE		345,199	332,827	15,131	0.00623
Kaggle Dataset		19,537	14,471	515	0.01156
Superviz25-SQL	Attack	336,281	**119,721**	5,133	**0.01255**
WAF-A-MoLE		393,345	12,669	**7,336**	0.00991
Kaggle Dataset		11,382	10,805	242	0.01099

Table 3.4: Diversity Metrics Across Superviz25-SQL, WAF-A-MoLE, and Kaggle Dataset datasets. Table from [23].

Another dataset, Superviz26-SQL, was published at ANUBIS in 2026 [5]. It focuses on the evaluation of detectors across several domains: a database about flights and airports, a database of a video rental shop, a database of a bicyle manufacturing company, and a database of an HR department. The originality of this work is that this dataset contains several variants: Superviz26-SQL-LODO for evaluating the transferability of models between domains, Superviz26-SQL-CD for evaluating models in a context of concept drift, and Superviz26-SQL-FSL for evaluating the impact of few-shot learning in transfer learning. We evaluate the diversity of the dataset and propose reference experimental protocols with baseline results.

3.8 Conclusion

Creating datasets is challenging. Cybersecurity datasets can last several weeks, meaning they must run without issues for as long. But, as shown previously, many issues can occur: computers can crash, probes can be improperly configured or fail, data may be lost, and so on. Each iteration of the dataset therefore requires several weeks, or even months, of experiments, which can quickly add up to a long time that is incompatible with a Ph.D. timeline, especially these data are required for other experiments. And when datasets contain errors, one can only wonder: "are my conclusions even correct?"

Therefore, in an attempt to produce high-quality datasets with a much shorter iteration duration, I focused my work on a third research question: how to generate synthetic cybersecurity datasets with artificial intelligence?

Chapter 4

Synthetic Security Data Generation

4.1 Introduction

Empirical sciences can only move forward because ideas and theories can be empirically tested, compared and refuted [82]. When datasets and evaluation methodologies are biased, the validity of scientific contributions becomes questionable.

Currently, several approaches for creating datasets exist. The first one is measuring actual traffic on a real system with real users. This approach can lead to very realistic datasets, but is rarely used for several reasons: it is not replicable by design, due to the inclusion of human participants, and sharing such data can lead to privacy and confidentiality issues, and often requiring anonymization or sharing only metadata, which often degrade detection capabilities.

The second approach is relying on testbeds, i.e., virtual machines without actual users. They generally include scripts [17, 36, 68] to emulate user behavior, such as Internet browsing, email sending, etc. This approach is the most widely used [13] due to its limited confidentiality and privacy risks, and because it can (in theory) be reproducible, even if dataset creators rarely release the configuration and software required to actually reproduce it. Realism is more limited due to the lack of actual users, and it also tends to be less diverse due to the complexity of setting up a wide range of scripted user behaviors. Finally, as mentioned in the conclusion of the last chapter, creating a dataset with these methods is slow, which limits the number of iterations that can be done.

A third approach has been proposed recently. For a few years, many articles have been published on the use of artificial intelligence to generate synthetic network traffic without any emulation or simulation. This chapter contains my contribution to this area of research.

In my research project, I focus mainly on the generation of *benign* network traffic for a few reasons. First, an important obstacle to using machine-learning-based intrusion detection in actual environments is high false positive rates [20]. To better evaluate this metric, it is of utmost importance to have a wide variety of benign behaviors which is very difficult to obtain from real users due to privacy and confidentiality reasons. This is less true for attacks due to the wider availability of automated attacking tools. Second, I believe one can more easily stochastically create variants of user behaviors than of attacks. Attacks exploit vulnerabilities by satisfying strict constraints, making them more difficult to tweak. For example, slightly modifying the timing of benign behavior may not have a large impact, while some attacks may completely fail due to such a change. However, I recognize that the generation of attacks is important and complementary to my own work, and that the end goal of creating datasets requires both benign and malicious traffic to be useful.

The next section will introduce a short background on network communication that is central to the contributions.

4.2 Networking fundamentals

As the reader may not be a networking expert, I'll focus this summary on the fundamentals required for this chapter. Networking focuses on the communications between computers so they can share information and resources. Because several computers typically share a common physical medium (such as a cable or radio frequencies), communication data are split into several data units with a fixed format defined by network protocols. Due to the many different requirements for smooth communication, there are several protocols with different tasks and scopes. For this introduction, we will focus on the Internet model that is composed of four layers: the link layer (that handles the communications within the same local network), the internet layer (that handles the communication between distant networks), the transport layer (that provides a channel for communication and can provide reliability mechanisms) and the application layer (that provides high-level service, like browsing or printing).

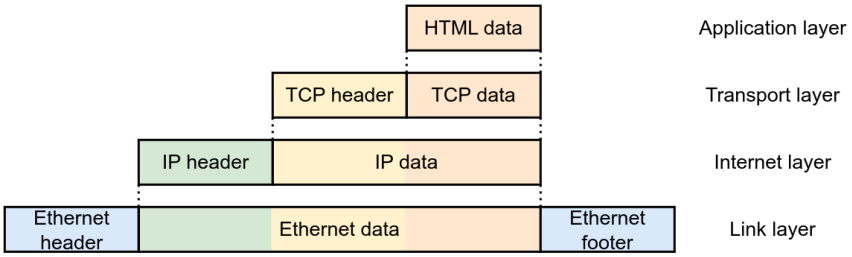


Figure 4.1: Examples of encapsulated protocols in the Internet model

For each layer, the data unit is split in two: a header with metadata and the payload (with the exception of the link layer that can also have a footer). The layers are encapsulated, meaning that the data unit of the application layer is wrapped into the data unit of the transport layer, which is itself wrapped in the data unit of the internet layer, etc., as shown in Fig. 4.1. Depending on the layer, the protocol data unit is called frame, packet, datagram, segment, etc. In the following, I will abusively use the term "packet" to denote any protocol data unit.

In the following, I will assume that the link layer relies on the Ethernet protocol and the Internet layer on the IP protocol due to their ubiquity in IT networks. More generally, my work primarily focuses on the transport and application layers. Indeed, in the context of IDS evaluation, the higher levels are generally more closely studied than the lower levels that carry less information about the content of the data unit and more about the state of the network. This limitation will be further discussed in Chapter 5. Similarly, I will assume there is no IP fragmentation nor TCP segmentation, and no VLAN tags.

The transport layer protocol can typically be TCP or UDP. TCP is meant for high-reliability connections and has mechanisms to ensure dependability. UDP, on the other hand, is used for low-latency applications or for protocols where the TCP overhead would be too high.

There are many different application layers, and sometimes the protocol changes within the same conversation (such as with TLS tunneling). We can distinguish between text-based protocols and binary-based protocols depending on whether they are human-readable or not. Some protocols rely on cryptography. In fact, about 95% of Web communications are encrypted[1].

[1] https://transparencyreport.google.com/https/

Due to the high throughput of network communications, IDS typically rely on a higher-level abstraction rather than analyzing each packet. We call a *flow* a set of packets exchanged between two IP addresses for certain ports (i.e., processes) on a particular transport protocol. For example, the set of TCP packets between source IP 192.168.0.3 with source port 7690 and destination IP 128.93.162.123 with destination port 80 form a flow[2]. Each flow contains the packets pertaining to one logical communication, such as reaching a website. From such flows, one can extract flow statistics. These statistics include the total number of packets, the duration of the communication, the mean ingress throughput, etc. There are many formats for flow statistics: the original NetFlow format from Cisco, the open IPFIX format, or the CICFlowMeter format widely used in cybersecurity research.

Besides, each computer is associated to an address, called the IP address[3], that I will assume is unique and permanent. While this assumption is incorrect in general, at least due to private IPs and the finite duration of DHCP lease, it is generally verified in datasets. This address is used by routers to route packets between subnetworks to their intended recipient.

4.3 State of the Art

This section contains a short summary of my survey on the subject [21] published at ANUBIS in 2025. This survey was conducted in 2025 and its corpus contains 68 articles. Most articles focus on generating flow statistics (60%) or individual packet metadata (36%), but a few also generate packet payloads. As expected, generative AI is the most commonly used set of techniques, including Generative Adversarial Networks (GAN) and variants, present in 69% of the articles, and Variational Autoencoder (VAE) and variants, present in 12% of the articles. More recent generation techniques, notably diffusion-based models and LLMs, are also used in several articles. Finally, a few non-deep learning techniques, such as SMOTE, its variant ADASYN and Bayesian networks have also been used.

However, a recurrent issue is the evaluation of synthetic data. There is no straightforward metric to evaluate synthetic data, like accuracy can be for supervised learning. Therefore, articles use many different metrics, such as TSTR ("Train on Synthetic, Test on Real"), marginal distribution distances (Jensen-Shannon Divergence (JSD), Earth Mover Distance (EMD), etc.), pairwise correlation (Pearson correlation, Spearman correlation, etc.), similarity distance between data points (cosine similarity, Euclidean distance, total variation distance, etc.), networking-specific metrics and qualitative, visual evaluation.

This lack of common metrics and baselines, makes these works difficult to evaluate and compare. However, several articles [43, 34, 27] show that very simple methods such as SMOTE are competitive, or even outperform, more complex methods such as GAN and VAE. This observation suggests that more can be done to enhance the quality of the synthetic data.

4.4 Flow Statistics Generation

As a first step, we worked on flow statistics generation. In the Ph.D. of Adrien Schoen, after unfruitful experiments with deep learning methods, I proposed experimenting with Bayesian

[2] In practice, other constraints such as a maximum duration of inactivity are used.
[3] In my work, I focus on IPv4 only.

networks that were central in my own Ph.D. [69, 70]. A Bayesian network is a probabilistic graphical model in which each node represents a random variable[4] and each arrow denotes a conditional dependence. An example is shown in Fig. 4.2.

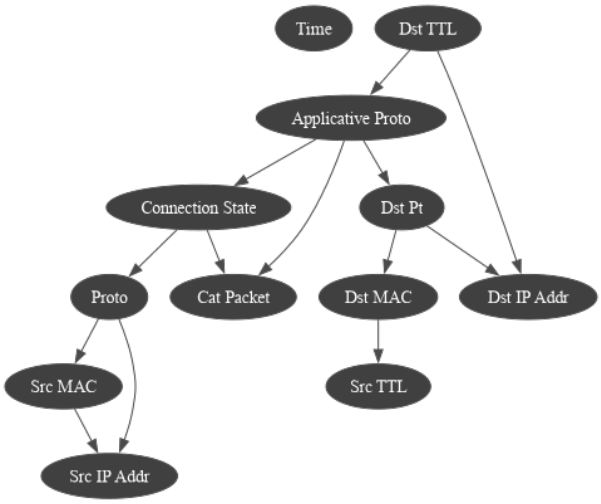


Figure 4.2: Example of a Bayesian network (the conditional probability tables are omitted) used for network flow generation.

Bayesian networks have many convenient properties: they can represent any probability distribution, their graph structure makes them interpretable and there is a simple, linear algorithm for generating new samples. However, many associated problems are NP-hard, most notably their learning, making them unsuited for high-dimension problems. Besides, they tend to struggle with mixing continuous and discrete features, and with deterministic relations between features (which can violate the assumption of faithfulness).

This article, published at RESSI in 2022 and extended for the WTMC workshop in 2024 [50, 33], has two main contributions: a generation method and an evaluation framework.

For the generation method, we use Bayesian networks to generate flow statistics containing either 7, 11 or 30 features depending on the dataset. The process can be summarized as follows: at training time, the data is first preprocessed and then a Bayesian network is learned, and at inference time, the Bayesian network draws new vectors that are then postprocessed. Most of our contributions are in the preprocessing step (the postprocessing step approximates its inverse), which prepares features for the Bayesian network. It includes numerical features (such as flow duration, number of packets, etc.) as well as discrete features with a high cardinality, i.e., a large set of possible values (such as port number and IP addresses). For the numerical features, we experiment with quantile quantification and with a variational Gaussian mixture model. For the port features than can take any of 65536 values, we decided to keep only the 30 most common values and to assign a unique value for all ephemeral ports[5] For IP addresses, we keep the private IPs and merge all public IPs into a single value. This is motivated by the fact

[4]For consistency with the rest of the dissertation, I will use the term "feature" instead.

[5]An ephemeral port is randomly selected at the start of the community and have a value greater than 1024 (the exact range depends on the OS), while the vast majority of non-ephemeral ports have a value lesser than 1024.

that public IPs are generally anonymized for privacy reasons and, therefore, could be generated randomly.

The evaluation framework identifies four important criteria: realism (the synthetic sample could probably happen in a real environment), diversity (how much of the variety of the training dataset is reproduced), novelty (the synthetic samples should not simply reproduce training data) and compliance (the generated data should comply with networking rules). It could be argued that compliance is a subcriterion of realism, but we understand realism in a statistical context and compliance in a standard-oriented context. For example, it is possible to have realistic but non-compliant communications (not all implementations actually respect their standard [46]), and we could have compliant and non-realistic communications (for example, the third packet of a TCP handshake can contain a payload but this feature is rarely used by implementations). This tension between realism and compliance will be further discussed in Section 4.6.

There is also a tension between novelty and realism. The generation algorithm called random oversampling simply resample observed vectors to produce new ones[6]. While this strategy does not bring any novelty, it can only produce realist vectors. On the other hand, generating new vectors means taking the risk to generate unrealistic ones.

For each criterion, we propose several metrics, as presented in Table 4.2. The experiments show that the Bayesian networks produce higher-quality data than state-of-the-art generation methods. We also show that a naive generation method (that simply draws each feature from the observed data independently) has better results than several deep-learning-based models. The detailed results are indicated in Table 4.1.

	Description	**Real data**	**Naive**	**BN_{bins}**	**BN_{GM}**	**CTGAN**	**WGAN**	**NetShare** [51]
JSD	Realism and Diversity for categorical features (↓)	**0.017**	**0.017**	**0.025**	0.031	0.148	0.090	*0.23*
EMD	Realism and Diversity for numerical features (↓)	**0.002**	**0.002**	**0.014**	*0.080*	0.016	0.055	0.062
CMD	Realism of correlation between categorical features (↓)	**0.013**	0.160	**0.018**	**0.018**	0.126	**0.071**	*0.257*
PCD	Realism of correlation between numerical features (↓)	**0.761**	1.186	**0.630**	**0.891**	0.949	1.152	*1.191*
Density	Realism of data distribution (↑)	**1.000**	*0.079*	**0.906**	**0.887**	0.867	0.862	0.391
Coverage	Diversity of data distribution (↑)	**0.967**	*0.161*	**0.952**	**0.955**	0.903	0.655	0.214
MD	Novelty (=)	**7.175**	5.766	**6.929**	**6.987**	6.862	6.864	*5.247*
DKC	Compliance (↓)	**0.003**	*0.088*	0.004	**0.003**	0.008	**0.002**	0.023
Global Rank	Average Ranking (↓)	**-**	4.375	**1.75**	**2.375**	3.625	3.375	*5.5*

Table 4.1: For each metric : *Red* indicates the worst model, **Orange** the second-best model and **Green** the best model. (↑): Higher is better, (↓): Lower is better, (=): Closest to the real data is better. The last line gives the average rank given by all metrics to each model. Table from [33].

[6]This method is typically used to rebalance classes on the dataset and not to produce new samples.

	Criterion				Input			Data type	
	Real.	*Div.*	*Nov.*	*Comp.*	*Marg.*	*Cond.*	*Joint*	*Cat.*	*Num.*
JSD	✓	✓			✓			✓	
EMD	✓	✓			✓				✓
CMD	✓					✓		✓	
PCD	✓					✓			✓
Density	✓						✓	✓	✓
Coverage		✓					✓	✓	✓
MD			✓				✓	✓	✓
DKC				✓			✓	✓	✓

Table 4.2: Metrics proposed in our evaluation framework. Real.: Realism, Div.: Diversity, Nov.: Novelty, Marg. Distr.: Marginal Distribution, Joint Distr.: Joint Distribution. Table from [33].

4.5 Temporal Flow Statistics Generation with FlowChronicle

A limitation of the previous work is that each flow is drawn independently. However, it is well known that there are some temporal correlations between network flows. For example, DNS requests typically precede other flows, and we could expect some common patterns, such as opening an email that loads its images (an IMAP flow followed by an HTTPS flow), or the concurrent flows in peer-to-peer protocols such as Bitcoin. We expanded the previous work in the context of the SecGen associate team between Inria and CISPA. This work, in collaboration with Ph.D. student Joscha Cüppers from CISPA, led to a publication at CoNEXT in 2024 [25].

This work, called FlowChronicle, proposes a pattern language to mine[7] patterns from observations. To mine the "best" set of patterns, we use a score called the Minimum Description Length [76] (MDL) that, as its name indicates, measures the length (in bits) of the shortest possible description of the training data using a set of patterns. It is based on the idea of Occam's razor, i.e., that out of all possible descriptions, the shortest one is the best one. Formally, it is based on the Kolmogorov complexity [78]. For a given model class $\mathcal{M}$, MDL identifies the best model $M \in \mathcal{M}$ as the one that minimizes the number of bits needed to describe both model and data given the model, $L(D, M) = L(M) + L(D|M)$ where $L(M)$ is the length of model M in bits and $L(D|M)$ the length of data D given M in bits. In the context of FlowChronicle, a model is a set of patterns. The definition of the pattern language significantly impacts the equations used in the MDL score, so it is important to design a pattern language that encompasses the expected "true" model without being overly complex.

A novelty of this work is the proposition of a hybrid pattern language with a deterministic part (called the "partial flows") and a stochastic part (a Bayesian network). The partial flows are a sequence of flow statistics whose values can be either of the following categories: a Fixed value that is defined ("hard-coded") in the partial flow, a Free variable whose probability distribution is defined in the Bayesian network, or a Reused variable that is always equal to some Free variable. A simplified example[8] is presented in Fig 4.3.

[7] Here, "mine" can be broadly understood as "extract". Data mining seeks to extract knowledge from data, for example in the form of patterns, while machine learning seeks to learn models that can later be used to perform a task, such as classification. Even though the delimitation can sometimes be hazy (here, patterns are mined to perform a task, namely, data generation), I'll use the data mining vocabulary in this section.

[8] Simplified in the sense that partial flows have actually 11 columns and that the conditional probability tables

Partial flows

Flow	Source IP	Dest. IP	Dest. Port
#1	β_A	8.8.8.8	53
#2	A	β	80

Bayesian Network

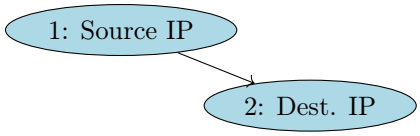


Figure 4.3: Example of a simplified pattern

In this example, the pattern consists of two partial flows denoted #1 and #2. The source IP address of the flow #1 is random and its marginal distribution is described in the conditional probability table of the "Source IP" node of the Bayesian network. The destination is always 8.8.8.8 (a DNS server owned by Google) on destination port 53 (used for DNS). The source IP address of the flow #2 reused the variable A that is defined to be equal to the source IP address of the flow #1. So the two source IP addresses of the flows will always be the same. The destination IP address is random and its conditional probability is defined in the conditional probability table of the "Dest. IP" node of the Bayesian network. This is a *conditional* distribution, meaning that it depends on the source IP address. Finally, the destination port is always 80 (used for HTTP).

The definition of this language is motivated by several observations. First, we know from practice that patterns must have randomness, especially in the source IP addresses, because many different computers may use the same service (be it a Website, a printer, etc.), so deterministic patterns are ill-suited. Second, Bayesian networks succinctly represent probability distributions, making them well-suited to the MDL score, and they can be somewhat easy to interpret.

In terms of expressiveness, "Reused variables" do not add anything: one could replace them with a Free variable whose conditional probability table actually enforces the equality. However, by attributing its own category, we can describe this kind of relation with fewer bits than with a whole conditional probability table. Besides, it makes patterns easier to analyze: in this example, we know that both source IP addresses are identical without checking the conditional probability tables.

To compute the MDL score, we also need to encode the training data given a set of patterns. To do that, we will *cover* the data with the patterns, as shown in Fig. 4.4. In this example, there are three patterns (on the left), denoted ε, p and q. The training data (on the right) consist of 8 flows. Instead of describing explicitly the data, which would require many bits, we use the patterns. For example, we identify that pattern q occurs for flows at time 56, 113 and 145. So, to describe these flows, we can simply encode that the pattern q is used, starting from time 52, and only encodes the free variables. To ensure we can also cover all the training data, notice how the pattern ε is structured: it contains a single partial flow with only free variables. Therefore, it can cover any flow. It is difficult (NP-hard to be precise) to find the optimal cover, so we use a greedy algorithm.

The MDL score instantiatied for this pattern language is:

of the Bayesian network are not displayed.

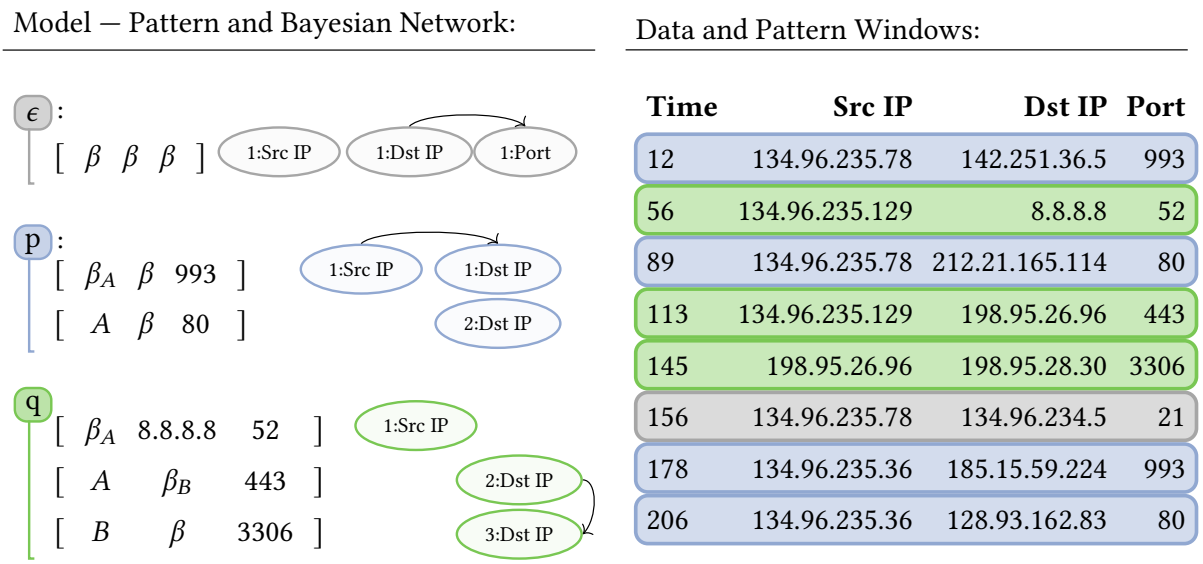


Figure 4.4: Examples of patterns (on the left) and the pattern occurrences used to cover the dataset. β denotes a free variable. Figure from [25].

$$L(D \mid M) = \sum_{p \in M} (L_{\mathbb{N}}(|W_p|) + L(W_p))$$

where W_p denotes the set of occurrences of pattern p used to describe D. The length of W_p is the timestamps plus the free cells, encoded based on the probabilities given by the Bayesian network,

$$L(W_p) = \sum_{i=1}^{|W_p|} \left(L(t_1 \text{ of } w_i) + \sum_{k=2}^{|p|} L(t_k \text{of } w_i \mid t_{i-1}) \right) - \log(Pr(w_i | BN_p, \{w_j | j < i\}))$$

where $L(t) = \log(t_{\max} - t_{\min})$ and $t_{\max}$ (resp., $t_{\min}$) refers to the maximum (resp., lowest) timestamp in the data D, $L(t_i \mid t_j) = L_{\mathbb{N}}(t_i - t_j)$.

The experimental results show that FlowChronicle outperforms the state-of-the-art, as shown in Table 4.3. Other experiments specifically focused on evaluating temporal series also confirm this tendency.

A limitation of this approach is the scalability of MDL-based pattern mining: we could not reasonably mine from the whole dataset, so we mine patterns from several time windows. This limitation is due to the NP-hard complexity of identifying the set of patterns minimizing the MDL score.

In the context of the Ph.D. of Antoine Cellier, we are working on a similar problem: identifying temporal sequences not for any flows, but specifically for Web services. Indeed, Web technology accounts for a large share of network traffic and simply connecting to a website can initiate multiple flows to retrieve media (images, videos, etc.), CSS scripts, fonts, Google analytics, ads, etc. Preliminary results are promising and we plan to submit them at a conference before the end of 2026.

	Density	CMD	PCD	EMD	JSD	Coverage	DKC	MD	Rank
	Real. ↑	*Real.* ↓	*Real.* ↓	*Real./Div.* ↓	*Real./Div.* ↓	*Div.* ↑	*Comp.* ↓	*Nov.* =	*Average Ranking*
Reference	(0.69)	(0.06)	(1.38)	(0.00)	(0.15)	(0.59)	(0.00)	(6.71)	-
IndependentBN [33]	7 (0.24)	5 (0.22)	6 (2.74)	8 (0.11)	4 (0.27)	4 (0.38)	4 (0.05)	4 (5.47)	5.25
SequenceBN	6 (0.30)	2 (0.13)	5 (2.18)	7 (0.08)	3 (0.21)	3 (0.44)	2 (0.02)	3 (5.51)	3.875
TVAE [64]	3 (0.49)	4 (0.18)	3 (1.84)	2 (0.01)	5 (0.30)	5 (0.33)	6 (0.07)	5 (5.17)	4.125
CTGAN [64]	2 (0.56)	3 (0.15)	2 (1.60)	3 (0.01)	2 (0.15)	2 (0.51)	8 (0.11)	2 (5.70)	3.0
E-WGAN-GP [63]	8 (0.02)	7 (0.34)	8 (3.63)	5 (0.02)	7 (0.38)	8 (0.02)	7 (0.07)	6 (4.66)	7.0
NetShare [51]	5 (0.32)	6 (0.28)	**1 (1.47)**	6 (0.03)	6 (0.36)	6 (0.22)	5 (0.05)	7 (3.82)	5.25
Transformer [62]	**1 (0.62)**	8 (0.78)	7 (3.62)	**1 (0.00)**	8 (0.55)	7 (0.03)	3 (0.05)	8 (3.75)	5.375
FlowChronicle	4 (0.41)	**1 (0.03)**	4 (2.06)	4 (0.02)	**1 (0.10)**	**1 (0.59)**	**1 (0.02)**	**1 (5.87)**	**2.125**

Table 4.3: Ranking of our different models without considering the preservation of temporal dependencies. For each metric, the average of the score is given between parentheses. Real.: Realism, Div.: Diversity, Comp.: Compliance, Nov.: Novelty, ↓: Lower is better, ↑: Higher is better. =: closer to Reference is better. Table from [25].

4.6 Packet Generation with TADAM

Generating flow statistics can be sufficient to evaluate some intrusion detection systems, but others may require access to full packets. Generating a sequence of packets consistent with given flow statistics is not easy, so we propose an intermediate step: generating, for each packet, metadata such as packet direction, inter-packet arrival time and TCP flags (a part of the TCP header). A similar idea has also been independently proposed by [28].

Intuitively, such sequences depend mainly on the flow's application protocol. For example, a DNS communication is almost always composed of two packets: a single outgoing packet with a request and a single incoming packet with an answer. Protocols relying on TCP will contain some TCP-specific sequences (such as the initial handshake, as well as some form of connection end), and one could expect that the number, sizes and directions of the packets are different between Web streaming (mostly server-to-client), uploading files to a FTP server (mostly client-to-server), etc.

Formal language theory proposes several language classes to describe such sequences. Fortunately, the majority of networking protocols are described with finite-state automata[9] because they make for simplified implementations with bounded memory usage per connection. Therefore, we limit our search to finite-state automata, or, more specifically, to one of their variants.

Fortunately, finite-state automata learning has a long tradition, with the first major work on the subject being the L* algorithm from Angluin [80]. However, the majority of the literature focuses on active learning, where the learner tries to identify a target automaton by submitting queries to a teacher (such as "is this string accepted by the target automaton?", called a membership query, and "is there a string that is accepted by either the learned or the target automaton but not both?", called an equivalence query). Active learning has led to many interesting theoretical results [2] but it cannot be applied in our context, where we assume that we only have access to observations and not to a live system we can query.

Several algorithms for passively learning regular sets[10] do exist but they generally require both positive (i.e., present in the set) and negative (i.e., absent from the set) examples. However,

[9] This is however an approximation. Nested communications, for example, would need Dyck's languages to be described properly and these languages are famously not regular.

[10] Regular sets are equivalent to finite state automata.

observations only give us access to positive examples.

The necessity of negative examples is for theoretical guarantees: without them, there is no way of verifying if the learned model accepts too many strings or not. Luckily, two algorithms have been proposed to learn regular sets from positive examples only, namely TAG [42] and RTI+ [75]. However, by design, every example they encounter will be included in their model.

This last property is sadly not compatible with real-world, noisy observations. For example, a probe could invert adjacent packets (as it occurred in the CICIDS2017 dataset), there could be retransmission due to lost packets, unseen packets due to alternate routing, and a wide variety of similar phenomena. Such noise could easily clutter the learned model, making it much more complex than the true automata. Therefore, I worked with Lénaïg Cornanguer, a postdoctoral researcher from CISPA and first author of TAG [42], on a new algorithm to learn regular automata from positive, noisy examples only. This work has been published at SDM in 2025 [10] and is reproduced in the appendix on page 88.

Like FlowChronicle, this work relies on the MDL score to select the best model, in this case a timed probabilistic automaton. A timed probabilistic automaton is an automaton whose edges are labeled with a transition probability and a probability distribution that represents inter-arrival times in our application. To compute the MDL score, we must notably define $L(D|M)$ by accepting any observation with any automaton. If that observation is part of the automaton, one can easily estimate its description length through its probability (as indicated by Shannon's source coding theorem). But, if that observation is not part of the automaton, then the classical method is to assume an uninformative prior, i.e., that each symbol has the same probability of appearing, which can be very costly.

However, this approach is not adapted to noisy observations. For example, if one typically only observes strings[11] in the form $(ab)*$, i.e., *ab*, *abab*, *ababab*, etc., then observing *aabb* should intuitively have a lower cost than observing *aaaa* because symbol inversion can happen. Our proposition is therefore to compute, for each string, its distance to the timed probabilistic automaton. This algorithm, inspired by the classical dynamic programming algorithm for computing the Damerau–Levenshtein distance between two strings [79], computes the projection of any string to the automaton, i.e., the closest string accepted by the automaton when using string operations.

More precisely, since we also take into account timestamps, we do not manipulate words but timed words. A timed word $w = (a_0, t_0)(a_1, t_1)... \in (\Sigma \times \mathbb{N})^*$ is a finite sequence of symbols and non-decreasing timestamps. In the context of probabilistic real-time automata, we can re-write a timed word as a sequence of symbols and delays $w = (a_0, d_0)(a_1, d_1)...$ where $d_0 = 0$ and $d_i = t_i - t_{i-1}$.

We consider four different noise types, illustrated with an initial timed word $w = (a_0, d_0)(a_1, d_1)(a_2, d_2)$: deletion ($w = (a_0, d_0)(a_2, d_2)$), insertion ($w = (a_0, d_0)(a_3, d_3)(a_1, d_1)(a_2, d_2)$), transposition ($w = (a_0, d_0)(a_2, d_2)(a_1, d_1)$), and symbol repetition ($w = (a_0, d_0)(a_1, d_1)(a_1, d_3)(a_2, d_2)$) (note that we allow the repeated symbol to be associated with a different delay).

Due to the presence of noise, many words in the training data might not be accepted by the automaton even if it is equivalent to the true generating process of the training data. For this reason, we do not use the classical data encoding that relies on the word probability [76], because all the noisy words rejected by the automaton would have a null probability and would

[11] The exact term in formal language theory is *word*.

have to be encoded explicitly. Instead, we integrate into the encoding the notion that such words could be accepted by the automaton with some light changes. To do so, we associate the timed words with an *edit operation sequence* that describes how to read and correct them such that they are accepted by the automaton. We propose several edit operations that mirror the noise types presented earlier: add (add a pair (a, d) to the word), skip (ignore a pair (a, d)), transpose (permute two consecutive pairs), deduplicate (remove a pair (a, d) following a pair with an identical symbol), and follow (read the pair (a, d) as it is). Note that the "follow" operation allows us to handle the already accepted words in the same way as the words needing to be corrected.

Once the MDL score is defined, we then propose a learning algorithm. To this end, we propose a local search algorithm that iteratively improves a solution by computing new, close candidates. The candidates are computed from the current solution by applying several transformations. For brevity's sake, I will not detail the transformations we propose.

As shown in Figure 4.5, experiments suggest that, compared to the state of the art, TADAM is more robust to noise while learning smaller models.

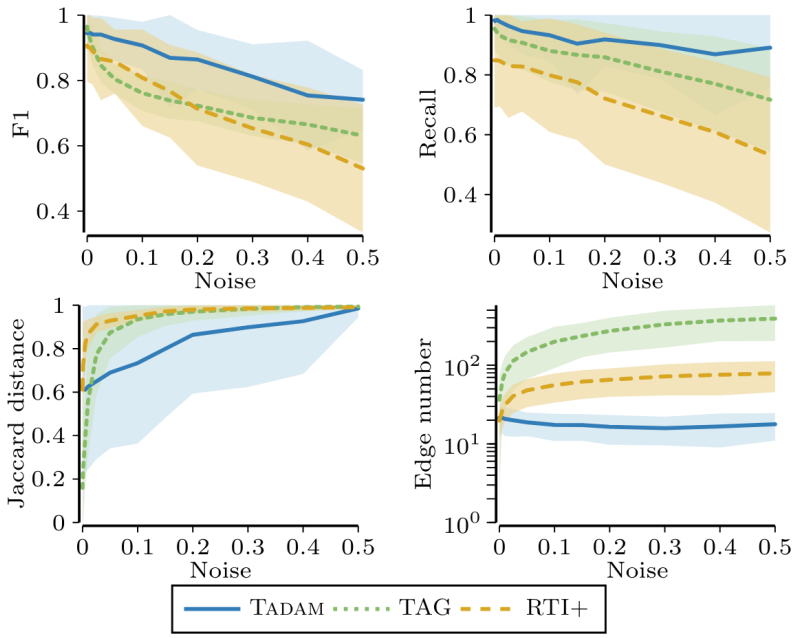


Figure 4.5: Full results of the synthetic data experiment showing the impact of the noise proportion in the training data on TADAM and its competitors. Figure from [10].

We experimented with TADAM on various tasks, including anomaly detection, whose results are summarized in Table 4.4. Although we do not achieve the performance levels of state-of-the-art techniques for log-based anomaly detection, in particular deep learning techniques such as CNN or LogRobust for which F1-score over 0.98 have been reported [44], results are promising.

"But why go through such a complex method when RFCs (i.e., documentation) already exist? There is no need to mine how a TCP handshake looks like" is a common comment on this work. I agree to such comments to a certain extent. Some protocols are widely studied and automata already exist for them. However, there is a difference between documentation and implementation. As I mentioned earlier, the TCP handshake allows data to be transferred

Learner	AU-ROC	TPR	FPR	F1
TADAM	**0.982**	0.998	**0.025**	**0.705**
TAG	0.891	**1**	0.142	0.298
RTI+	0.790	**1**	0.292	0.171
HMM	0.608	0.640	0.085	0.288

Table 4.4: Anomaly detection performance on HDFS_v1 dataset. Table from [10].

in the ACK packet, yet it is rarely used except in some Microsoft implementations. Similarly, reading the documentation of the HTTP protocol tells us nothing about all the different uses of the Web such as browsing, gaming, videoconference, streaming, chatting, shopping, etc., all of which significantly affect the shape of network communications. While extracting automata from documentation would be useful for identifying implementation errors and vulnerabilities, in this work we focus to reproduce actual observed behavior. Besides, many proprietary protocols have no official documentation and could not be reproduced this way.

Fig. 4.6 and 4.7 contain two automata (DNS and SMTP) learned from observations with TADAM. The symbols are tuples containing TCP flags (when applicable), packet directions and payload indicators. The payload indicators are automatically extracted with the tool *tshark*.

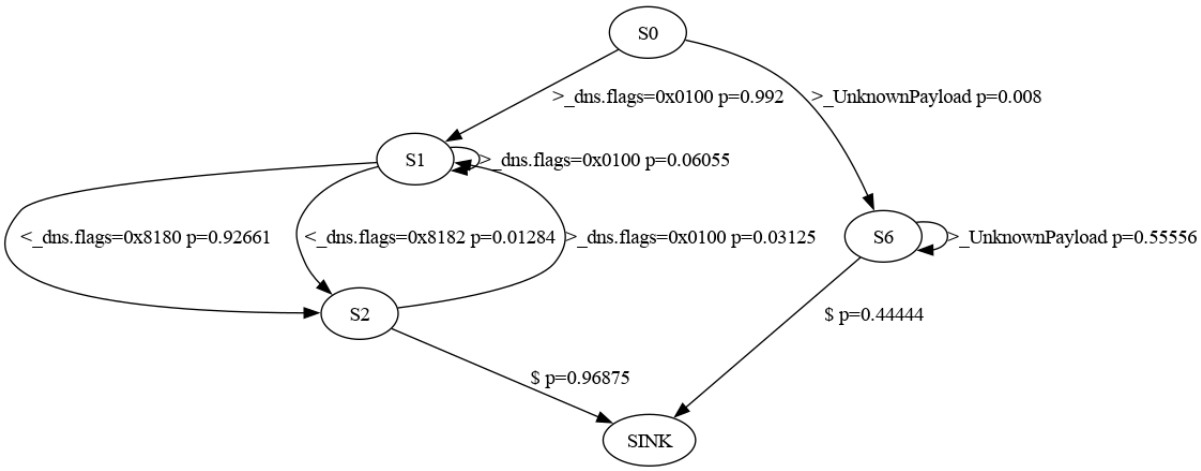


Figure 4.6: Example of a DNS automaton learned with TADAM from the CICIDS2017 dataset. On each edge, the first part is the direction of the packet (> for client-to-server and < for server-to-client), and the second part is an indicator about the payload. Inter-arrival time distributions have been removed for legibility. $ is not a packet but an end-of-string indicator.

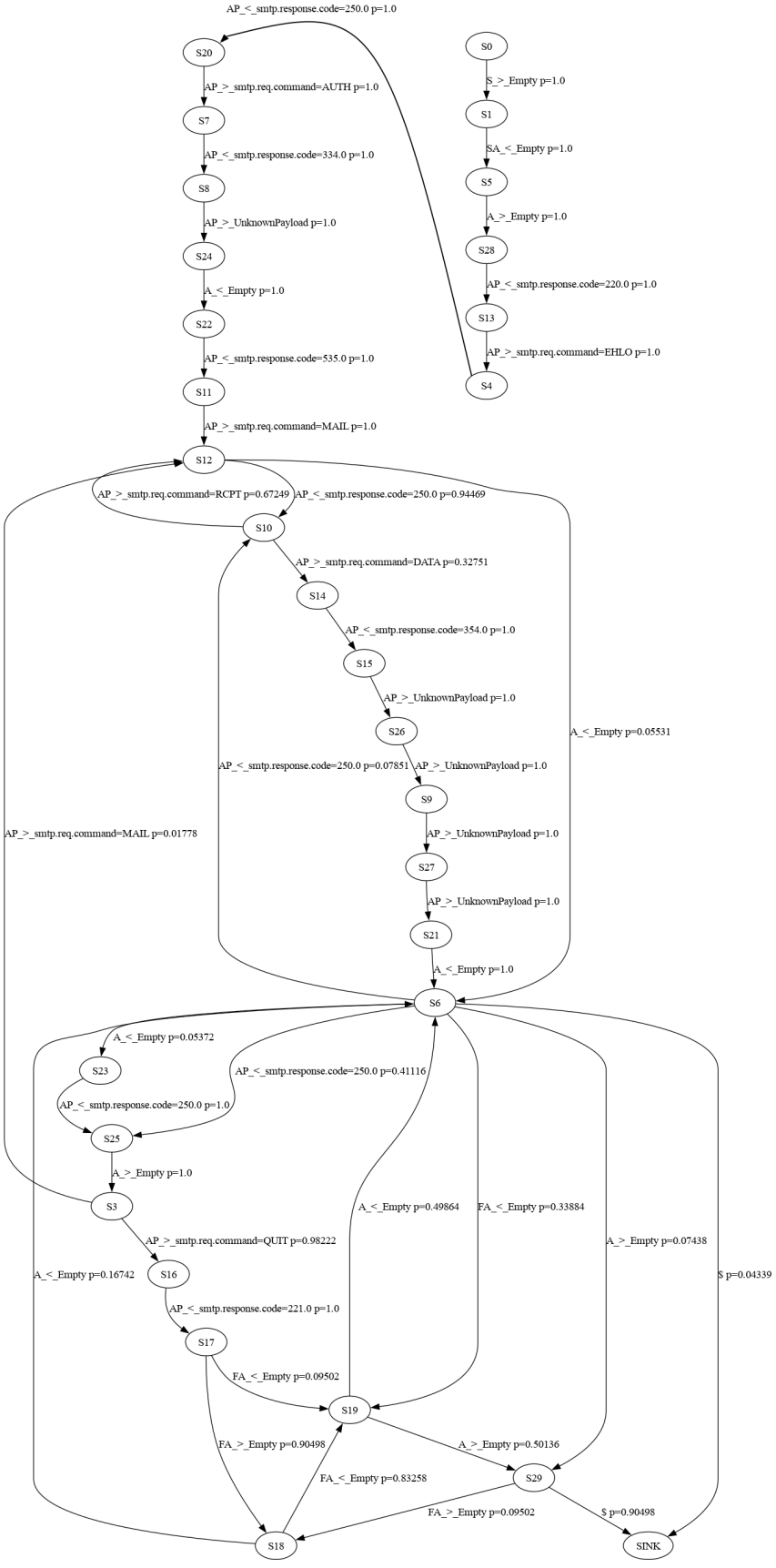


Figure 4.7: Example of a SMTP automaton learned with TADAM from the CUPID dataset.

4.7 Ongoing Work on End-to-end Pcap Generation with Fos-R

The previous works paved the way towards full end-to-end pcap[12] generation. The following work has been submitted but has not been accepted yet for publication. In this article, we present Fos-R[13], a generation pipeline with five stages, detailed in Fig. 4.8.

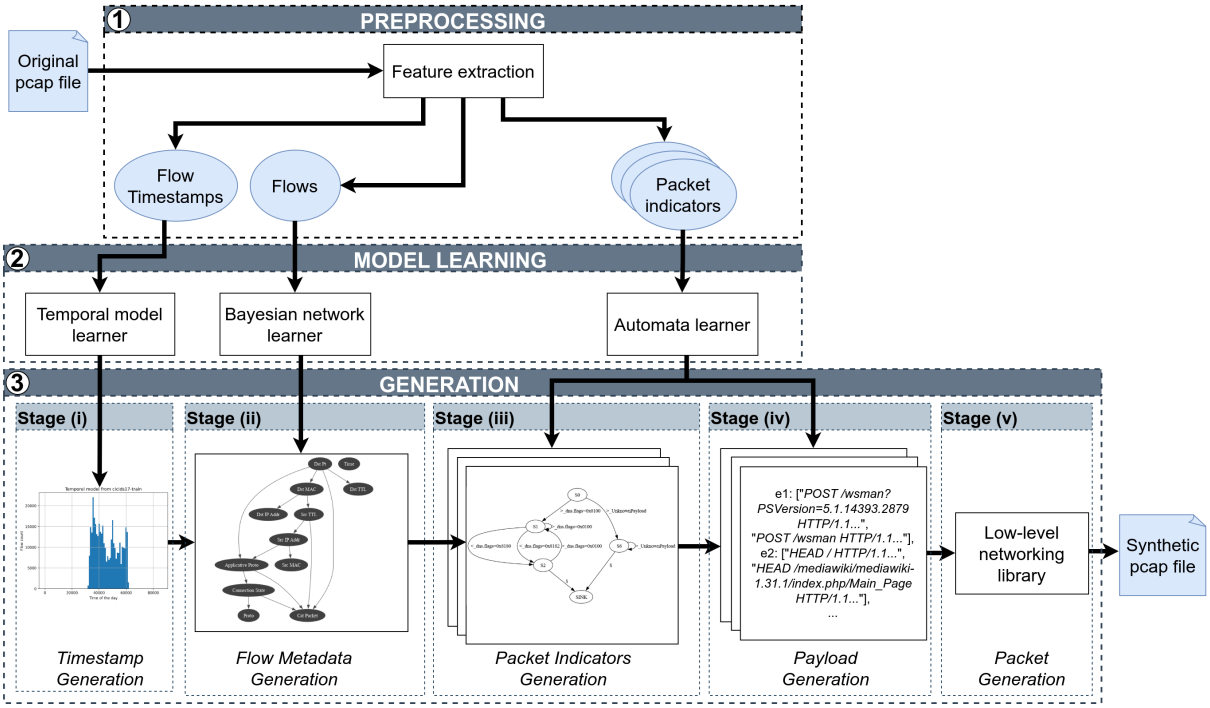


Figure 4.8: Overview of the Fos-R framework

Stage (i): Timestamp Generation. This stage generates timestamps according to a time model. Depending on the implementation, it could be the timestamp of a single flow or the start of a more complex behavior that encompasses several flows. This stage can account for differentiating working hours from nighttime, and weekdays from weekends, alongside other seasonality phenomena. It could also account for time zones, so one can accurately generate traffic for a local time zone even though the training data was captured on another continent. In our example, the generated timestamp is t_0.

Stage (ii): Flow Metadata Generation. This stage generates metadata on a network flow based on the timestamp, including source IP, destination IP, and application protocol. Many related work propose to generate such flow metadata, either independently or with temporal dependencies. In our example, at timestamp t_0, the source IP 192.168.0.1 (i.e., a local IP) contacts the destination IP 8.8.8.8 using the DNS protocol over UDP datagrams.

Stage (iii): Packet Indicators Generation. To avoid generating a complete packet directly, we propose to first propose a sequence of packet indicators for a given flow, as proposed by [28]. A packet indicator is a tuple that can include, for example, the direction of the packet, its inter-arrival time since the last packet in that flow, the payload size, and its TCP flags (if

[12] A pcap is a file containing a raw packet capture.
[13] Pronounced like the French word "faussaire".

relevant). In our example, the DNS communication could consist of two packets: the first tuple is (direction: forward, inter-arrival time: 0ms, payload size: 41 bytes) and the second tuple is (direction: backward, inter-arrival time: 4ms, payload size: 170 bytes).

Stage (iv): Payload Generation. This stage generates the payload for each packet in a way that is consistent with the generated packet indicators. The payload could be generated with LLMs [11], randomly generated [28], replayed from previous captures, or generated from a grammar obtained through RFC or reverse engineering [74]. In this example, the payload of the first packet could be an A query to obtain the IPv4 address of the domain example.com, and the payload of the second packet would be the response containing the corresponding IPv4 address, the class and the time-to-live of the record, etc.

Stage (v): Packet Generation. This stage generates the L2, L3 and L4 headers of each packet according to the previously generated values, such as destination port or IP addresses, and complete the rest (including checksum, protocol version, window size, etc.) according to the specification or standard implementation. This stage may account for congestion, fragmentation and segmentation, and routing. In our example, the two packets are finalized with a checksum, ports, IPv4 addresses, MAC addresses, etc.

We propose an implementation for each stage that relies on the previous works. Another contribution of this work is a method for conditioning automata-based generation of flow metadata. Indeed, it is not straightforward to condition the sampling of an automaton given, for example, the number of forward and backward packets. In fact, there are probably many combinations of numbers of forward and backward packets for which there is no path in the automaton from the initial set to an accepting state that satisfies this constraint.

An example of a generated flow is presented in Fig. 4.9. We also measured the generation throughput on a laptop with no GPU. The results are presented in Table 4.5.

Time	Date	Source	Destination	Dst port	Protocol	Length	Info
59253.780000	1970-01-01 16:27:34,163001Z	192.168.1.35	192.168.1.9	25	TCP	54	49539 → 25 [SYN] Seq=0 Win=65535 Len=0
59253.780700	1970-01-01 16:27:34,163701Z	192.168.1.9	192.168.1.35	49539	TCP	54	25 → 49539 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
59253.780901	1970-01-01 16:27:34,163902Z	192.168.1.35	192.168.1.9	25	TCP	54	49539 → 25 [ACK, CWR] Seq=1 Ack=1 Win=32799 Len=0
59253.781954	1970-01-01 16:27:34,164955Z	192.168.1.9	192.168.1.35	49539	SMTP	145	S: 220 Exchange.ds.lab Microsoft ESMTP MAIL Service ready at Mon, 22
59253.783167	1970-01-01 16:27:34,166168Z	192.168.1.35	192.168.1.9	25	SMTP	67	C: EHLO WS2-PC
59253.784342	1970-01-01 16:27:34,167343Z	192.168.1.9	192.168.1.35	49539	SMTP	293	S: 250-Exchange.ds.lab Hello [192.168.1.35] | SIZE 37748736 | PIPELIN
59253.785137	1970-01-01 16:27:34,168138Z	192.168.1.35	192.168.1.9	25	SMTP	122	C: AUTH ntlm TlRMTVNTUAABAAAAB4IIogAAAAAAAAAAAAAAAAAAAAAGAbEdAAAADw==
59253.785388	1970-01-01 16:27:34,168389Z	192.168.1.9	192.168.1.35	49539	SMTP	288	S: 334 TlRMTVNTUAACAAAABAAEADgAAAAFgomiribNJWjlcScAAAAAAAAAAG4AbgA8AA
59253.785889	1970-01-01 16:27:34,168890Z	192.168.1.35	192.168.1.9	25	SMTP	648	C: TlRMTVNTUAADAAAAGAAYAH4AAAAkASQBlgAAAAAAAABYAAAAGgAaAFgAAAAMAAwAcg
59253.899046	1970-01-01 16:27:34,282047Z	192.168.1.9	192.168.1.35	49539	TCP	54	25 → 49539 [ACK] Seq=565 Ack=676 Win=33247 Len=0
59258.910918	1970-01-01 16:27:39,293919Z	192.168.1.9	192.168.1.35	49539	SMTP	93	S: 535 5.7.3 Authentication unsuccessful
59258.911118	1970-01-01 16:27:39,294119Z	192.168.1.35	192.168.1.9	25	SMTP	81	C: MAIL FROM:<cmontg@ds.lab>
59259.087596	1970-01-01 16:27:39,470597Z	192.168.1.9	192.168.1.35	49539	TCP	54	25 → 49539 [ACK] Seq=604 Ack=703 Win=33437 Len=0
59264.099468	1970-01-01 16:27:44,482469Z	192.168.1.9	192.168.1.35	49539	SMTP	78	S: 250 2.1.5 Recipient OK
59264.099627	1970-01-01 16:27:44,482628Z	192.168.1.35	192.168.1.9	25	SMTP	79	C: RCPT TO:<pkento@ds.lab>
59264.237567	1970-01-01 16:27:44,620568Z	192.168.1.9	192.168.1.35	49539	TCP	54	25 → 49539 [ACK] Seq=628 Ack=728 Win=33626 Len=0
59264.564983	1970-01-01 16:27:44,947984Z	192.168.1.9	192.168.1.35	49539	TCP	54	25 → 49539 [FIN, ACK] Seq=628 Ack=728 Win=33689 Len=0
59264.564985	1970-01-01 16:27:44,947986Z	192.168.1.9	192.168.1.35	49539	TCP	54	[TCP Window Update] 25 → 49539 [ACK] Seq=629 Ack=728 Win=33752 Len=0
59264.909955	1970-01-01 16:27:45,292956Z	192.168.1.9	192.168.1.35	49539	TCP	54	25 → 49539 [FIN, ACK] Seq=629 Ack=728 Win=33815 Len=0
59264.909960	1970-01-01 16:27:45,292961Z	192.168.1.9	192.168.1.35	49539	TCP	54	[TCP Window Update] 25 → 49539 [ACK] Seq=630 Ack=728 Win=33878 Len=0
59265.232551	1970-01-01 16:27:45,615552Z	192.168.1.9	192.168.1.35	49539	TCP	54	25 → 49539 [FIN, ACK] Seq=630 Ack=728 Win=33940 Len=0
59265.232553	1970-01-01 16:27:45,615554Z	192.168.1.9	192.168.1.35	49539	TCP	54	[TCP Window Update] 25 → 49539 [ACK] Seq=631 Ack=728 Win=34002 Len=0
59265.553017	1970-01-01 16:27:45,936018Z	192.168.1.9	192.168.1.35	49539	TCP	54	25 → 49539 [FIN, ACK] Seq=631 Ack=728 Win=34064 Len=0
59265.553019	1970-01-01 16:27:45,936020Z	192.168.1.9	192.168.1.35	49539	TCP	54	[TCP Window Update] 25 → 49539 [ACK] Seq=632 Ack=728 Win=34126 Len=0
59270.564891	1970-01-01 16:27:50,947892Z	192.168.1.9	192.168.1.35	49539	SMTP	78	S: 250 2.1.5 Recipient OK
59270.565090	1970-01-01 16:27:50,948091Z	192.168.1.35	192.168.1.9	25	SMTP	60	C: DATA
59270.565811	1970-01-01 16:27:50,948812Z	192.168.1.9	192.168.1.35	49539	SMTP	100	S: 354 Start mail input; end with <CRLF>.<CRLF>
59270.575402	1970-01-01 16:27:50,958403Z	192.168.1.35	192.168.1.9	25	SMTP	281	C: DATA fragment, 227 bytes
59270.575404	1970-01-01 16:27:50,958405Z	192.168.1.35	192.168.1.9	25	SMTP	555	C: DATA fragment, 501 bytes
59270.575405	1970-01-01 16:27:50,958406Z	192.168.1.35	192.168.1.9	25	SMTP	56	C: DATA fragment, 2 bytes
59270.575407	1970-01-01 16:27:50,958408Z	192.168.1.35	192.168.1.9	25	SMTP/…	59	from: dsunsh@ds.lab, subject: ncPGH7kU2m31NzrGInzocKPbuDsCc8bm, (tex
59270.576325	1970-01-01 16:27:50,959326Z	192.168.1.9	192.168.1.35	49539	TCP	54	25 → 49539 [ACK] Seq=702 Ack=1469 Win=34619 Len=0
59270.676325	1970-01-01 16:27:51,059326Z	192.168.1.9	192.168.1.35	49539	SMTP	199	S: 250 2.6.0 <d18df1a3-ccab-496b-b5b0-32504e6340ca@Exchange.ds.lab> [
59270.876887	1970-01-01 16:27:51,259888Z	192.168.1.35	192.168.1.9	25	TCP	54	49539 → 25 [ACK] Seq=1469 Ack=847 Win=34741 Len=0
59373.489592	1970-01-01 16:29:33,872593Z	192.168.1.35	192.168.1.9	25	SMTP	60	C: QUIT
59373.489792	1970-01-01 16:29:33,872793Z	192.168.1.9	192.168.1.35	49539	SMTP	102	S: 221 2.0.0 Service closing transmission channel
59373.489950	1970-01-01 16:29:33,872951Z	192.168.1.35	192.168.1.9	25	TCP	54	49539 → 25 [FIN, ACK] Seq=1475 Ack=895 Win=34924 Len=0
59373.490250	1970-01-01 16:29:33,873251Z	192.168.1.9	192.168.1.35	49539	TCP	54	25 → 49539 [FIN, ACK] Seq=895 Ack=1476 Win=34985 Len=0
59373.490458	1970-01-01 16:29:33,873459Z	192.168.1.35	192.168.1.9	25	TCP	54	49539 → 25 [ACK] Seq=1476 Ack=896 Win=35045 Len=0

Figure 4.9: A synthetic SMTP flow

Models	CICIDS17	CUPID	DEDALE
Throughput (GiB/s)	5.68	**6.80**	4.50
Throughput (Gbps)	45.44	**54.40**	36.00
MPPS	**18.673**	16.461	16.859

Table 4.5: Generation throughput of Fos-R

4.8 Conclusion

In an area dominated by LLMs, it may seem odd to rely on non-deep-learning methods to generate data. This choice is not based on dogma but on a pragmatic observation: to achieve useful synthetic data generation, we need modular, explainable and low-overhead methods.

A fundamental issue with machine learning is summarized in the saying "garbage in, garbage out": the quality of the model cannot exceed the quality of its training data. However, as shown in Chap. 3, training data may have many issues. So, how can we pretend to generate high-quality data when the training data has so many biases? I consider explainability to be a key to solving this problem. Because the learned models and the mined patterns are interpretable, an expert can identify such biases and remove them before the generation.

And this is not a theoretical consideration: we actually observed it. During the experiments of FlowChronicle, we asked an expert to verify the patterns, and he identified an issue. One pattern contained three partial flows of HTTP connections, each starting from a different local IP address and targeting different public IP addresses. Intuitively, it should not happen: how would three different computers synchronize their communications often enough that a pattern is mined? We discovered that there was a bias in the script used for mimicking user activity: every minute, it would visit a random website. But since every computer has the same scripts, they would be synchronized. So, this pattern was in fact an experimental artifact, and not a genuine user behavior. By removing it from the set of patterns, we could be sure not to replicate this bias.

In my opinion, modularity is essential. Due to the wide variety of networking technologies, it is difficult to learn a single model to generate all kinds of data. With vertical modularity, we can have models that specialize in flow generation and others in packet generation. With horizontal modularity, we can have a generalist model for network flows and specialized models for Web or IoT flows, as well as one model per network protocol for packet generation. This allows for easier extension of the generation capabilities and each model is still small enough for manual verification.

Finally, low-overhead generation enables lower iteration times on-the-fly generation in honeypots and cyber ranges. This property was assessed with our challenge for the BreizhCTF 2025, where Fos-R was deployed on 750 virtual machines for a total of 23,000 cumulative hours with minimal overhead.

However, this work on synthetic data generation is still in its infancy, and further work is needed to make it truly useful for the community. These future works are presented in the next (and last) chapter.

Chapter 5

Conclusion and perspectives

I consider there is a looming evaluation crisis for intrusion detection systems. With the surge of machine learning, and the deep learning, the practices of the cybersecurity community have changed. The community largely abandoned evaluation on testbeds for static datasets containing training data sets for more convenient experiments. Machine learning in itself can be difficult to use and evaluate [40], leading to biased evaluations [3]. Furthermore, erroneous datasets can lead to overestimating the actual performances of detectors that obtain near-perfect scores, while the same datasets contain large issues (such as a whole mislabeled attack) that go undetected for years [48, 18]. I believe this can explain the current mismatch between academia and industry on intrusion detection systems, where methods developed in academia are largely ignored by the industry due to a lack of performances (notably false positive rates [20]).

All is not lost, of course: articles fixing past datasets have been published, new datasets with better documentation, characterization and reproducibility have been published (including our own [23, 5, 16]), and the Superviz project for PEPR Cybersecurity has a work package dedicated to the development of methodologies for characterizing datasets. Finally, the interest of the ANUBIS workshop I co-organize, which is focused on the evaluation of intrusion detection systems, indicates a growing awareness of these kinds of issues.

Synthetic data generation, while not a silver bullet, is another tool towards better IDS evaluation, in complement to better testbeds and other means of accessing real data. But because synthetic data generation can produce months-long datasets in a few minutes, it allows for quicker iterations and more personalized datasets. However, realism, diversity, utility and compliance still need to be improved. As mentioned earlier, there are still many assumptions and limitations to our work on synthetic data generation. As this work is multidisciplinary, with contributions to the networking, cybersecurity and artificial intelligence communities, I'll structure the future work into three sections, one for each community.

Cybersecurity: towards useful synthetic datasets and generation Several use cases could benefit from synthetic data generation, each with its own challenges. Using a synthetic data generator, it could be feasible to create datasets focused on the evaluation of IDS in the presence of concept drift (i.e., a non-stationary distribution), be it gradual, incremental or sudden. Concept drift is, in fact, always present in IT systems and the most common remedy is retraining regularly (e.g., each week) a model, which is wasteful. Creating such datasets (like we did with Superviz26-SQL) could facilitate and encourage research in this direction. I am currently working with IMT Atlantique on this subject.

More and more, datasets include both network traffic and systems logs. Synthetic system log generation has received less attention than traffic generation, but a few works propose to generate provenance graphs [14]. We will work on this topic, and more precisely on how to generate joint and consistent system and network data, in the Ph.D. of Antoine Cellier.

Another application is the injection of synthetic traffic into a honeynet (i.e., a network

without any users used to observe or deceive the attackers). This can increase the realism of the honeynet and make it look more innocuous to the attackers. We showed the technical applicability of this method by injecting synthetic traffic during our BreizhCTF 2025 challenge. I am working on this subject with Daishi Kondo from the University of Tokyo in the context of the DeceptIA associate team.

Finally, our work focuses on the generation of benign network traffic for the reasons presented earlier. However, an evaluation dataset must also include attacks. I recently obtained a chair at the SequoIA AI cluster on the application of style transfer for inserting measured attacks into synthetic data.

Networking: towards a more realistic generation Our work on synthetic generation, so far, does not take into account the structure of the underlying networks. Taking into account congestion could lead to better model the impact of some DDoS attacks on the quality of service on a network.

There is also no routing and the observed data do not correspond to any particular observation points of the network, so, for example, we cannot replicate behavior such as capturing only one side of a conversation because the other side uses an unmonitored route. We also assume the networking equipments are stateless, even if nowadays many services are cached. For example, a DNS server will not recursively resolve a domain if it was already resolved within some time-to-live duration.

Finally, our generation framework is dedicated to IT networks, even though there are many other types of networks of interest for cybersecurity, such as IoT, OT, and 5G networks.

Artificial intelligence: reducing the semantic gap A "semantic gap" denotes the difference of semantics between the raw data (pixel, bytes on a disk, packets, etc.) and the abstractions typically used to describe them in a useful manner (respectively, the subject of an image, a file in a file system, a videoconference call, etc.). Because the models are learned from raw observations, we aim to learn these abstractions.

A current limitation is that application payloads are replayed and sampled independently, leading to flows that can have inconsistent semantics (for example, a DNS answer can pertain to a different domain that the requested one) and seriously limit diversity (an FTP connection can only access files seen in the training data, for example). Even though these protocols are generally documented, it may be interesting to learn the protocol grammars to better generate messages. Some existing works [72] show great promise, but LLM could be of huge help as well.

Better payload generation could also help with another use case: to anonymize a private dataset. By reproducing similar behaviors but modifying the network topology and IP address ranges, one could share an otherwise private dataset.

While all of these ideas are incremental, I believe they all contribute to the same end goal: datasets with high-quality, reproducible, with explainable and auditable models, that can help produce intrusion detections systems with better capabilities, with higher transferability to the industry and, overall, a higher societal value.

Bibliography

[2] Sophie Fortz, Fatemeh Ghassemi, Léo Henry, Falk Howar, Thomas Neele, Jurriaan Rot, and Marnix Suilen. "A research agenda for active automata learning". In: *International Journal on Software Tools for Technology Transfer* (2026), pp. 1–22.

[6] Talha Abrar, Ahmad Shamail, Mohammad Jaffer Iqbal, Amaan Ahmed, Muhammad Abdullah, Muhammad Shayan, Fareed Zaffar, Thomas Pasquier, David Eyers, and Ashish Gehani. "On the Reproducibility of Provenance-based Intrusion Detection that uses Deep Learning". In: *Conference on Reproducibility and Replicability (REP'25)*. ACM. 2025.

[9] Tristan Bilot, Baoxiang Jiang, Zefeng Li, Nour El Madhoun, Khaldoun Al Agha, Anis Zouaoui, and Thomas Pasquier. "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.

[11] Javier Aday Delgado-Soto, Jorge E López de Vergara, Iván González, Daniel Perdices, and Luis de Pedro. "GPT on the wire: Towards realistic network traffic conversations generated with large language models". In: *Computer Networks* 265 (2025), p. 111308.

[13] Patrik Goldschmidt and Daniela Chudá. "Network intrusion datasets: A survey, limitations, and recommendations". In: *Computers & Security* 156 (2025), p. 104510.

[14] Yi Huang, Yao Guo, Xiangqun Chen, and Ding Li. "Provsyn: Synthesizing provenance graphs for data augmentation in intrusion detection systems". In: *arXiv preprint arXiv:2506.06226* (2025).

[15] Baoxiang Jiang, Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, Anis Zouaoui, Shahrear Iqbal, Xueyuan Han, and Thomas Pasquier. "ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems". In: *Security Symposium (USENIX Sec'25). USENIX*. 2025.

[17] Maxime Lanvin and Frédéric Majorczyk. "Get out of DEDALE with RESCOUSSE: a New Dataset and Testbed for Evaluating the Detection of APT attacks among Network and System Logs". In: *European Symposium on Research in Computer Security*. Springer. 2025, pp. 3–23.

[18] Frédéric Majorczyk, Barbara Pilastre, and *Fanny Dijoud*. "A New Hope for DARPA OpTC". In: *2025 Annual Computer Security Applications Conference Workshops (ACSAC Workshops)*. IEEE. 2025, pp. 551–561.

[20] Shahroz Tariq, Mohan Baruwal Chhetri, Surya Nepal, and Cecile Paris. "Alert fatigue in security operations centres: Research challenges and opportunities". In: *ACM Computing Surveys* 57.9 (2025), pp. 1–38.

[24] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. "KAIROS: Practical Intrusion Detection and Investigation using Whole-system Provenance". In: *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2024, pp. 3533–3551.

[26] Zian Jia, Yun Xiong, Yuhong Nan, Yao Zhang, Jinjing Zhao, and Mi Wen. "MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning". In: *33rd USENIX Security Symposium (USENIX Security 24)*. 2024, pp. 5197–5214.

[27] Francisco S. Melícias, Tiago F. R. Ribeiro, Carlos Rabadão, Leonel Santos, and Rogério Luís De C. Costa. "GPT and Interpolation-Based Data Augmentation for Multiclass Intrusion Detection in IIoT". en. In: *IEEE Access* 12 (2024), pp. 17945–17965. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3360879. URL: https://ieeexplore.ieee.org/document/10418592/ (visited on 06/15/2025).

[28] Gabin Noblet, Cédric Lefebvre, Philippe Owezarski, and William Ritchie. "NetGlyph: representation learning to generate network traffic with transformers". In: *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE. 2024, pp. 1–9.

[29] Mati Ur Rehman, Hadi Ahmadi, and Wajih Ul Hassan. "FLASH: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning". In: *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2024, pp. 139–139.

[34] Maximilian Wolf, Dieter Landes, Andreas Hotho, and Daniel Schlör. "Systematic Evaluation of Synthetic Data Augmentation for Multi-class NetFlow Traffic". en. In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD)* (Aug. 2024). (Visited on 06/15/2025).

[35] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. "Sok: Pragmatic assessment of machine learning for network intrusion detection". In: *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2023, pp. 592–614.

[36] Sowmya Myneni, Kritshekhar Jha, Abdulhakim Sabur, Garima Agrawal, Yuli Deng, Ankur Chowdhary, and Dijiang Huang. "Unraveled—A semi-synthetic dataset for Advanced Persistent Threats". In: *Computer Networks* 227 (2023), p. 109688.

[40] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. "Dos and don'ts of machine learning in computer security". In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 3971–3988.

[42] Lénaïg Cornanguer, Christine Largouët, Laurence Rozé, and Alexandre Termier. "TAG: learning timed automata from logs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 4. 2022, pp. 3949–3958.

[43] Drake Cullen, James Halladay, Nathan Briner, Ram Basnet, Jeremy Bergen, and Tenzin Doleck. "Evaluation of Synthetic Data Generation Techniques in the Domain of Anonymous Traffic Classification". en. In: *IEEE Access* 10 (2022), pp. 129612–129625. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3228507. URL: https://ieeexplore.ieee.org/document/9980373/ (visited on 06/15/2025).

[44] Van-Hoang Le and Hongyu Zhang. "Log-based anomaly detection with deep learning: how far are we?" In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE '22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, 1356–1367. ISBN: 9781450392211. DOI: 10.1145/3510003.3510155. URL: https://doi.org/10.1145/3510003.3510155.

[46] Aina Toky Rasoamanana, Olivier Levillain, and Hervé Debar. "Towards a systematic and automatic use of state machine inference to uncover security flaws and fingerprint TLS stacks". In: *European symposium on research in computer security*. Springer. 2022, pp. 637–657.

[51] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. "Practical gan-based synthetic ip header trace generation using netshare". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. 2022, pp. 458–472.

[52] Liat Antwarg, Ronnie Mindlin Miller, Bracha Shapira, and Lior Rokach. "Explaining anomalies detected by autoencoders using Shapley Additive Explanations". In: *Expert systems with applications* 186 (2021), p. 115736.

[53] Gints Engelen, Vera Rimmer, and Wouter Joosen. "Troubleshooting an intrusion detection dataset: the CICIDS2017 case study". In: *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2021, pp. 7–12.

[55] Arnaud Rosay, Florent Carlier, Eloïse Cheval, and Pascal Leroux. "From CIC-IDS2017 to LYCOS-IDS2017: A corrected dataset for better performance". In: *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. 2021, pp. 570–575.

[61] Laetitia Leichtnam, Eric Totel, Nicolas Prigent, and Ludovic Mé. "Sec2graph: Network attack detection based on novelty detection on graph structured data". In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2020, pp. 238–258.

[62] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language Models are Unsupervised Multitask Learners". In: (2019).

[63] M. Ring, D. Schlör, D. Landes, and A. Hotho. "Flow-based network traffic generation using Generative Adversarial Networks". In: *Computers & Security* 82 (2019), pp. 156–172. DOI: 10.1016/j.cose.2018.12.012.

[64] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. "Modeling Tabular data using Conditional GAN". In: *Advances in Neural Information Processing Systems*. 2019.

[68] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. "Flow-based benchmark data sets for intrusion detection". In: *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS)*. ACPI South Oxfordshire, UK. 2017, pp. 361–369.

[71] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[72] Georges Bossert, Frédéric Guihéry, Guillaume Hiet, et al. "Netzob: un outil pour la rétroconception de protocoles de communication". In: *SSTIC 2012* (2012), p. 43.

[73] Christian Rossow, Christian J Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten Van Steen. "Prudent practices for designing malware experiments: Status quo and outlook". In: *2012 IEEE symposium on security and privacy*. IEEE. 2012, pp. 65–79.

[74] Joao Antunes, Nuno Neves, and Paulo Verissimo. "Reverse engineering of protocols from network traces". In: *2011 18th Working Conference on Reverse Engineering*. IEEE. 2011, pp. 169–178.

[75] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. "A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data". In: *Grammatical Inference: Theoretical Results and Applications: 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings 10*. Springer, 2010, pp. 203–216.

[76] Peter Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[77] Samuel T King and Peter M Chen. "Backtracking intrusions". In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 2003, pp. 223–236.

[78] Paul Li and Ming Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.

[79] B John Oommen and Richard KS Loke. "Pattern recognition of strings with substitutions, insertions, deletions and generalized transpositions". In: *Pattern Recognition* 30.5 (1997), pp. 789–800.

[80] Dana Angluin. "Learning regular sets from queries and counterexamples". In: *Information and computation* 75.2 (1987), pp. 87–106.

[81] James P Anderson. "Computer security threat monitoring and surveillance". In: *Technical Report, James P. Anderson Company* (1980).

[82] Karl Popper. "The logic of scientific discovery". In: (1959).

Publications

Ph.D. students I co-supervised are *italicized*.

Journals

[7] Eric Alata and **Pierre-François Gimenez**. "A theory of injection-based vulnerabilities in formal grammars". In: *Theoretical Computer Science* (2025), p. 115554.

[41] Fabien Charmet, Harry Chandra Tanuwidjaja, Solayman Ayoubi, **Pierre-François Gimenez**, Yufei Han, Houda Jmila, Gregory Blanc, Takeshi Takahashi, and Zonghua Zhang. "Explainable artificial intelligence for cybersecurity: a literature survey". In: *Annals of Telecommunications* 77.11 (2022), pp. 789–812.

[45] Tomás Concepción Miranda, **Pierre-François Gimenez**, Jean-François Lalande, Valérie Viet Triem Tong, and Pierre Wilke. "Debiasing android malware datasets: How can i trust your results if your dataset is biased?" In: *IEEE Transactions on Information Forensics and Security* 17 (2022), pp. 2182–2197.

[56] **Pierre-François Gimenez**, Jonathan Roux, Eric Alata, Guillaume Auriol, Mohamed Kaaniche, and Vincent Nicomette. "RIDS: Radio intrusion detection and diagnosis system for wireless communications in smart environment". In: *ACM Transactions on Cyber-Physical Systems* 5.3 (2021), pp. 1–1.

[60] Hélène Fargier, **Pierre-François Gimenez**, and Jérôme Mengin. "Experimental evaluation of three value recommendation methods in interactive configuration". In: *Journal of Universal Computer Science* 26.3 (2020), pp. 318–342.

Conference Proceedings

[3] *Fanny Dijoud*, **Pierre-François Gimenez**, Michel Hurfin, Frédéric Majorczyk, and Barbara Pilastre. "GRAAL: GRAph-based Analysis of Logs for Advanced AI-based Intrusion Detection Systems". In: *EuroS&P 2026-11th IEEE European Symposium on Security and Privacy*. 2026.

[4] *Grégor Quétel*, **Pierre-François Gimenez**, Thomas Robert, and Laurent Pautet. "Parser Instrumentation for Semantic-Aware Applicative Intrusion Detection". In: *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer Nature Switzerland Cham. 2026, pp. 359–373.

[10] Lénaïg Cornanguer and **Pierre-François Gimenez**. "TADAM: Learning Timed Automata from Noisy Observations". In: *Proceedings of the 2025 SIAM International Conference on Data Mining (SDM)*. Society for Industrial and Applied Mathematics. 2025, pp. 114–123.

[16] Sébastien Kilian, Valérie Viet Triem Tong, Jean-François Lalande, Frédéric Majorczyk, Alexandre Sanchez, Natan Talon, Pierre-Victor Besson, Hélène Orsini, Pierre Lledo, and **Pierre-François Gimenez**. "CasinoLimit: An Offensive Dataset Labeled with MITRE ATT&CK Techniques". In: *The 28th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. 2025.

[22] **Pierre-François Gimenez**, Sarath Sivaprasad, and Mario Fritz. "Certifiably robust malware detectors by design". In: *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer Nature Switzerland Cham. 2025, pp. 125–139.

[25] Joscha Cüppers, *Adrien Schoen*, Gregory Blanc, and **Pierre-François Gimenez**. "Flowchronicle: synthetic network flow generation through pattern set mining". In: *Proceedings of the ACM on Networking* 2.CoNEXT4 (2024), pp. 1–20.

[30] **Pierre-François Gimenez** and Jérôme Mengin. "Learning Conditional Preference Networks: An Approach Based on the Minimum Description Length Principle". In: (2024).

[37] **Pierre-François Gimenez** and Jérôme Mengin. "Conditionally Acyclic CO-Networks for Efficient Preferential Optimization". In: *26th European Conference on Artificial Intelligence (ECAI 2023)*. Vol. 372. 10.3233/FAIA230352. Frontiers in Artificial Intelligence and Applications. 2023, pp. 843–850.

[38] *Maxime Lanvin*, **Pierre-François Gimenez**, Yufei Han, Frédéric Majorczyk, Ludovic Mé, and Eric Totel. "Towards understanding alerts raised by unsupervised network intrusion detection systems". In: *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*. 2023, pp. 135–150.

[39] *Vincent Raulin*, **Pierre-François Gimenez**, Yufei Han, and Valérie Viet Triem Tong. "BAGUETTE: Hunting for Evidence of Malicious Behavior in Dynamic Analysis Reports". In: *SECRYPT 2023-20th International conference on security and cryptography*. 2023, pp. 1–8.

[48] *Maxime Lanvin*, **Pierre-François Gimenez**, Yufei Han, Frédéric Majorczyk, Ludovic Mé, and Éric Totel. "Errors in the CICIDS2017 dataset and the significant differences in detection performances it makes". In: *International Conference on Risks and Security of Internet and Systems*. Springer Nature Switzerland Cham. 2022, pp. 18–33.

[57] *Vincent Raulin*, **Pierre-François Gimenez**, Yufei Han, Valérie Viet Triem Tong, and Léopold Ouairy. "GUI-Mimic, a cross-platform recorder and fuzzer of Graphical User Interface". In: *GreHack 2021-9th International Symposium on Research in Grey-Hat Hacking*. 2021, pp. 1–7.

[58] Malcolm Bourdon, **Pierre-François Gimenez**, Eric Alata, Mohamed Kaaniche, Vincent Migliore, Vincent Nicomette, and Youssef Laarouchi. "Hardware-performance-counters-based anomaly detection in massively deployed smart industrial devices". In: *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*. IEEE. 2020, pp. 1–8.

[59] Aliénor Damien, **Pierre-François Gimenez**, Nathalie Feyt, Vincent Nicomette, Mohamed Kaâniche, and Eric Alata. "On-board diagnosis: a first step from detection to prevention of intrusions on avionics applications". In: *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE. 2020, pp. 358–368.

[65] Hélène Fargier, **Pierre-François Gimenez**, and Jérôme Mengin. "Learning lexicographic preference trees from positive examples". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

International Workshop Proceedings

[1] Eric Alata and **Pierre-François Gimenez**. "JunId: Empowering Static Analysis with Asymptotically Optimal Intersection with Incomplete Sentences". In: *2026 IEEE Symposium on Security and Privacy Workshops (SPW)*. IEEE. 2026, pp. 26–38.

[5] *Grégor Quétel*, **Pierre-François Gimenez**, Thomas Robert, and Laurent Pautet. "Superviz26-SQL: A Multi-Domain Benchmark for SQL Attack Detection". In: *ESORICS 2026 International Workshops*. Springer Nature Switzerland Cham. 2026.

[8] Eric Alata and **Pierre-François Gimenez**. "Towards programming languages free of injection-based vulnerabilities by design". In: *2025 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2025, pp. 38–55.

[12] Rémi Garcia, Abdelkader Lahmadi, **Pierre-François Gimenez**, and Charles Sala. "ROSCA: Robust and Scalable Security Alert Correlation and Prioritisation using the MITRE ATT&CK Framework". In: *Proceedings of the First International Workshop on Analytics, Telemetry, and Cybersecurity for HPCC*. 2025, pp. 30–37.

[21] **Pierre-François Gimenez**. "Synthetic Network Traffic Generation for Intrusion Detection Systems: a Systematic Literature Review". In: *European Symposium on Research in Computer Security*. Springer Nature Switzerland Cham. 2025, pp. 102–118.

[23] *Grégor Quétel*, Eric Alata, **Pierre-François Gimenez**, Thomas Robert, and Laurent Pautet. "Superviz25-SQL: high-quality dataset to empower unsupervised SQL injection detection systems". In: *ESORICS 2026 International Workshops*. Springer Nature Switzerland Cham. 2025, pp. 45–64.

[33] *Adrien Schoen*, Gregory Blanc, **Pierre-François Gimenez**, Yufei Han, Frédéric Majorczyk, and Ludovic Me. "A tale of two methods: Unveiling the limitations of gan and the rise of bayesian networks for synthetic network traffic generation". In: *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2024, pp. 273–286.

[69] Hélène Fargier, **Pierre-François Gimenez**, and Jérôme Mengin. "Recommendation for product configuration: an experimental evaluation". In: *18th International Configuration Workshop (CWS 2016) within CP 2016: 22nd International Conference on Principles and Practice of Constraint Programming*. 2016, pp–9.

National Conferences and Journals

[31] *Fanny Dijoud*, **Pierre-François Gimenez**, Michel Hurfin, Frédéric Majorczyk, and Barbara Pilastre. "Survey on system-level graph-based and anomaly-based intrusion detection". In: *RESSI 2024-Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information*. 2024, pp. 1–2.

[32] *Grégor Quétel*, Eric Alata, **Pierre-François Gimenez**, Laurent Pautet, and Thomas Robert. "A parser-based data collector for intrusion detection". In: *RESSI 2024-Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information*. 2024, pp. 1–2.

[47] *Maxime Lanvin*, **Pierre-François Gimenez**, Yufei Han, Frédéric Majorczyk, Ludovic Mé, and Éric Totel. "Detecting APT through graph anomaly detection". In: *RESSI 2022-Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information*. 2022, pp. 1–3.

[49] *Vincent Raulin*, **Pierre-François Gimenez**, Yufei Han, and Valérie Viet Triem Tong. "Towards a representation of malware execution traces for experts and machine learning". In: *RESSI 2022-Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information* (2022), pp. 1–3.

[50] *Adrien Schoen*, Gregory Blanc, **Pierre-François Gimenez**, Yufei Han, Frédéric Majorczyk, and Ludovic Mé. "Towards generic quality assessment of synthetic traffic for evaluating intrusion detection systems". In: *RESSI 2022-Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information*. 2022, pp. 1–3.

[66] Hélène Fargier, **Pierre-François Gimenez**, and Jérôme Mengin. "Recommandation par inférence bayésienne. application à la configuration de produit". In: *Revue des Sciences et Technologies de l'Information-Série RIA: Revue d'Intelligence Artificielle* 32.1 (2018), pp. 39–74.

[70] Hélène Fargier, **Pierre-François Gimenez**, and Jérôme Mengin. "Recommendation using Bayesian inference for product configuration". In: *8èmes Journées Francophones sur les Réseaux Bayésiens et les Modèles Graphiques Probabilistes (JFRB 2016)*. 2016.

Miscellaneous

[19] Malcolm Murray, Henry Papadatos, Otter Quarks, **Pierre-François Gimenez**, and Simeon Campos. "Mapping ai benchmark data to quantitative risk estimates through expert elicitation". In: *arXiv preprint arXiv:2503.04299* (2025).

[54] Tomas Concepcion Miranda, **Pierre-Francois Gimenez**, Jean-Francois Lalande, Valèrie Viet Triem Tong, and Pierre Wilke. "Dada: Debiased Android DAtasets". In: *IEEE Dataport* (2021).

[67] **Pierre-François Gimenez**. "Apprentissage de préférences en espace combinatoire et application à la recommandation en configuration interactive.(Preferences learning in combinatorial spaces and application to recommandation in interactive configuration)." PhD thesis. Paul Sabatier University, Toulouse, France, 2018.

GRAAL: GRAph-based Analysis of Logs for Advanced AI-based Intrusion Detection Systems

Fanny Dijoud
Univ. Rennes, Inria, IRISA
fanny.dijoud@inria.fr

Pierre-François Gimenez
Univ. Rennes, Inria, IRISA
pierre-francois.gimenez@inria.fr

Michel Hurfin
Univ. Rennes, Inria, IRISA
michel.hurfin@@inria.fr

Frédéric Majorczyk
DGA-MI
frederic.majorczyk@def.gouv.fr

Barbara Pilastre
AMIAD
barbara.pilastre@intradef.gouv.fr

***Abstract*—Intrusion Detection Systems (IDS) are essential tools for detecting and analyzing malicious system activity. Anomaly-based IDS have gained popularity due to their ability to detect zero-day attacks, unlike signature-based IDS. Although this approach is promising, recent AI-based IDS still suffer from high false positive rates, non-scalability and biases, which limit their practicality in real-world deployments.**

This paper presents GRAAL, an end-to-end unsupervised graph-based anomaly-based IDS that allows a scalable multi-level detection, achieves a low false positive rate, and provides intuitive and interpretable outputs to assist analysts in threat detection and investigation. GRAAL proposes a method to extract features from heterogeneous provenance graphs, using a combination of structural and attribute embeddings. These vectors are then processed by multiple autoencoder models to detect anomalies at both graph and system entity levels. GRAAL's models leverage the relationship between the graph and entity levels, sharing knowledge through transfer learning and combining their results. We compare GRAAL against six state-of-the-art IDS on several datasets. These comparisons reveal methodological and reproducible biases in the evaluation of the current IDS and lead us to define best practices. To perform a comprehensive comparison, we mitigate biases still present in these state-of-the-art IDS. These extensive evaluations of GRAAL show that GRAAL outperforms these IDS, with a higher precision.

***Index Terms*—Intrusion detection system, anomaly detection, autoencoder, system log**

1. Introduction

An Intrusion Detection System (IDS) monitors system activity to detect malicious behaviors, such as Advanced Persistent Threats (APT). These attacks are sophisticated, long-lasting and often combine several advanced techniques and tactics. Because of their stealth and persistence, APT are difficult to detect. In recent years, deep-learning techniques have been promisingly applied to APT' detection, via unsupervised anomaly-based detection [1]–[4].

While a signature-based IDS can only identify known attacks, anomaly-based IDS are well-suited for detecting undocumented attacks that may exploit zero-day vulnerabilities. Indeed, these IDS learn the legitimate system activity, which is not poisoned, and therefore do not need any knowledge of attacks. During detection, these IDS identify abnormal behaviors based on the degree of deviation from the learned baseline of normal activities. To effectively detect APT, graph-based anomaly-based IDS have recently emerged as a promising research direction [5]–[7]. These IDS use system raw logs to construct provenance graphs [8]. For host-based IDS, the graphs' nodes are system entities (e.g., processes, files) and edges are interactions between them (e.g., create, read). While a log is simply a sequence of lines (called events), a provenance graph highlights causal relationships and behavioral patterns among the involved system entities.

So far, graph-based anomaly-based host-based IDS have not been widely adopted for several reasons. First, their performances are not yet sufficient for widespread use in many contexts. They often exhibit high false positive rates, which can lead to alert fatigue [9], and false negatives, which undermine the system's purpose and trustworthiness. These unsatisfactory performances are likely due to two main factors: the nodes' types definition in the provenance graph, and the lack of consistency between different detection levels.

Indeed, many IDS [5], [6], [10] only consider a subset of possible types, removing certain types (e.g., registry). We reckon this removing leads to overlook critical details. In contrast, few others [7] consider all possible node types without prioritizing their importance. This can generate noise and thus have a negative impact on detection performance. We argue for a balanced approach that copes with heterogeneity, while recentralizing information around nodes that are central to system activity (e.g., processes). Regarding the detection levels, a recent IDS solution [7] considers two different detection levels: graph and system entity. Having more than one level of detection enables a more precise identification of malicious activities and therefore provides analysts with a more detailed explanation. However, the two levels of detection are managed separately, both during training and inference phases [7]. To obtain a more effective solution, we argue that an IDS should handle graph and entity detection levels in an integrated and cohesive manner, via transfer learning and combination of results.

The second reason for the lack of adoption of these IDS is that the performances reported by their authors can be overestimated due to biases in their evaluation process. The presence of such biases, together with their variation across different solutions, makes performance comparisons questionable. To ensure a meaningful overall comparison, we identify and also mitigate biases in the assessed solutions. Thus, our study also promotes a fair methodology and best practices.

Finally, current IDS are not scalable, as they fail to perform intrusion detection on a complete large dataset. All these obstacles, related to i) performance, ii) biases, and iii) scalability, must be addressed to achieve wider adoption.

In this work, we address the aforementioned challenges by introducing GRAAL (GRAph-based Analysis of Logs), a new provenance-based anomaly-based IDS. GRAAL turns system activity into highly heterogeneous graphs, keeping all information. Then GRAAL captures both structural and semantic aspect of the graphs, focusing on processes (the main actors in provenance graphs). Finally, GRAAL performs unsupervised anomaly detection at graph and system entity levels, by combining these two detection levels and leveraging transfer learning. We evaluate GRAAL against six baselines of the state of the art (GAE [11], Kairos [5], MAGIC [7], Flash [6], Orthrus [12] and Velox [13]) on three well-known datasets (Streamspot [14], Wget [15] and DARPA OpTC [16]). We conduct a methodological review of biases and identify several biases still present in the evaluation process of the compared systems. We therefore propose adjustments to these systems to enable a fair comparison. Evaluation results indicate that GRAAL can outperform state-of-the-art approaches with a significant precision improvement.

In summary, our contributions are the following:

- We propose GRAAL, an end-to-end unsupervised graph-based anomaly-based host-based IDS that relies on heterogeneous provenance graphs, proposes an extraction features method and offers a multi-level coordinated monitoring of system activity.
- We evaluate GRAAL against six state-of-the-art baselines on three widely used datasets, yielding very competitive and promising results.
- We investigate state-of-the-art evaluation methodologies, propose modifications to mitigate identified biases, and provide an open source implementation of these adjustments[1].
- We provide an open source implementation of GRAAL, to ensure the reproducibility and reusability of our approach, whether for improvement or comparison with other works of the community[1].

2. Background

This section introduces some basic concepts that are commonly used in unsupervised graph-based anomaly-based IDS: system log, provenance graphs, autoencoders, and graph autoencoders.

1. https://gitlab.inria.fr/fanny.dijoud/graal.git.

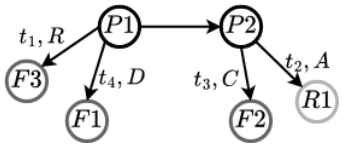


Figure 1: A provenance graph, with 3 different node types: process (P), files (F) and registry (R); and with 4 different action types: create (C), read (R), delete (D) and add (A). Actions are time-ordered with an associated timestamp t_i.

2.1. System Log and Provenance Graphs

Log files are detailed records of system activity. For example, when a process accesses or reads a file, this interaction between the two objects (the process and the file) is documented with a line of text in a log file. This line is called an event. It contains at least the action (e.g., ”read”), the source of the action (e.g., a process), the target of the action (e.g., a file), and the timestamp of the action. The format of these log files can vary depending on the system monitoring tool used to collect system activity and write the log files records: auditd, Windows event log or others.

In this work, we focus on the extended Cyber Analytics Repository (eCAR) format, based on MITRE's CAR [17]. CAR defines a 3-tuple object-action-fields to describe an event. The eCAR format can be extended by adding metadata about an event. These metadata provide additional details about the context such as host, user, command line and process information. This format (described in the appendix, Figure 9) is valuable because the additional metadata may provide critical insights when performing intrusion detection.

At host level, provenance graphs created from log files, as defined by King and Chen [8], are widely used to trace and highlight the dependencies between system objects [18], [19]. Indeed, these graphs offer a visual representation of the system activity that can lead to a better understanding and analysis of the system behavior. A provenance graph is a directed graph in which nodes represent system objects (also called entities) (e.g., processes, files) and edges represent all actions they can perform (e.g., create, read). Figure 1 shows an example of a simple provenance graph. The structure of a graph is defined by its nodes and edges. A graph may also include context details, or attributes of objects and actions, that are stored as features of nodes or edges. These edges or nodes features are useful semantic information for system monitoring (e.g., image path). Although the provenance graph definition is clear and shared by all, its construction depends on the available log files and the chosen graph-based IDS strategy.

2.2. Autoencoder and Graph Autoencoder

Autoencoders [20] are neural networks mostly used for representation learning [21] and anomaly detection applications [22]–[24]. Autoencoders are trained regarding two related steps: 1) to represent input data into a lower-dimensional space, called latent space, via an encoder and 2) to reconstruct input data from their lower-dimension space representation via a decoder. Autoen-

coders are trained to minimize the reconstruction error, between the input data and its related reconstructed output. The reconstruction error is computed using a loss function such as the Mean Squared Error (MSE). An encoder consists of a series of layers that gradually reduce the dimension of the input data. A decoder gradually increases the dimension of the latent space representation, using the same layers as the encoder, but in reverse order. A key advantage of autoencoders is that they are unsupervised neural networks: they do not need labeled data to perform their tasks. Indeed, they learn by comparing the input and the reconstructed output.

Graph autoencoders (GAE) [25] are end-to-end unsupervised neural networks designed to perform nodes/links prediction on graphs, or to achieve node/graph embedding. The GAE's structure is the same as the structure of an autoencoder: an encoder, a latent space, and a decoder. However, the encoder takes as input a graph, more precisely, the adjacency matrix and feature matrix. The adjacency matrix indicates whether the pairs of nodes are adjacent or not in the graph. The feature matrix contains features for all nodes in the graph. In the GAE, the encoder grasps the topological structure and semantic information of the input graph: multiple graph convolutions are used to reduce features' dimension and turn them into a node embedding matrix. Thus, the encoder projects the feature matrix into the latent space to return a node matrix embedding in lower dimension. The decoder aims to reconstruct the input adjacency matrix via an inner product between the latent node embeddings.

3. Related Work

This study focuses on intrusion detection using unsupervised anomaly detection methods designed to counter undocumented attacks. Consequently, we do not consider signature-based methods such as Poirot [26] and Holmes [27], nor supervised solutions such as Apt-kgl [28], Anubis [29] and Atlas [30]. Instead, we refer exclusively to recent state-of-the-art methods and AI models that leverage unsupervised machine learning techniques based on provenance graph representations. This choice is motivated by the realism of an unsupervised framework for intrusion detection use cases, where labeled data are scarce and intrusions cannot always be predefined. These approaches generally proceed in two main steps: (1) a learning phase where the model legitimates system activity, and (2) an inference phase where the model returns anomaly scores, which are compared against a user-defined threshold to identify anomalies when the score exceeds the threshold.

In this study we consider the baseline solution GAE from PyGOD library [11], [31] and five well-documented and well-referenced recent solutions as points of comparison: Kairos [5], Flash [6], Magic [7], Orthrus [12] and Velox [13]. We do not compare older IDS such as Streamspot (2016) [32], Unicorn (2020) [33], Threatrace (2022) [34], Pikachu (2022) [10]. Indeed, current IDS presented above already outperform results of these older IDS. Moreover, we do not consider IDS that do not provide an available implementation and thus cannot be used for comparison purposes, such as ProGrapher (2023) [35] and R-caid (2024) [36]. Finally, we focus on current IDS that are evaluated on at least one of the datasets used in this work. Indeed, these datasets are a commonly well-established basis for comparison. Thus we do not consider Shadewatcher (2022) [37].

GAE from PyGOD (NeurIPS'22) [11], [31]. PyGOD (Python Graph Outlier Detection) is an open-source python library for graph outlier detection. We focus on the GAE algorithm that is the graph outlier detection baseline (see Section 2).

Magic (USENIX Sec'24) [7]. Magic is a self-supervised APT detector using masked graph representation learning and outlier detection methods. It encodes nodes with a masked Graph Attention Network (GAT) [38] autoencoder learned for edges prediction. It performs detection on two completely separated levels: graph and entity. Graph representation is a pooling of the nodes embeddings.

Kairos (IEEE S&P'24) [5]. Kairos is a deep learning–based, graph-level anomaly detector that models system activities as provenance graphs over fifteen-minute windows. A classifier predicts edges from a Temporal Graph Neural Network (TGN) embeddings, and aggregated prediction errors yield graph-level anomaly scores.

Flash (IEEE S&P'24) [6]. Flash is a graph-based entity-level detector. It captures the semantic, temporal, and structural properties of provenance graphs through representation learning using Word2Vec [39] and graph neural networks (GNN). For anomaly detection, Flash detects anomalous nodes by comparing predicted and actual node types.

Orthrus (USENIX Sec'25) [12]. Orthrus is an anomaly-based entity-level detector. A lightweight TGN predicts edges type, from Word2Vec nodes and edges embeddings. To perform node detection, they apply both a K-means clustering and threshold methods.

Velox (USENIX Sec'25) [13]. The architecture of Orthrus serves as the foundation for Velox. Velox replaces the complex TGN-based encoder with a simple linear model to provide a light-weighted neural network.

We also notice a number of biases that lead to misleading results. Previous works already identify common pitfalls in the design, implementation, and evaluation of learning-based security systems [40]–[42]. Arp et al. [40] notice pitfalls in 30 top-tier current security papers that rely on machine learning for tackling different security issues. They provide statistics on the occurrence of their defined pitfalls but, without examining implementations, overlook several important ones. In addition, they do not focus on unsupervised IDS, making the ranking pitfalls maybe less relevant in our context. Recently, Abrar et al. [43] highlight reproducibility issues of current IDS, including Kairos [5], Magic [7] and Flash [6]. However, they make no attempt to correct the reproducibility problems they have identified, nor do they mention methodological biases. Another publication from Bilot et al. [13] notices nine shortcomings that prevent the practical deployment of provenance-based intrusion detection state-of-the-art systems. Yet these shortcomings are not very specific (e.g., "Unfair comparison with baselines").

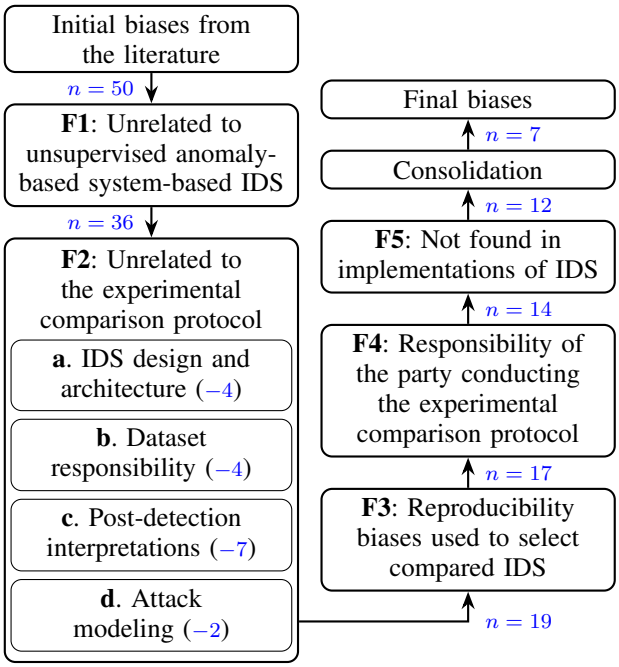


Figure 2: Methodological review of the biases.

4. Biases and Best Practices

An experimental need drives our methodology for identifying bias in related works: enabling fair comparisons of GRAAL with other IDS. Indeed, some recent work, such as Magic, Kairos, Flash, and Orthrus, have been criticized by Bilot et al. [13] for their unfair comparisons. While Bilot et al. focus their criticism on the comparison of tuned and untuned systems under their bias "Unfair comparison", we extend their analysis by examining additional implementation factors that can lead to unfair comparisons.

Our methodological process is summarized in the PRISMA diagram in Figure 2. Our initial set of biases is extracted from prior studies: [13], [40]–[43]. From this set, we apply successive filters (denoted F1 to F5) to retain only those relevant to fair comparison. The complete analysis is available in Appendix B (Table 8).

The first filter, F1, excludes 14 biases outside the scope of unsupervised IDS. The second filter, F2, excludes bias unrelated to the experimental comparison protocol and is composed of four subfilters (F2.a to F2.d). F2.a excludes bias tied to internal IDS design, such as "Overly complex architecture". F2.b excludes dataset bias such as "Label inaccuracy". F2.c excludes bias in result interpretation such as "CPU discussion". Finally, we limit our study to attacks present in the datasets, excluding aspects such as adaptive adversaries (F2.d), which relate to IDS robustness against adversarial attacks and are therefore out of scope.

The third filter, F3, concerns biases that prevent IDS execution, such as unavailable source code. Only the IDS that do not suffer from these biases have been selected for experimental evaluation, so, by construction, these biases will not be present in the evaluated IDS.

The fourth filter, F4, excludes biases that affect the experimental protocol of the comparison, such as "Inappropriate baseline" and "Not measuring instability". In this article, it is our responsibility to avoid these biases: even if they are presented in the related work, they do not prevent us from performing a fair evaluation.

The fifth filter, F5, is a manual review of the presence of biases in the implementation of IDS. Our approach has three steps: (1) attempt to reproduce the reported results, (2) manually analyze the source code to track biases, and (3) contact the authors to clarify any misunderstandings and obtain additional information or artifacts. No Large Language Model (LLM) has been involved in any of these steps. Details of the biases found for each of the compared IDS are available in Section 6.2.

Finally, we consolidate the remaining twelve biases by removing duplicates and refining them to better reflect observations in IDS implementations, resulting in seven final biases. For example, we merged "Biased parameter selection" [40] and "Impractical thresholding methods" [13], and we split "Data snooping" [40] into two separate biases, distinguishing between using test data during training and using ground truth information during detection.

We classify the seven resulting biases into two categories, A and B. Category A includes biases related to the training and detection methodologies, while category B encompasses biases that hinder reproducibility. The description of these biases is available in Table 1. Table 2 presents the presence of biases in the IDS we examined. Bias A3 and B7 are the most frequent biases. For bias A3, this is due to the common practice of evaluating IDS only on a subset of DARPA OpTC, the largest available dataset. Kairos [5], Flash [6], Orthrus [12] and Velox [13] only considered a subset of the hosts (either 3 or 6, all

TABLE 1: Description of the 7 final biases

Bias	Description
A1	**The division of the dataset** into training, validation, and test sets must be **clearly defined and well documented**. In the case of temporal data, the split must preserve chronological order: timestamps cannot be shuffled across sets, i.e., each set should correspond to consecutive time periods.
A2	**Test data** must not be used during the **training phase**.
A3	A model must be evaluated on the **entire test set**, without leveraging any prior knowledge of the attacks it contains.
A4	**Ground truth** must not be used during the **detection phase**, but only to analyze the accuracy of the detection results.
A5	All **parameters** of a solution must be clearly defined and documented. The method used to determine their values must be explicitly stated and **must never rely on knowledge of the test data**. Among the parameters, threshold value requires special attention as it significantly impacts the detection performance. Therefore, the process for setting its value must be fair, rigorous and justifiable.
B6	Each dataset used, along with its associated ground truth (i.e., dataset labeling), must be **available**.
B7	All relevant **artifacts** must be shared to enable others to reproduce the detection results.

TABLE 2: Presence of biases in the analyzed IDS. ✗ indicates that a given approach exhibits this bias

IDS	A1	A2	A3	A4	A5	B6	B7
Magic [7]	✗	✗			✗		
Kairos [5]	✗		✗		✗	✗	✗
Flash [6]			✗	✗	✗		✗
Orthrus [12], [13]			✗				✗
Velox [13]			✗				✗
GRAAL							

TABLE 3: Number of datasets and comparators used by the authors to evaluate their IDS

IDS	Datasets	Comparators
Magic [7]	3	7
Kairos [5]	3	2
Flash [6]	4	2
Orthrus [12]	2	5
Velox [13]	3	8
GRAAL	3	6

compromised except one, out of more than 500 hosts). Moreover, Kairos and Flash are evaluated only on the days when these hosts are compromised, using a different test set for each host. For bias B7, its wide spread emphasizes the challenge of reproducing the detection results reported by the authors of IDS. Indeed, even if these five IDS are publicly available on GitHub repositories, some artifacts are missing to enable the exact reproduction of the results.

Since the code of the studied IDS is available, we could correct some of these biases ourselves. In particular, we re-trained models using a clear separation between training, validation and test data (A1); the test data is never used during any learning phase (A2); the entire test set is used for the evaluation (A3); and the ground truth is only used to compute evaluation metrics such as the F1-score (A4). Selecting an appropriate threshold for an anomaly detector is difficult (A5). So, to avoid any risk of putting other IDS at a disadvantage, we decided to keep their thresholds unchanged, even when they were methodologically unsound (for example, hard-coded or based on the test data). So, for such IDS, the reported results should be considered optimistic. To allow reproducibility, we identify one ground truth that can serve as reference for each dataset (B6).

To mitigate these biases, it was necessary to modify the IDS under study. We limited these modifications to the strict minimum. To ensure full transparency and reproducibility, all these modifications are documented, IDS by IDS, in Section 6.2, and the source codes of the modified versions have also been made publicly available[1].

In line with best practices, the number of datasets and compared IDS are reported in Table 3. We remark that for this community, an average of three datasets and five compared IDS constitutes a common practice for properly evaluating IDS performances.

5. The GRAAL Framework

In this section, we introduce GRAAL (GRAph-based Analysis of Logs), an unsupervised host-based anomaly-based graph-based IDS designed to detect APT attacks at two different levels: the graph level and the entity level. As shown in Figure 3, our approach consists of five steps:

(i) Provenance Graph Construction (Sec. 5.1). We first transform raw logs into a common format (i.e., eCAR format). Then, based on this log format, we construct successive provenance graphs that represent host activity over a defined time period (denoted T). Figure 3 depicts two distinct graphs constructed during the consecutive time windows 1 and 2. To illustrate the next step, we focus on a single provenance graph generated for an arbitrary time window.

(ii) Embedding (Sec. 5.2). For each graph, GRAAL focuses on a specific type of nodes (e.g., process). In Figure 3, the two processes are delimited by a dotted line. GRAAL analyzes a global view (at the graph level) as well as several local views (at entity level). GRAAL combines encoding and Word2Vec [39] embedding to create features vectors capturing both structural and semantic aspects of the graph. In the example, this second step produces three feature vectors: one for the entire graph (P) and one for each process of this graph ($P1$ and $P2$).

(iii) Global Graph Model (Sec. 5.3). GRAAL learns a global benign behavior through an autoencoder, using as input the set of feature vectors representing the generated graphs. A single model is trained on multiple graphs, regardless of their time windows and host machines. Reconstruction errors are computed by calculating the distance between the predicted and actual graphs' vector representations. When only this model is employed, we refer to it as GRAAL-g (rather than GRAAL) to clearly indicate a partial use of the overall solution, to perform detection at **g**raph level.

(iv) Local Entity Models (Sec. 5.3). GRAAL transfers the knowledge of the graph model (i.e., transfer GRAAL-g's weights) to learn specific benign behaviors at the entity level, using autoencoders extended with one fully connected layer. A different model can be trained for each host for accommodating different users or system usage. The autoencoder takes as input the feature vectors representing each selected entity of the graphs and computes a reconstruction error. When only these models are used, we refer to them as GRAAL-e because detection is performed at the **e**ntity level.

(v) Anomaly Detection (Sec. 5.4). Finally, GRAAL combines the reconstruction errors from both the global graph model and the local entity models. This combination of two levels is aimed to reduce false positives and enables a more analyst-friendly representation of results.

In the threat model assumed for GRAAL, we suppose that attackers want to take control and maintain their presence in the target system. We assume that the data used to train models do not contain any attackers' activity and are not poisoned. Moreover, we suppose that the logging system and the logs cannot be tampered by the attackers. We do not consider IDS adversarial attacks. This threat model is the most common among state-of-the-art IDS.

5.1. Provenance Graph Construction

As explained in Section 2.1, raw logs can have different formats depending on the operating systems and tools used to monitor the host activity. We decide to transform these raw logs into a common unique format that will be used as input for the construction of the graphs. It becomes easier and more consistent to build provenance graphs and perform intrusion detection upon various datasets. Future users only have to implement a new parser from raw logs to our chosen intermediate format to use the GRAAL framework. We choose to adopt the eCAR format which is the format already adopted in the DARPA OpTC dataset [16] (see Section 6.1). This format is rather comprehensive, allowing the logging of precise metadata, such as the command line of a process.

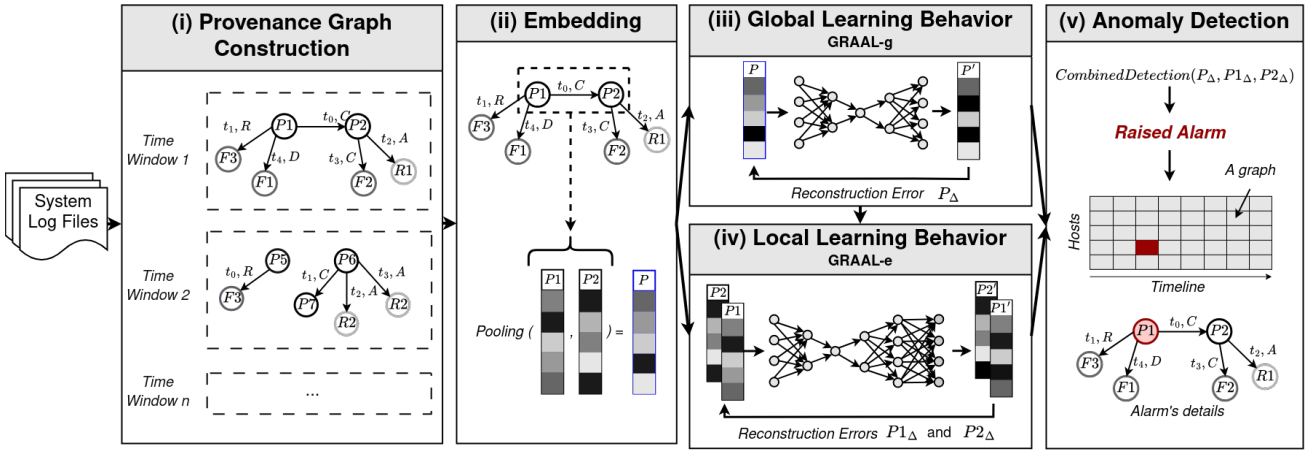


Figure 3: Overview of GRAAL framework.

As it is stored in JSON format, eCAR can be extended easily when needed.

GRAAL turns eCAR rows into a heterogeneous provenance graph by transforming each eCAR log event into a directed, typed, and timestamped edge, that links the actor node (i.e., the source of the action) to the object node (i.e., the object being acted upon). GRAAL's graphs can refer to various edge types and node types: the user can configure them as needed, simply by giving the list of the considered nodes' type and edges' type.

During the construction, to determine whether a log event maps to an existing node in the graph or requires the creation of a new node, we rely on unique identifiers. When such an identifier is not available in the raw log format, we use one of the metadata as an unique identifier, such as the file path for a file object, or the registry key for the registry object. Other types of metadata such as the process security identifier (SID) are stored as node attributes. Depending on the metadata and node types, they can be used for encoding or only for graph visualization.

Since we build provenance graphs that represent host activity over a period of T minutes, we add another attribute to each node to estimate its lifetime, which can represent useful information for intrusion detection. In the next section, we explain how this approximated lifetime is defined and how it is encoded into the feature vectors.

5.2. Encoding and Embedding

Once the provenance graphs are constructed, we apply different encoding and embedding techniques to extract information from the graphs and generate feature vectors. These vectors serve as inputs to our models.

The constructed graphs are heterogeneous: graph contains multiple nodes and edges types. To handle this heterogeneity, we decide to pre-select only one entity type and create feature vectors for the corresponding nodes. These vectors not only preserve the information from the represented nodes, but also as much information as possible about their neighbors. Even if neighbors contain node types other than the selected one. In general, we choose an entity of reference which is central in the host activity such as the "Process" object for the DARPA OpTC dataset [16]. In that dataset, a process is always the actor of an event. Thus each node of the provenance graph is either a process entity or directly linked to a process entity. Choosing to create features via encoding and embedding for these entities ensures that coverage of the whole graph is maintained.

In GRAAL, the detection is performed at two different levels: graph and entity. So we define two kinds of features vectors: graph vectors and entity vectors. Both vectors are composed of a structural and an attribute parts. The structural part encodes information about the graph structure (e.g., number and types of neighboring nodes). Whereas the attribute part encodes and embeds information about the attributes of nodes. When nodes' attributes are not available in the system logs, GRAAL only uses the structural part. To summarize, a graph composed of n selected-type nodes has thus one graph feature vector and n entity feature vectors. We explain the construction of these vectors in the paragraphs below.

Encoding of the structural part of a node. Here we present the composition of an entity feature vector. The differences with the graph feature vector are explained later. The structural part of an entity feature vector contains information about a node and its one-hop neighborhood. The structural part of the entity vector is split into four vectors: *x1*, *x2*, *x3* and *x4*. Figure 4 shows a graph and the entity vector of the process $P2$.

x1. The vector *x1* contains information about the entity type of the nodes present in the one-hop neighborhood, of the considered node. In Figure 4, the graph is composed of nodes related to three different entity types: process (P), file (F) and registry (R). The size of *x1* is equal to twice the number of types: 2×3. In the first half of the vector, we keep the number of neighbors of each type that are the targets of an outgoing edge. In Figure 4, $P2$ is actor upon one file $F2$ and one registry $R1$; so the first half of the vector is $[0, 1, 1]$. Symmetrically, the second half contains the number of neighbors of each type that are the sources of an incoming edge: $P2$ is an object under only one process $P1$ so this half of the vector is $[1, 0, 0]$.

x2. The vector *x2* is structured in the same way as *x1* but contains the cardinality of outgoing and incoming edge types connected to the node. In the example of Figure 4, the graph is composed of four edge types: create (C), read (R), delete (D) and add (A). So *x2*'s size is 2×4. The

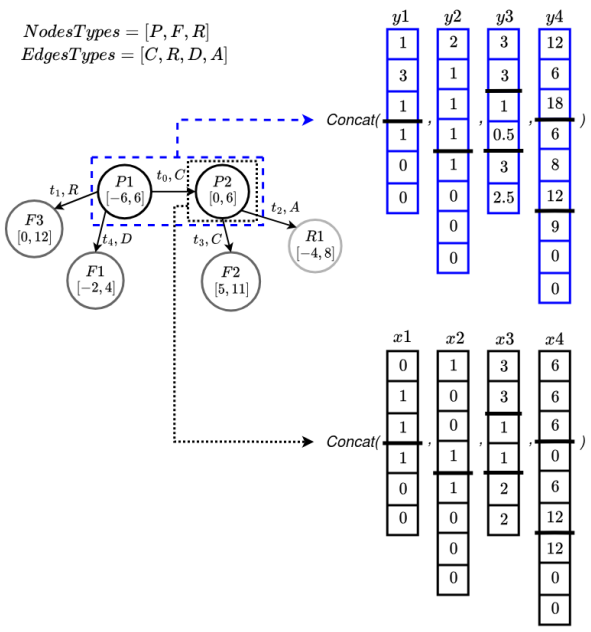


Figure 4: Example of an encoding of the structural part of the vectors. The encoding for the graph vector is framed in blue $(y1, y2, y3, y4)$. The encoding for the entity vector is framed in black $(x1, x2, x3, x4)$. Among others, each node has a type and a lifetime and each edge has a type and a timestamp.

first half of the vector focuses on edges where the entity is an actor and the second half focuses on edges where the entity is an object.

x3. The vector *x3* is split in three and contains information on degrees of the process node: degree, inbound degree and outbound degree. In an entity vector, the three values are duplicated. As explained afterward, this is not the case for a graph vector.

x4. The vector *x4* contains information on the lifetime of the entity and the lifetime of its one-hop neighborhood. By construction, our provenance graphs are associated to a time window of T minutes. During the construction of a graph, we add an attribute called "lifetime" to each entity. During the window of T minutes, the lifetime of a process created at timestamp t_c and terminated at timestamp t_t, will be $[t_c, t_t]$. However, there may be no event indicating the creation or the termination of a node during these T minutes. If there is only an event indicating the creation of the node at timestamp t_c, the lifetime of the node will be approximated by $[t_c, t_c + T]$. In the same way, if there is only an event indicating the termination of the node at timestamp t_t, the lifetime of the node will be approximated by $[t_t - T, t_t]$. If there are some events (that are not creation or termination) involving this node at the timestamps t_{e1}, t_{e2} and t_{e3}, then the approximated lifetime of the node is $[t_{e1} - T, t_{e3} + T]$. Indeed, t_{e1} is the first event and t_{e3} is the last event, that involve this node during the T minutes (described in the appendix, Figure 10). By construction, this approximation can never exceed $3T$ minutes. This lifetime attribute can distinguish entities spanning multiple graphs from those in a single graph. The vector *x4* is split in three tiers. The first tier will encode the duration of the lifetime of the node itself (replicated three times for the entity vector; for the graph vector, the values may be different). The second tier of the vector contains, for each node type, the mean of the duration of the lifetime of nodes that are the targets of an outgoing edge. The last tier is computed in the same way but for nodes that are the sources of an incoming edge.

Encoding/embedding of the attribute part of a node. Here, we detail the encoding/embedding of an entity feature vector. The differences with the graph vector are explained later. The attribute part of the entity feature vector contains information about its attributes. We focus on attributes that are critical for the security, such as the process' security identifier (SID). Depending on the dataset, these attributes may be present or not. For the Streamspot dataset [14], logs information are minimalist: the source object, the destination object, the event type and the timestamp. For this dataset, we are not able to construct the attribute part of the vectors. Thus, only the structural part is computed. Here, we present the choices made for the DARPA OpTC dataset [16] which contains the most precise event logs. For this dataset, we embed the image path (i.e., the path of the executable file), the parent image path (i.e., the path of the process' parent), and the command line, which also includes the arguments. We also encode the process' security identifier (SID) and the presence of the username in the image path and parent image path. For the embedding of paths and command line, we use two Word2Vec [39] models. These models are configured in the continuous bag of words (CBOW) mode: the model has to predict a target word given the surrounding words. The repository, file name, executable arguments and the files' extensions are considered as words for the models. For the SID encoding and the presence of the username in the different paths, we use a one-hot encoding based on the different number of classes. For the SID, we consider six different classes of the subauthority part of the SID: LocalSystem, LocalService, NetworkService, Domain, Windows Manager, and User-Mode Driver Framework. For the encoding of the presence of the username in the image path and parent image paths, we check for the presence of the string `/Users/` in the paths. We define four classes for the encoding: both, only parent, only child and none.

The graph feature vector. The graph vector is constructed by aggregating the entity feature vectors of that graph. For each dimension, we use different aggregation functions to compute the values, such as the sum, mean, min or max functions. In Figure 4, vectors *y1*, *y2*, *y3* and *y4* represent the structural part of the graph vector. For *y1* and *y2*, we use the sum function. In this example, the second dimension of vector *y1* is 3 because $F1$, $F2$ and $F3$ are targets of outgoing edges of processes $P1$ and $P2$. For *y3*, in each tier of the vector, we use the max function for the first dimension and the mean function for the second dimension. For example, the fifth dimension of *y3* is 3 as the maximum of outgoing edges of a process node is 3 ($P1$ edges) and the sixth dimension of *y3* is 2.5 as the mean value of the number of outgoing edges of a process node is $(3 + 2)/2$. Regarding *y4* which encodes the lifetime of the nodes and their neighbors, for the first tier of the vector, we use respectively the max, min and the sum functions for dimension 1, 2 and 3. For both the other tiers of *y4*, we use the mean function to aggregate

the different values. For the attribute part of the graph vector, we use the mean function for the embedding of the image path, the parent image path and command line, and we use the sum function for the one-hot encoding of the SID and the username presence in the paths.

This definition of these vectors leads to major advantages. Indeed, previous IDS only considered a subset of all possible type of nodes and edges. Thus, they overlook critical details. With our defined encoding vectors, we consider all types of nodes and edges. Thus, all the activity is contained in these vectors. Moreover, we focus on attributes that have concrete impact on the security, such as the SID, and thus eliminate potential noises from other attributes. The relevance of these defined vectors is empirically validated in Section 6.5. The other advantage of this embedding is that there is some homogeneity between a graph feature vector and an entity feature vector. Indeed, even if they represent different granularity levels of the graph, they have exactly the same vectors' size and each dimension has the same meaning, which is a key point for the learning strategy presented in Section 5.3.

5.3. Local Entity and Global Graph Models

We train two different models to perform anomaly detection at two different levels: graph and entity. The global graph model (or graph model for short) is trained with the graph feature vectors. We call this model GRAAL-g. In contrast, the local entity model (or entity model for short) is trained with the entity feature vectors. We call this model GRAAL-e. We first train the graph model GRAAL-g that uses an autoencoder composed by four linear-ReLU layers with the Mean Squared Error (MSE) loss function. Once this model is trained, we apply transfer learning by reusing this autoencoder as a basis for the entity models GRAAL-e. This is possible because of the homogeneity of the graph vectors and the entity vectors. More precisely, the entity models are composed of the four layers of the graph model and are extended with one fully connected layer at the end, as shown in Fig. 3. The weights of the graph autoencoder are frozen during entity model training; only the last layer of the entity models is updated. The loss function is also the MSE.

These models have few layers (only four and five), making GRAAL fast for training and inference, and light on resources consumption (as empirically shown in Section 6.5). Variation of GRAAL-g's number of layers (up to six) and variation of GRAAL-e's number of layers (up to nine) do not significantly impact GRAAL's performances (described in the appendix, Figure 13). Moreover, we choose the autoencoder architecture since it is a standard and widely used method for unsupervised anomaly detection [24], [44]. It is simple, efficient, and requires limited computational resources, allowing us to propose a more scalable solution, as experimentally assessed in Section 6.5.6. Besides, more complex models may not yield significant performance gains and can increase cost and the risk of overfitting [13], [40].

To our knowledge, it is the first time that transfer learning is used between two different granularity levels of the activity, in a context of intrusion detection. The purpose of this transfer learning is to make the entity model only learn the local information. Indeed, the global

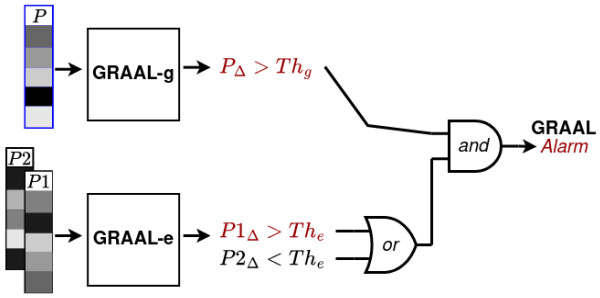


Figure 5: Anomaly detection strategy. Th_g is the threshold of GRAAL-g and Th_e is the threshold of GRAAL-e.

graph model learns the global behavior of a type of entity (e.g., processes), by looking at numerous of these entities in a single input. Once the global behavior is learned, the local entity model will only learn the small accepted local deviation of behavior of a particular entity. So, by reusing the latent space representation learned by the graph model, this approach ensures a consistency between the two kinds of models and allows for a much faster learning of the entity models. The relevance of this choice is empirically validated in Section 6.5.

It is possible to train not only one global model but multiple global models. The motivation is that machines used by different roles (e.g., administrative staff vs. system administrators) exhibit distinct usage patterns, which are reflected in their provenance graphs. However, in this work, we explore one case: constructing a single generic global graph model alongside a specialized local entity model for each individual host. By combining the two models (see Section 5.4), we aim to reduce false positives without significantly compromising the recall. Performance evaluations (see Section 6) confirm this expectation.

5.4. Anomaly Detection

The two types of model perform anomaly detection at two different levels: graph and entities. The graph model detects abnormal graphs (i.e., detection of abnormal activity over a given period), whereas the entity model detects abnormal system entities (i.e., detection of a process with abnormal activity). For both models, detection is based on reconstruction errors between the input and the output of the models. During inference, an alarm is typically raised when the reconstruction error exceeds a certain threshold. In GRAAL, we define a threshold for each model: one for the graph model and one for each entity model. For each model, the threshold is set to the maximum loss observed on the validation set. Under our threat model, we assume there are no attack in the training and validation sets, so the thresholds represent the worst reconstruction error for benign activity present in the training set.

Figure 5 provides an overview of the three frameworks (GRAAL, GRAAL-g and GRAAL-e) and the corresponding relations between them. The vector P is the graph feature vector of a graph, whereas vectors $P1$ and $P2$ are entity feature vectors that each represents an entity of the graph (the same vectors as in Figure 3). GRAAL-g computes the reconstruction error of a graph' representative

feature vector using the graph model. GRAAL-e computes the reconstruction error of each entity representative feature vector of this graph using the entity model. Thus, for a graph G that contains n entities, $n+1$ reconstruction errors are computed. If the reconstruction error of a graph is above the threshold Th_g computed by GRAAL-g, it is considered as abnormal. If the reconstruction error of a graph entity is above the threshold Th_e computed for GRAAL-e, it is considered as abnormal.

GRAAL combines the results of GRAAL-g and GRAAL-e to determine whether an alarm should be raised or not. If the graph G is tagged as abnormal by GRAAL-g and if at least one entity of this graph is tagged as abnormal by GRAAL-e, then G is considered as abnormal by GRAAL. The abnormal graphs of GRAAL are therefore equal to a subset of the abnormal graphs of GRAAL-g. Similarly, an anomaly detected at the entity level must be corroborated by an anomaly detected at the graph level. Otherwise, the entity tagged as abnormal by GRAAL-e is no longer considered as abnormal by GRAAL. Thus, the results' combination can lead to revising an initial decision from abnormal to normal if the suspicion is not confirmed by the model(s) associated to the dual detection level.

To present the results to the analyst, we propose two visualizations (illustrated at the bottom of box (v) in Figure 3): a timeline showing when and where an alarm was raised (described in the appendix, Figure 11) and a graph representation that identifies the abnormal processes.

6. Evaluation and Analysis

6.1. Datasets

We conduct experiments using three publicly available datasets, namely Streamspot [14], Wget [15], and DARPA OpTC [16]. These datasets enable us to compare GRAAL with many state-of-the-art detectors, as they form the foundation of many detectors' evaluations. Indeed, these datasets exhibit variations in volume, provenance, granularity and attacks, allowing the evaluation of many aspects of an IDS.

Streamspot dataset [14]. Streamspot is a simulated dataset collected and released by Manzoor et al. [32]. It contains 600 graphs distributed in six different scenarios. Five scenarios are benign host activity scenarios (watching YouTube, browsing CNN, downloading files, viewing emails, and playing video games), whereas the last one contains drive-by download attack scenarios. This dataset is widely used in the community and is a baseline used by 3/5 detectors we consider.

Wget dataset [15]. The Wget dataset, also called Unicorn, has been designed by Han et al. [33]. It contains 125 benign batches of logs and 25 supply-chain attacks batches collected with Camflow. Camflow saves logs directly as graphs, using their own graph definition: they define their own node and edge types. In these graphs, edge direction denotes the direction of information flow. 2/5 of the considered detectors are evaluated on this dataset.

DARPA OpTC dataset [16]. DARPA Operationally Transparent Cyber (OpTC) is another simulated dataset, collected and released by the Defense Advanced Research Projects Agency (DARPA). This dataset contains benign and malicious activity of 500 Windows hosts. It contains six days of benign activity and three days of combined benign and malicious activities. It includes three different attack scenarios. The first scenario impacts 17 hosts during the first day of the test set: attackers deployed a PowerShell Empire stager, escalated privileges, and stole credentials. Then they pivoted via WMI and reached the Domain Controller (DC) to extract user hashes. The second scenario is performed the second day, going through the night, and the beginning of the third day. It involves 7 other hosts. Attackers used phishing to deploy custom PowerShell Empire stagers and gained control of the DC with DeathStar. They established persistence, moved laterally via RDP, exfiltrated data, and set up overnight C2 access. The last scenario occurred the last day and targeted 2 other hosts: attackers exploited a vulnerable software update to deliver a malicious binary, gaining system-level access. Then they harvested credentials, established persistence, and attempted RDP access with a new admin account. As the dataset only described the attacks with vague descriptions in natural language, we rely on Nikulshin and Tahli's [45] detailed labels as the ground truth of this dataset. The volume, the richness of information and the numerous attacks contained in this dataset make it more realistic and more difficult than Streamspot or Wget. This dataset is widely used in the community and is a baseline for 4/5 considered detectors.

6.2. Handling Biases Across Detectors

We focus the comparison on six detectors. The details of each approach are available in Section 3. The current section exposes biases presented in Section 4 and changes made to alleviate them, while preserving the integrity of the detectors. The low-level details regarding the biases found into the implementation of IDS are provided in the Appendix, Section B.2. The impacts of the modifications are discussed in Section 6.5.

GAE [31]. For the GAE, the graphs are built in the same way as for GRAAL. As the features of the nodes need to be homogeneous, we only use the types of the nodes encoded as one-hot vectors and the timestamps.

Flash [6]. When attempting to retrieve Flash's results, we obtain results that are close but not identical to those reported for DARPA OpTC (see bias B7 described in Section 4). Moreover, Flash reduces the test set to three hosts (Hosts 051, 201 and 501) and only to the days each host is compromised (see bias A3). In this work, the test set is fixed to these three hosts during all the testing days, including days without attacks. Moreover, we notice that Flash uses the ground truth to return more relevant results. Indeed, Flash defines their true positive as the two-hop entities of the true positive nodes detected by their model. However, nodes detected by the model are supposed to be identified as true positive or false positive only to compute metrics (see bias A4). Thus, we remove the use of the ground truth, and the alerts are only the nodes detected without the extension to the two-hop entities. Their confidence threshold and similarity threshold are kept as hard-coded values proposed in the implementation (see bias A5).

Kairos [5]. We reproduce authors' results of Kairos for Streamspot. However, trained models are not available

for DARPA OpTC (see bias B7), and the training data are not clearly defined (see bias A1). We therefore train new models using the same training data that are used to train GRAAL. For DARPA OpTC, Kairos is evaluated on only for six hosts: five hosts are compromised during the attacks (Hosts 051, 201, 402, 501 and 660), whereas the last one is not compromised (Host 207). Kairos creates graphs representing every 15 minutes of system activity and performs graph-level detection. However, because batches are used to implement the creation of their graphs, some graphs can represent activity intervals slightly longer than 15 minutes. To compare the results of Kairos and GRAAL on the same activity, we slightly change Kairos' implementation to build graphs of exactly 15 minutes. Since the 15-minutes window length is chosen arbitrarily, we consider that this slight change should not impact the performance. Moreover, just as Flash, Kairos is evaluated on the six hosts only on the days they are compromised (see bias A3). To alleviate this bias, we evaluate Kairos on the three days of the testing set for all hosts. In its implementation, Kairos uses hard-coded thresholds, sometimes different between attacks and non-attack days (see bias A5). In this work we set a unique threshold for all hosts, but this threshold maximizes the F1-score and is thus favorable to Kairos. Finally, the labels used by Kairos on DARPA OpTC are not available (see bias B6). Thus, we use Nikulshin and Tahli's [45] labels (see Section 6.1).

Magic [7]. We reproduce authors' results of Magic for the Streamspot and Wget datasets. Regarding the graph-level detection, we notice that the split between the training and testing sets is not clear for both datasets. Indeed, it seems that some graphs used to train the model are also used to perform the detection. Thus, we slightly change the implementation to make a clear split (see bias A1 and A2). Moreover, as Magic does not perform detection on DARPA OpTC, we implemented a parser for DARPA OpTC to train and evaluate Magic on this dataset. For graph detection, we keep the parameters they used for the Streamspot dataset. Regarding the threshold, Magic uses a threshold that maximizes the F1-score (see bias A5). We keep this definition of the threshold.

Orthrus [12]. Orthrus is evaluated only on the same three hosts as Flash (see bias A3). Artifacts to perform detection with Orthrus were not available (see bias B7), thus we trained the model based on the provided implementation from Bilot et al. [13]. We do not change the parameters of Orthrus.

Velox [13]. Velox is evaluated only on the same three hosts as Flash and Orthrus (see bias A3). The configuration of the model used by authors of Velox is available. However, due to significant implementation instability, we can not reproduce the exact results from the authors' publication (see bias B7). We do not change the parameters of Velox.

6.3. Metrics

We evaluate the performances using common metrics: true positive (TP), false positive (FP), true negative (TN), false negative (FN), precision, recall, f1-score, the area under the ROC curve (AUC) and the attack detection precision score (ADP). We keep detailed metrics such as TP, FP, TN and FN, to enable a more concrete and comprehensive analysis. The ADP score has been recently defined by Bilot et al. [13] as a new metric for entity detection. ADP is a complementary metric to the AUC score, as both metrics are threshold-independent. As many approaches use a biased threshold, these scores allow to fairly compare the approaches.

6.4. Experimental Methodology

In the following, we explain the experimental methodology to compare GRAAL to state-of-the-art methods. More precisely, we seek to answer the following research questions:
RQ1: how do the approaches compare at the graph level?
RQ2: how do the approaches compare at the entity level?
RQ3: how effective is the node embedding?
RQ4: how complementary are the entity and graph levels?
RQ5: how effective is the transfer learning between levels?
RQ6: how scalable is GRAAL?

Here is the experimental protocol for each research question.

6.4.1. RQ1: How do the approaches compare at the graph level?. To answer that, we compare GRAAL, GAE, Kairos, Flash and Magic on the three datasets, on graph detections.

Wget is a graph-based dataset: the activity is already split in graphs and the labels are at graph level. So we perform only GRAAL graph-level detection. Moreover, this dataset contains Camflow graphs. These Camflow graphs do not contain a process node type. The camflow node type that comes closest to a process behavior, is the Task type. Thus, GRAAL selected node type is Task for this dataset. For this Wget dataset, all the detectors use a train ratio of 80%. We keep this original train ratio to train GRAAL and GAE. As this dataset contains only a few number of graphs (i.e., 150), we perform a five-fold cross-validation to reduce the 95% confidence interval of the results. Unlike other IDS that use biased thresholds, GRAAL sets the thresholds based on the validation sets: because this dataset is small, GRAAL's thresholds are estimated with another cross-validation protocol on the validation set, taking the median values.

Streamspot is also a graph-based dataset. For this dataset, two train ratios are used by current IDS. Kairos uses a train ratio of 25% while Magic and Flash use a train ratio of 80%. Here again, we keep their initial values and makes two different trainings for GRAAL and GAE using these two ratios. Just like we do for the Wget dataset, we perform a five-fold cross-validation to reduce the confidence interval of the results because Streamspot is also a small dataset (i.e., 600 graphs). The GRAAL's thresholds are also estimated as for the Wget dataset.

The DARPA OpTC dataset is an event-level dataset: the labels are at event level. To perform graph detection on this dataset, we considered that a graph is labelled "attack" if it contains at least one event labelled "attack". We perform graph detection with the same 15-minutes time windows as Kairos. We train all the approaches on the last four benign days and evaluate them on the complete testing set for the six hosts also selected by Kairos.

6.4.2. RQ2: How do the approaches compare at the entity level?. DARPA OpTC is the only dataset for which the labels are provided at the entity level. We compare GRAAL, Flash, Orthrus and Velox for this detection level. The results focus on the 3 hosts selected by Flash, Orthrus and Velox.

For DARPA OpTC, the selected nodes are processes. However, Flash, Orthrus and Velox take into account multiple node types in their results. To enable a fairer comparison, we limit the detection of these three IDS to process nodes. This choice facilitates consistent comparison with future IDS, as processes are always included in the node type set of an IDS, whereas other types are not systematically considered. As an example, Orthrus considers three node types: process, file and netflow. Whereas, Flash defines only two node types: process and object types. Thus, the intersection of the node types of these two IDS only contains one type: process. This choice preserves the integrity of the compared IDS, as we only apply this node type restriction when computing metrics.

6.4.3. RQ3: How effective is the node embedding?. We confirm the effectiveness of the embedding strategy presented in Section 5.2 by performing outlier detection with a classic One-Class Support Vector Machine (OCSVM) which takes our defined embedding vectors as inputs. To perform OCSVM graph detection, we consider the experiment described in Section 6.4.1, concerning the DARPA OpTC dataset. To perform OCSVM entity detection, we consider the experiment described in Section 6.4.2, that also concerns the DARPA OpTC dataset. Moreover we perform an ablation study to validate the usefulness of the different parts of the embedding, removing the structural part, the attribute part and the dimensions related to lifetime from the complete embedding. To perform this ablation study, we also consider the experiment described in Section 6.4.2, concerning the DARPA OpTC dataset.

6.4.4. RQ4: How complementary are the entity and the graph levels?. To highlight the complementarity of those two levels, we compare the performances of GRAAL-g and GRAAL for the graph detection, and GRAAL-e and GRAAL for the entity detection. To perform GRAAL's graph detection, we consider the experiment described in Section 6.4.1, that concerns the DARPA OpTC dataset. Thus we train 6 GRAAL-e models (one for each host), to perform the combined detection of GRAAL-g and GRAAL-e, resulting in GRAAL's detection, as explained in Section 5.4. To perform GRAAL's entity detection, we consider the experiment described in Section 6.4.2. Thus we train a GRAAL-g model on the 3 hosts, to combine GRAAL-g and GRAAL-e results.

6.4.5. RQ5: How effective is the transfer learning between levels?. We assess the transfer learning strategy presented in Section 5.3 by experimenting GRAAL-e detection with three different learning strategies: GRAAL-e from scratch (i.e., by initializing with random weights), fine-tuning (i.e., by initializing GRAAL-e training with GRAAL-g' weights) or transfer (i.e., by fixing GRAAL-e's weights with GRAAL-g's weights, only the last layers' weights are updated). We vary the number of hosts used to train GRAAL-g and the Word2Vec models: 3 hosts and all the hosts of the DARPA OpTC dataset (i.e., 475 hosts). The GRAAL-e results are for the same 3 hosts selected previously by Flash.

TABLE 4: Graph-level detection comparison (*thresholds are biased) (± correspond to the 95% confidence interval)

Dataset	IDS	Train-ratio	Precision	Recall	F1-score	AUC
Streamspot	GAE*	0.25	0.31	1.00	0.47±0.00	**0.40**±0.00
	Kairos	0.25	0.95	1.00	0.97±0.02	**0.99**±0.01
	GRAAL	0.25	0.93	0.95	0.94±0.03	**1.00**±0.01
	GAE*	0.80	0.63	1.00	0.77±0.00	**0.40**±0.00
	Flash*	0.80	1.00	0.95	0.97±0.00	**0.96**±0.00
	Magic*	0.80	1.00	1.00	1.00±0.00	**1.00**±0.00
	GRAAL	0.80	1.00	0.95	0.97±0.00	**1.00**±0.00
Wget	GAE*	0.80	0.50	1.00	0.67±0.00	**0.56**±0.03
	Flash*	0.80	0.28	0.36	0.32±0.35	**0.43**±0.31
	Magic*	0.80	0.94	0.94	0.94±0.01	**0.97**±0.02
	GRAAL	0.80	0.90	0.94	0.92±0.01	**0.96**±0.02
OpTC-6hosts	GAE*	×	0.11	0.85	0.19	**0.53**
	Kairos*	×	0.40	0.31	0.35	**0.63**
	Magic*	×	0.12	0.55	0.19	**0.55**
	GRAAL	×	0.98	0.40	0.57	**0.74**

6.4.6. RQ6: How scalable is GRAAL?. We are able to perform intrusion detection on the entire DARPA OpTC dataset, i.e., the activity of 500 hosts. Of these 500 hosts, 25 are inactive outside the attacks days, so their activities are not represented into the models' training set, thus we do not consider these 25 hosts in the experimentation. In addition, we perform GRAAL detection for multiple graphs' duration: 1, 5, 10, 15, 20 or 30 minutes. This detection implies the same 3 hosts previously selected by Flash. To add more insights about GRAAL we also compare the execution time and models' size between GRAAL and its main competitor at graph-level.

6.5. Experimental Results

In the following, we analyze experimental results and answer the research questions.

Hardware setup. GRAAL training and evaluation were performed on a server running Debian 11, with Intel Xeon E5-2650 v4 processor with 12 cores and 256 GiB of RAM.

6.5.1. RQ1: How do the approaches compare at the graph level?. All approaches (except for GAE) have similar results on the Streamspot dataset (see Table 4), with an AUC close to 1.00. As remarked by other authors, the detection task on this dataset is relatively easy. The performances of GAE are quite low, with an AUC of 0.40. This is expected as this model is not specifically tailored to detect intrusions. We notice that reducing the train ratio down to 0.25 does not have a significant impact on GRAAL's AUC.

The performances on the Wget dataset (see Table 4) are globally worse than on Streamspot. The performances of Flash are particularly variable and lower than the published ones, with a mean AUC of 0.43 and a 95% confidence interval of 0.31. As the authors did not perform a cross-validation, we consider the results presented in Table 4 to be a better estimation of its actual performances. Here again, the performances of GAE are quite

TABLE 5: Entity-level detection comparison on three hosts of the DARPA OpTC Dataset (*threshold is biased)

Host	IDS	TP	FP	TN	FN	Precision	ADP
201	**Orthrus**	1	0	18.6k	519	**1.00**	**0.99**
	Velox	2	7	18.6k	518	0.22	**0.39**
	Flash*	4	143	18.5k	516	0.03	**0.32**
	GRAAL	2	0	18.6k	518	**0.99**	**0.99**
501	Orthrus	0	1	20.7k	97	0.00	**0.06**
	Velox	0	1	20.7k	97	0.00	**0.02**
	Flash*	5	884	19.9k	92	0.01	**0.25**
	GRAAL	16	13	20.8k	81	**0.55**	**1.00**
051	Orthrus	0	3	16.1k	151	0.00	**0.25**
	Velox	0	0	16.1k	151	0.00	**0.12**
	Flash*	4	140	16.1k	147	0.03	**0.25**
	GRAAL	34	11	16.2k	117	**0.76**	**1.00**

TABLE 6: Results on DARPA OpTC for internal GRAAL comparison

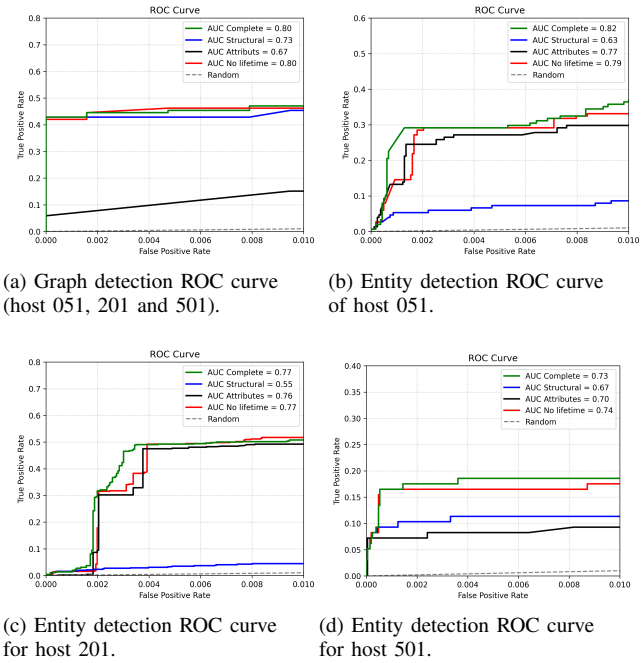
Detection	IDS	TP	FP	TN	FN	Precision	AUC
Graph 6hosts	OCSVM	66	25	1.3k	84	0.73	**0.81**
	GRAAL-g	61	2	1.4k	89	0.97	**0.74**
	GRAAL	60	**1**	1.4k	90	**0.98**	**0.74**
Process host 201	OCSVM	0	0	18.6k	520	0.00	**0.66**
	GRAAL-e	2	3	18.6k	518	0.40	**0.77**
	GRAAL	2	**0**	18.6k	518	**1.00**	**0.77**
Process host 501	OCSVM	9	11	20.7k	88	0.45	**0.61**
	GRAAL-e	16	16	20.7k	81	0.50	**0.73**
	GRAAL	16	13	20.8k	81	**0.55**	**0.73**
Process host 051	OCSVM	3	6	16.2k	148	0.33	**0.77**
	GRAAL-e	44	21	16.2k	107	0.68	**0.82**
	GRAAL	34	11	16.2k	117	**0.76**	**0.82**

low, with an AUC of 0.56. Magic's performances (0.97 AUC) are slightly better than GRAAL's (0.96), but this difference is not statistically significative, as shown by the 95% confidence interval.

To focus on the DARPA OpTC dataset (see Table 4), GRAAL significantly overtakes other state-of-the-art approaches in F1-score and AUC. More specifically (described in the appendix, Figure 9), GRAAL predicts 61 graphs as anomalous, with 60 TP and only 1 FP. Finally, GRAAL does not raise alarms on the benign host 207 and achieves to detect attacks at the beginning of their execution (described in the appendix, Figure 11). Kairos' results are worse than those presented by its authors, mainly due to the completeness of the test set (see Section 6.2).

6.5.2. RQ2: How do the approaches compare at the entity level?. GRAAL has better performances than Flash and Velox: the precision and ADP of GRAAL are far higher (see Table 5). Flash raises a very high number of FP, with only a few TP. This is largely due to the implementation changes, with the removal of the use of the ground truth (see Section 6.2). Orthrus and Velox's results are worse than those presented by their authors, mainly due to training instability (see Section 6.2). Whereas GRAAL's ADP are nearly always 1.00, the precision significantly depends on the evaluated host and thus on the performed attack. As shown in Table 5, the malicious activity of Host 501 seems to be the hardest one to detect (with a precision of 0.55), whereas the activity of Host 201 seems easier to detect (with a precision of 0.99). For this host 201, Orthrus' result seems competitive with GRAAL, both with regard to ADP and precision. However, the relevant part of the ROC curve (i.e., FP rate under 0.01, as explained in the appendix Section D.2) implies that GRAAL performs better than Orthrus (described in the appendix, Table 10 and Figure 12).

6.5.3. RQ3: How effective is the node embedding?. As presented in Table 6, a OCSVM that takes our embeddings (presented in Section 5.2) as inputs, performs graph outlier detection with a precision of 0.73. This is far higher than precisions of GAE, Kairos and Magic presented in Table 4, respectively 0.11, 0.40 and 0.12. Looking at a more holistic metric, the OCSVM's achieve a f1-score of 0.55 (described in the appendix, Figure 11), outperforming all the compared IDS. At entity-level, the OCSVM also



(a) Graph detection ROC curve (host 051, 201 and 501).

(b) Entity detection ROC curve of host 051.

(c) Entity detection ROC curve for host 201.

(d) Entity detection ROC curve for host 501.

Figure 6: GRAAL ROC curves limited to 0.01 false positive rate (FPR). Ablation study regarding the different part of the embedding.

performs significantly better than compared IDS, except for Host 201. Thus, our embedding alone is efficient enough to outperform results of nearly all the compared IDS. Nevertheless, the OCSVM's results are still not as good as those obtained with GRAAL, highlighting the importance of the GRAAL models.

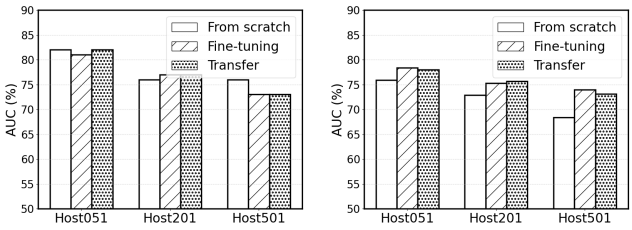
Figure 6 presents the relevant part of the ROC curve (i.e., FP rate under 0.01) for GRAAL detection, using truncated embeddings. For graph detection, the structural part of the embedding is more accurate than the attribute part. However, entity detection of Hosts 051 and 201 leads to the opposite conclusion: the attribute part is more accurate than the structural part of the embedding. Thus, the effectiveness of these two parts of embedding depends on the detection level. However, their combination (i.e., the complete embedding) leads to the best ROC curve with the best AUC for both detection level. The removing of the

encoding of the lifetime (see Section 5.2) leads to a small decrease of the performances of GRAAL, as shown on the Figure 6. The results do not highlight significantly the benefit of this lifetime, but we are convinced that lifetime may prove more relevant when evaluating GRAAL on other datasets with other attacks.

6.5.4. RQ4: How complementary are the entity and the graph levels?. Presented in Table 6, GRAAL achieves less false positive than its components themselves GRAAL-g and GRAAL-e. Indeed, GRAAL is composed of the combination of the detections of GRAAL-e and GRAAL-g using the GRAAL detection strategy, presented in Section 5.4. For the graph detection, the removed FP is a graph slightly before the attack of the third scenario on Host 051. A graph of Host 201, during the last evaluated day, is a TP for GRAAL-g, and a FN for GRAAL. Thus, GRAAL does not detect this graph anymore. According to the description document, Host 201 is involved only in the attack of the first day, and thus should not be labeled during the last day. At entity level, Table 6 highlights the benefit of using GRAAL over GRAAL-e. Indeed, for Hosts 201, 501 and 051, GRAAL removes respectively 3, 3 and 10 FP. Thus, for both detection level, GRAAL reduces the number of false positives returned by the individual GRAAL-e and GRAAL-g models. The AUC of GRAAL-e and GRAAL is essentially the same. The same applies to GRAAL-g and GRAAL. The reason is that the combination does not impact the reconstruction error values of a lot of processes. Indeed, only 0.02%, 0.03% and 0.10% of the reconstruction errors changed for respectively Hosts 201, 501 and 051. Thus, with less than 0.1% of changes by host, the AUC does not vary significantly. However, these few changes are relevant as explained previously: they reduce the number of FP. Moreover, this combination provides a more detailed explanation of the raised alerts. Indeed, GRAAL highlights suspicious graphs but also some suspicious entities in these graphs.

6.5.5. RQ5: How effective is the transfer learning between levels?. Figure 7 presents the impact of using transfer learning. With GRAAL-g learned on few hosts, the different learning strategies are equivalent. However, if the models are trained using more hosts as shown in Figure 7b, the "from scratch" learning strategy is always the least efficient. In average, the shared learning strategies "fine-tuning" and "transfer" yield better results than "from scratch" results. Thus, transferring knowledge from GRAAL-g to GRAAL-e is effective.

6.5.6. RQ6: How scalable is GRAAL?. Unlike other IDS, GRAAL is evaluated on the 475 DARPA OpTC hosts (including 26 malicious hosts). GRAAL's results are presented in Table 7. At graph level, GRAAL achieves a f1-score of 0.86, with only 7 FP. The 468 detected graphs take place in the activity of 26 hosts: 1 FP host (i.e., host 723) and 25 malicious hosts. Because of the combination results, at least 1 process has been detected per detected graph. Thus, with only 65 detected processes and 468 detected graphs, these processes raise alerts in multiple graphs. Moreover, only 3 FP processes are related to the FP graph of the FP host. Although GRAAL triggers few alerts at the entity level, the processes detected are



(a) GRAAL-g and Word2Vec trained on 3 hosts.

(b) GRAAL-g and Word2Vec trained on 475 hosts.

Figure 7: Comparison of learning strategy, to perform entity detection with GRAAL-e.

TABLE 7: Results on the complete DARPA OpTC dataset, with 475 hosts

Detection	IDS	TP	FP	TN	FN	Precision	F1-score
Graph	GRAAL	461	7	118.5k	138	0.99	0.86
Process	GRAAL	37	28	6,9M	1.5k	0.57	0.05

key processes in attacks. Indeed, these detected processes make it possible to trace a large part of the attackers' activity, as shown by the f1-score of 0.86 obtained by the GRAAL graph-level detection. Thus, in addition to achieve practical and realistic results for the response team, GRAAL manages to perform intrusion detection on a large amount of activity (the whole DARPA OpTC dataset), thanks to the simplicity of its design.

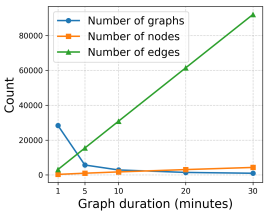
Indeed, all GRAAL experiments presented previously use the same number of layers and the same learning rate for all datasets. Thus GRAAL does not need to be highly optimized to be effective. We compare the execution time to create all the artifacts (i.e., graphs, models, encoding, etc.) and train the models of GRAAL and Kairos on this dataset. While Kairos needs at least 30 hours with a GPU to be deployed, GRAAL needs less than 7 hours without a GPU. Moreover, Kairos models' size is 546MB, whereas GRAAL models' size is less than 3MB, due to a small number of layers in its autoencoders.

We also performed an experiment with different time window durations. The results are presented in Figure 8. Longer time windows lead to fewer and larger graphs. The shorter the duration of the graphs, the longer the execution time, both for training and inference, except after ten minutes, where there is no remarkable effect. The execution time of feature extraction is influenced by a short graph duration, mainly due to the large number of graphs. However, the execution time of the training does not exceed a total of 3 hours. RAM requirements remain stable. No significant impact on the detection performances is observed.

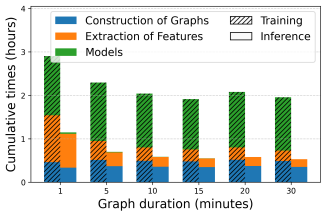
These experiments let us conclude that GRAAL's training and inference times correctly scale with the time windows duration and that GRAAL is indeed scalable.

7. Analysis of Alerts and Discussion

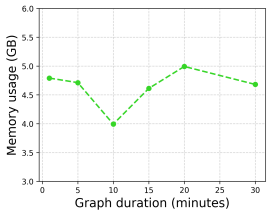
Examination of GRAAL's detected graphs and processes reveals insights that point to potential improve-



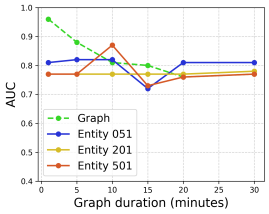
(a) Headcount based on the duration of the graphs.



(b) Stage-wise processing time breakdown.



(c) Resources utilization.



(d) GRAAL performances.

Figure 8: Statistics of GRAAL regarding the duration of the graphs. The train set contains 4 benign days. The test set contains 3 days with benign and malicious activity. Statistics focus on 3 hosts: 051, 201, 501.

ments. We focus the analysis on the DARPA OpTC dataset, as this dataset is more realistic than the two others.

At graph level GRAAL raises 1 false positive (FP) (in the appendix, Figure 11). This graph represents the activity of Host 501 during the attack scenario 2. Thus, even if this graph is not labelled as malicious by Nikulshin and Tahli's [45], this graph is located during the attack targeting this host. Focusing on false negatives (FN), 90 graphs labeled as malicious are missed by GRAAL. GRAAL does not detect any graphs during the night of the attack scenario 2, where the only information given by the description document is that the malware ran overnight. The malware performs no malicious actions other than nighttime beaconing that may be difficult to detect. GRAAL also does not detect the very beginning and the middle of the attack scenario 3 on Host 051, where attackers perform a lateral movement on another host. GRAAL does not also detect the end of the attack scenario 1 (for Hosts 201 and 402). Even if Nikulshin and Tahli's [45] labels the night of the first attack's day as part of the scenario 1, the dataset description document claims this attack ends in the middle of the afternoon. The duration of this first attack can therefore be interpreted in different ways. However, the location of the start of this attack is consistent between the document and the proposed labels, and GRAAL detects it.

At entity level, GRAAL detects only a few processes: 2, 29 and 45 for respectively Hosts 201, 501 and 051 (Figure 5). For Host 201, these 2 true positive (TP) processes enable the entire attack to be tracked. Indeed, the first process detected is the first attack agent, and the second process detected is the second attack agent. The first process is a base64-encoded PowerShell command that tries to access the Group Policy cached settings and is part of the attacker's reconnaissance phase. All the activity of this first detected process takes place in the graph where it is detected. GRAAL therefore detects the attack in its initial phase. Regarding the second detected process, all of it activity is also detected, but this time across 10 different graphs. This process initiates network discovery (approximately 250 pings on a given subnet). Moreover, the children of these 2 detected processes represent 70% of the malicious processes of this host. Thus, even if GRAAL does not detect a large amount of processes, the detected processes are valuable to analyze attacks. To focus on FP, GRAAL raised 13 FP for Host 501 and 11 FP for Host 051, for various reasons. Due to time windows, the activity of a process can be split into several different graphs. Thus, some FP processes are detected only at the end of their activity. One solution could be to implement sliding time windows, which would eliminate interruptions in process activity. Some few FP are due to errors in the dataset. Indeed, Anjum et al. [46] note few issues into the dataset, due to conflicted sources. Others FP are processes that perform a lot of operations on registers or files, such as a process RuntimeBroker.exe that launches verclsid.exe, a Windows system tool used to manage Component Object Model (COM) components. A whitelist could be a solution to prevent these FP. Another solution could be a specific re-training for these FP processes. For these 2 hosts, GRAAL detects 50 TP. Among these TP, there are powerShells that execute attack agents, processes related to malicious Remote Desktop Protocol (RDP) sessions, powershells responsible for data exfiltration, etc. GRAAL enables this analysis by providing experts with graph visualizations that include all nodes and attributes, such as command lines and image paths, and a timelapse of the detected graphs for a higher-level representation.

8. Conclusion

In this article, we introduce the GRAAL framework, an end-to-end unsupervised graph-based anomaly-based IDS that performs multi-level detection of system activity. The novelty of GRAAL lies in its feature extraction and its use of two detection levels that closely interact. GRAAL employs similar autoencoders trained through a coupled learning phase, along with a combination of inference results. This approach enables the creation of compact models that are trained quickly while achieving better than state-of-the-art performances in graph-based intrusion detection. Moreover, we successfully evaluate the entirety of a large dataset at both graph and entity levels, demonstrating the scalability of GRAAL.

We also identify several methodological and reproducible biases in the current literature on graph-based provenance-based anomaly-based detection. We experimentally assess the effect of several biases and aim at proposing to the community available implementation of the mitigated biases of the current IDS, in addition with the available implementation of GRAAL.

Acknowledgments

This project is funded by CREACH Labs (Brittany region of France, DGA - Direction Générale de l'Armement). Experiments were carried out using the Grid'5000 testbed.

References

[1] T. Sowmya and E. Mary Anita, "A comprehensive review of AI based intrusion detection system," Measurement: Sensors, vol. 28, p. 100827, 2023.

[2] N. Mohamed, "Artificial intelligence in cybersecurity: A review of solutions for APT-exploited vulnerabilities," in 15th International Conference on Computing Communication and Networking Technologies (ICCCNT'24), 2024, pp. 1–7.

[3] H. ur Rehman, Z. Jalil, and S. Fahim, "AI-Driven APT Detection Framework: Early Threat Identification Using ML," in 21st International Bhurban Conference on Applied Sciences and Technology (IBCAST'24), 2024, pp. 84–91.

[4] R. Buchta, G. Gkoktsis, F. Heine, and C. Kleiner, "Advanced Persistent Threat Attack Detection Systems: A Review of Approaches, Challenges, and Trends," vol. 5, no. 4, 2024.

[5] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "Kairos: Practical intrusion detection and investigation using whole-system provenance," in 2024 IEEE Symposium on Security and Privacy (SP). IEEE, 2024, pp. 3533–3551.

[6] 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 Computer Society, 2024, pp. 139–139.

[7] 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.

[8] S. T. King and P. M. Chen, "Backtracking intrusions," in Proceedings of the nineteenth ACM symposium on Operating systems principles, 2003, pp. 223–236.

[9] S. Tariq, M. Baruwal Chhetri, S. Nepal, and C. Paris, "Alert fatigue in security operations centres: Research challenges and opportunities," ACM Computing Surveys, vol. 57, no. 9, pp. 1–38, 2025.

[10] R. Paudel and H. H. Huang, "Pikachu: Temporal walk based dynamic graph embedding for network anomaly detection," in Network Operations and Management Symposium, 2022, pp. 1–7.

[11] K. Liu, Y. Dou, Y. Zhao, X. Ding, X. Hu, R. Zhang, K. Ding, C. Chen, H. Peng, K. Shu, L. Sun, J. Li, G. H. Chen, Z. Jia, and P. S. Yu, "Bond: Benchmarking unsupervised outlier node detection on static attributed graphs," in Advances in Neural Information Processing Systems, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 27 021–27 035.

[12] B. Jiang, T. Bilot, N. El Madhoun, K. Al Agha, A. Zouaoui, S. Iqbal, X. Han, and T. Pasquier, "Orthrus: Achieving high quality of attribution in provenance-based intrusion detection systems," in Security Symposium (USENIX Sec'25). USENIX, 2025.

[13] T. Bilot, B. Jiang, Z. Li, N. El Madhoun, K. Al Agha, A. Zouaoui, and T. Pasquier, "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.

[14] Venkatakrishnan, Venkat and Momeni, Sadegh and team at the University of Illinois - Chicago., "Streamspot data," 2024. [Online]. Available: https://github.com/sbustreamspot/sbustreamspot-data

[15] X. Han, "Wget Dataset," 2018. [Online]. Available: https://doi.org/10.7910/DVN/IA8UOS

[16] FiveDirections, "DARPA OpTC Data," 2020. [Online]. Available: https://github.com/FiveDirections/OpTC-data

[17] MITRE Corporation, "MITRE cyber analytics repository," 2024. [Online]. Available: https://car.mitre.org/data_model/

[18] X. Han, T. Pasquier, and M. Seltzer, "Provenance-based intrusion detection: opportunities and challenges," in 10th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2018), 2018.

[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. 102282, 2021.

[20] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," CoRR, vol. abs/2003.05991, 2020. [Online]. Available: https://arxiv.org/abs/2003.05991

[21] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," science, vol. 313, no. 5786, pp. 504–507, 2006.

[22] M. Catillo, A. Pecchia, and U. Villano, "Successful intrusion detection with a single deep autoencoder: theory and practice," Software Quality Journal, vol. 32, no. 1, pp. 95–123, 2024.

[23] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," arXiv preprint arXiv:1802.09089, 2018.

[24] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," arXiv preprint arXiv:1901.03407, 2019.

[25] T. N. Kipf and M. Welling, "Variational graph auto-encoders," NIPS Workshop on Bayesian Deep Learning, 2016.

[26] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in Proceedings of the 2019 ACM SIGSAC conference on computer and communications security, 2019, pp. 1795–1812.

[27] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in 2019 IEEE symposium on security and privacy (SP). IEEE, 2019, pp. 1137–1152.

[28] T. Chen, C. Dong, M. Lv, Q. Song, H. Liu, T. Zhu, K. Xu, L. Chen, S. Ji, and Y. Fan, "Apt-kgl: An intelligent apt detection system based on threat knowledge and heterogeneous provenance graph learning," IEEE Transactions on Dependable and Secure Computing, 2022.

[29] M. M. Anjum, S. Iqbal, and B. Hamelin, "Anubis: a provenance graph-based framework for advanced persistent threat detection," in Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, 2022, pp. 1684–1693.

[30] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "Atlas: A sequence-based learning approach for attack investigation," in 30th USENIX security symposium (USENIX security 21), 2021, pp. 3005–3022.

[31] K. Liu, Y. Dou, X. Ding, X. Hu, R. Zhang, H. Peng, L. Sun, and P. S. Yu, "PyGOD: A Python library for graph outlier detection," Journal of Machine Learning Research, vol. 25, no. 141, pp. 1–9, 2024. [Online]. Available: http://jmlr.org/papers/v25/23-0963.html

[32] 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.

[33] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," 2020.

[34] S. Wang, Z. Wang, T. Zhou, H. Sun, X. Yin, D. Han, H. Zhang, X. Shi, and J. Yang, "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.

[35] 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.

[36] 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.

[37] J. Zengy, X. Wang, J. Liu, Y. Chen, Z. Liang, T.-S. Chua, and Z. L. Chua, "Shadewatcher: Recommendation-guided cyber threat analysis using system audit records," in 2022 IEEE Symposium on Security and Privacy (SP). IEEE, 2022, pp. 489–506.

[38] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio et al., "Graph attention networks," stat, vol. 1050, no. 20, pp. 10–48 550, 2017.

[39] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," arXiv preprint arXiv:1301.3781, 2013.

[40] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 3971–3988.

[41] G. Apruzzese, P. Laskov, and J. Schneider, "Sok: Pragmatic assessment of machine learning for network intrusion detection," in 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P). IEEE, 2023, pp. 592–614.

[42] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. Van Steen, "Prudent practices for designing malware experiments: Status quo and outlook," in 2012 IEEE symposium on security and privacy. IEEE, 2012, pp. 65–79.

[43] T. Abrar, A. Shamail, M. J. Iqbal, A. Ahmed, M. Abdullah, M. Shayan, F. Zaffar, T. Pasquier, D. Eyers, and A. Gehani, "On the Reproducibility of Provenance-based Intrusion Detection that uses Deep Learning," in Conference on Reproducibility and Replicability (REP'25). ACM, 2025.

[44] K. Berahmand, F. Daneshfar, E. S. Salehi, Y. Li, and Y. Xu, "Autoencoders and their applications in machine learning: a survey," Artificial intelligence review, vol. 57, no. 2, p. 28, 2024.

[45] V. Nikulshin and C. Talhi, "Effective IDS under constraints of modern enterprise networks: revisiting the OpTC dataset," in 2024 7th Conference on Cloud and Internet of Things (CIoT). IEEE, 2024, pp. 1–8.

[46] M. M. Anjum, S. Iqbal, and B. Hamelin, "Analyzing the usefulness of the darpa optc dataset in cyber threat detection research," in Proceedings of the 26th ACM symposium on access control models and technologies, 2021, pp. 27–32.

Appendix A. System Logs

Figure 9 presentes an eCAR system event from the DARPA OpTC dataset. The process *5689402e-9c41-4fe1-aff4-a0b43a8417cf* terminates the process *5689402e-9c41-4fe1-aff4-a0b43a8417cf* (i.e., itself) at timestamp *2019-09-21T03:03:07.132-04:00*. This event occurs on the *063* host, with the user *agilbard*. This eCAR system event stresses multiple detailed information, such as the command line and the sid. However the minimal information contained in a system event are the *action*, the *actorID* with the actor's type (in eCAR log actors are always processes), the *objectID* with the object's type (i.e., *object*), and the *timestamp*.

Appendix B. Biases

B.1. Biases review

Table 8 groups the 50 biases defined in prior-works [13], [40]–[43].

The filter F1 considers biases that are mainly specific to the context of malware, and are therefore out of the scope of IDS. Thus, 13 out of 19 biases of Rossow et al. [42] are filtered out. We also do not consider "Closed and Open world" for Apruzzese et al. [41] because this bias is related to classification problems, not for unsupervised anomaly detection.

The filter F2.a focuses on system design and architecture. This filter removes notably several biases from Bilot et al. [13]. For example, the "Insufficient Detection Granularity" bias does not prevent a fair comparison, provided that all the IDS being compared share the same detection granularity. "Lacking Real-time Detection" also does not rule out a comparison because all the selected IDS generate graphs prior to inference. Finally, "Overly Complex Architectures" and "Spurious Correlations" from Arp et al. [40], are about IDS's internal experiments. Thus, these biases are unrelated to the experimental comparison protocol.

The filter F2.b rules out biases related to dataset responsibility. Indeed, dataset's authors are responsible for the data distribution and the labels' accuracy. The "Mention of the system used during execution" also refers to the dataset documented generation. Thus, these biases are unrelated to the experimental comparison protocol, provided that the compared IDS are evaluated on the same dataset, with the same labels.

The filter F2.c highlights biases that are only discussion and not related to experiments protocol. Indeed, the "Analyze of the reasons for false positives and false negatives" does not impact the experimental comparison protocol. Thus, with the addition of this filter, all biases from Rossow et al. [42] are filtered out.

The filter F2.d focuses on the attack modeling. We restrict our comparison to the level of attack sophistication captured in the evaluated datasets. (F2.d). "Adaptive adversaries", which relate to IDS robustness against adversarial attacks, are therefore out of scope.

The filter F3 stresses reproducibility biases used to select compared IDS. Indeed, IDS must have an end-to-end execution with available scripts (F3). We do not consider IDS that do not pass these biases, because we cannot determine from their implementations whether some biases are still present.

The filter F4 considers biases that are under the responsibility of the party making the experimental comparison protocol. These biases are reassessed during a new comparison campaign. "Inappropriate Baseline", "Unfair Comparison with Baselines" (well-tuned vs untuned) and "Not measuring Instability" imply that the experimental comparison protocol must already ensure that the performance of the IDS can be compared. Indeed, "Measuring instability" is only possible if a single iteration of the IDS already yields unbiased results. The "Baseline" is selected by the party making the comparison, thus this is not a bias linked to an IDS. As a party conducting a new comparison campaign, we mitigate these biases during our experimental comparison, by selecting appropriate baselines and measuring the instability of IDS when evaluated on small datasets.

The filter F5 is the result of an analysis of the IDS implementations. All selected IDS consider common metrics that are widely used into the community. Thus, we do not consider the two duplicate biases related to missing metrics.

We consolidate the remaining twelve biases by removing duplicates and refining some of them to more precisely capture findings observed in IDS implementations, resulting in seven final biases. All the remaining biases from Abrar et al. [43] are into the category **B**, that encompasses those that hinder reproducibility. The remaining bias from Apruzzese et al. [41] is a subset of the bias **A1**. The remaining biases from Arp et al. [40] and

Bilot et al. [13] contain duplicated biases: biases related to data snooping (**A2**), biases related to parameter selection on test data (**A5**) and biases related to the large number of data to evaluate (**A3**). Moreover, we refine the bias "Data snooping" as two separated biases, distinguishing the use of the test data during the training phase (**A2**), and the use of the ground truth information during the detection phase (**A4**).

B.2. Biases found in the IDS implementations

We focus the comparison on six detectors. The details of each approach are available in Section 3. The current section exposes additional low-level details regarding biases presented in Section 4 and Section 6.2.

Flash [6]. Flash reduces the test set to the compromised days of three hosts of the DARPA OpTC dataset (see bias A3). In the function `prepare_test_set`, only the following files are downloaded:

```
log_files = [
    ("AIA-201-225.ecar-2019-12-08T11
        -05-10.046.json.gz", "0201"),
    ("AIA-201-225.ecar-last.json.gz", "0201"
        ),
    ("AIA-501-525.ecar-2019-11-17T04
        -01-58.625.json.gz", "0501"),
    ("AIA-501-525.ecar-last.json.gz", "0501"
        ),
    ("AIA-51-75.ecar-last.json.gz", "0051")
]
```

When we checked the files names with the official dataset of DARPA OpTC (https://drive.google.com/drive/folders/1ZfTifLohtEclehIrl21ZDtf8xPL05FsJ), we found that these files correspond to the attack days for the hosts: host 201 is evaluated on Sept 23th, host 501 on Sept 24th and host 051 on Sept 25th. In this work, we perform detection on these three hosts during all the testing days, including days without attacks. Moreover, we notice that Flash uses the ground truth to return more relevant results. In https://github.com/DART-Laboratory/Flash-IDS/blob/ccbafee7eb796c76aceaabdc259ebd1f12b867cc/OpTC.ipynb, true positives (TP) are defined as:

```
TP = MP.intersection(GP)
...
two_hop_tp = Get_Adjacent(TP, mapp, edges, 2)
...
TPL = TP.union(FN.intersection(two_hop_tp))
```

The list of true positives (TPL) is obtained after a set operation that relies on the adjacent nodes of TP (`two_hop_tp`), that itself depends on the ground truth positive (GP). This use of the ground truth is not detailed in the experimental protocol of Flash. Thus, Flash defines their true positive as the two-hop entities of the true positive nodes detected by their model. However, nodes detected by the model are supposed to be identified as true positives or false positives only to compute metrics (see bias A4). Thus, we remove the use of the ground truth, and the alerts are only the nodes detected without the extension to the two-hop entities. Their confidence threshold and similarity threshold are kept as hard-coded values proposed into the implementation (see bias A5). In the same file, the confidence threshold is hard-coded, such as:

```
identified_ids,edges,mapp = evaluate_model(df,
    xgboost_model, 1, 0.6)
```

(the confidence threshold is the last parameter). The authors do not provide any source code that had been used to obtain these thresholds and the article is vague about this procedure, so it is not reproducible in its current state.

Kairos [5]. Training data is not clearly defined as mentioned in the issue https://github.com/ubc-provenance/kairos/issues/19 (see bias A1). Kairos evaluates its performance on the six hosts only on the days they are compromised (see bias A3). In https://github.com/ubc-provenance/kairos/blob/0e0b633beb46a1117c0a6d63be5d2481b59ac0dc/DARPA/OpTC/optc_graph_learning.ipynb, some hosts are only evaluated on a subset of the test set. In the section called "Anoamly detection" (sic), h201 is evaluated on Sept 23rd, 24th and 25th, while h402 is only evaluated on Sept 23rd. Besides, the predictions are initialized as benign (label 0):

```
for f in filelist:
    pred_label_h402[test_path+f]=0
```

But the code that updates the predictions is not present for Sept 24th and Sept 25th.

```
for i in label_h402:
    y.append(label_h402[i])
    y_pred.append(pred_label_h402[i])
```

Indeed, `label_h402` contains the label for Sept 23rd, 24th and 25th. To alleviate this bias, we evaluate Kairos on the three days of the testing set for all hosts. In its implementation, Kairos uses hard-coded thresholds, sometimes different between attacks and non-attack days (see bias A5). For example, for host 201, the threshold is 1000 on Sept 23rd (the attack day) and $10,000$ for Sept 24th and Septh 25th (days without attacks). In this work we set a unique threshold for all hosts, but this threshold maximizes the F1-score and is thus favorable to Kairos. Finally, as mentioned in the issue https://github.com/ubc-provenance/kairos/issues/19, the labels used by Kairos on DARPA OpTC are not available (see bias B6). Thus, we perform detection with the Nikulshin and Tahli's [45] labels of the DARPA OpTC dataset (see Section 6.1).

Magic [7]. Regarding the graph-level detection, we notice that the split between the training and testing sets is not clear for both datasets (see bias A1 and A2). Indeed, in https://github.com/FDUDSDE/MAGIC/blob/aa0b647eea74b6faa0e52eb444370c4411a32cbe/utils/loaddata.py#L113, the train set is selected with:

```
train_dataset = [i for i in range(len(dataset))
    if dataset[i][1] == 0]
```

where `dataset[i][1]` contains the label of a graph. So, this function collects all benign graphs for training, including the ones used later for testing. Thus, we slightly change the implementation to make a clear split, using same sets defined for others compared IDS. Regarding the threshold, Magic uses a threshold that maximizes the F1-score (see bias A5). In https://github.com/FDUDSDE/MAGIC/blob/aa0b647eea74b6faa0e52eb444370c4411a32cbe/model/eval.py#L79, the authors use:

```
prec, rec, threshold = precision_recall_curve(
    y_test, score)
f1 = 2 * prec * rec / (rec + prec + 1e-9)
max_f1_idx = np.argmax(f1)
best_thres = threshold[max_f1_idx]
```

to identify the threshold that maximizes the F1-score on test data. We keep this definition of the threshold.

Implementations of **Orthrus** [12], **Velox** [13] remain unchanged, as their biases A3 and B7 do not impact the implementation of these IDS. Indeed, only events of specific hosts (201, 501 or 051) are collected to form their considered dataset (cf. A3), and we could not retrieve original results (cf. B7).

Appendix C. Lifetime Encoding

Figure 10 presents different cases for the lifetime encoding of a process P. As explained in Section 5.2, the lifetime encoding of a process P, in the graph G of duration T, depends on the events related to P that occur during the time interval T.

Appendix D. Detailed Results

D.1. Graph Detection

Table 9 presents detailed results for graph detection. This table highlights true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN), which are not available in the Table 4. Because of the five-fold cross-validation, results on the Wget and Streamspot contain floating numbers (e.g., 222.58 FP).

Figure 11 presents detailed time-lapse of graphs detection of GRAAL on the DARPA OpTC dataset. This visualization highlights that GRAAL achieves to detect attacks at the beginning of their execution, except for host 051 (missing the first two graphs).

D.2. Entity Detection

Table 10 presents detailed results for entity detection. This table highlights recall, f1-score and AUC, which are not available in the Table 5. GRAAL's AUC and ADP are computed as explained in the appendix Section D.3.

To compare more precisely IDS for entity detection, Figure 12 focuses on the relevant part of the ROC curve: the beginning of the curve, with a false positive rate (FPR) under 0.01. Indeed, a FPR of 0.01 implies around 180, 200 and 160 false positives, for respectively host 201, 501 and 051. These large numbers of false positives are unrealistic for a response team. Thus, we focus on the curve where the FPR is under 0.01. For each host, GRAAL's curve is always higher than other curves. Moreover, the entity with GRAAL's maximal error reconstruction is always part of an attack: this explains the ADP of 1.00 and 0.99.

D.3. GRAAL Internal Detection

Table 11 presents detailed results for the internal comparison of GRAAL: OCSVM using the embeddings of GRAAL, GRAAL-g, GRAAL-e and GRAAL. This table highlights recall, f1-score and ADP, which are not available in the Table 6.

Figure 13 focuses on the variation of the number of layers for GRAAL-g and GRAAL-e. To perform this experiment, we consider the experiment described in Section 6.4.2, that concerns the DARPA OpTC dataset. GRAAL-g's number of layers varies: two, four or six layers. GRAAL-e's additional number of layers also varies: one, two or three additional layers. The ROC curves at graph level and at entity level for host 051 and 501 highlight that these variations do not have a significant impact on GRAAL's performances. The ROC curve for host 201 appears to be more affected. However, this assessment can be nuanced by taking a closer look at GRAAL's performance in terms of true positives (TP) and false positives (FP). In fact, regardless of the number of layers selected, GRAAL's performance remains consistent: the same 2 TP are raised and there is no FP.

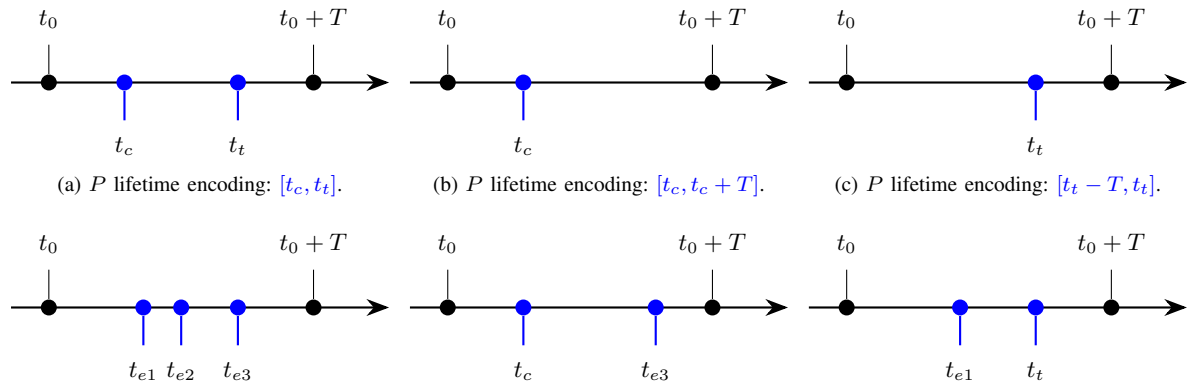
GRAAL's detection performances result from a combination that implies two different thresholds: a threshold for GRAAL-g and a threshold for GRAAL-e. To compute threshold-independent metrics for GRAAL, we focus on the variation of one threshold. For graph detection AUC metric, we vary the threshold of GRAAL-g. For entity detection AUC and ADP metrics, we vary the threshold of GRAAL-e. Moreover, the combination to achieve GRAAL detection has an impact on the reconstruction errors. Indeed, when a graph or an entity's suspicion is revised (as explain in Section 5.4), we consider that the reconstruction error of this revised element (i.e., graphs or entity) is reduced to be equal to the corresponding threshold (i.e., the graph threshold of GRAAL-g or the entity threshold of GRAAL-e). That way, an alarm is no longer raised, but these reconstruction errors remain high.

TABLE 8: The 50 biases defined by prior works. Biases are classified according to the associated filter (F1, F2.a, F2.b, F2.c, F2.d, F3, F4 and F5) or to to the seven final biases **A1**, **A2**, **A3**, **A4**, **A5**, **B6** and **B7**

Source	Bias	Description	Class
Arp et al. [40]	Sampling bias	Data does not sufficiently represent the true data distribution of the security problem.	F2.b
	Label inaccuracy	The ground-truth labels are inaccurate, unstable, or erroneous.	F2.b
	Data snooping	A learning model is trained with data that is typically not available in practice.	**A2, A4**
	Spurious correlations	Artificats unrelated to the security problem create shortcut patterns for separating classes.	F2.a
	Biased parameter selection	The final parameters of a learning-based method are not entirely fixed at training time. Instead, they indirectly depend on the test set.	**A5**
	Inappropriate Baseline	The evaluation is conducted without, or with limited, baseline methods.	F4
	Inappropriate Performance Measures	The chosen performance measures do not account for the constraints of the application scenario, such as imbalanced data or the need to keep a low false-positive rate.	F5
	Base rate fallacy	A large class imbalance is ignored, leading to an overestimation of performance.	**A3**
	Lab-Only Evaluation	System is solely evaluated in a lab setting, without discussing its practical limitations.	F2.c
	Inappropriate Threat Model	The security of machine learning is not considered.	F2.d
Apruzzese et al. [41]	Closed and open world	ML methods should be assessed also in "open world" scenarios.	F1
	Static and temporal data dependency	The iid principle (independent and identically distributed random variable) does not always hold in network environments.	**A1**
	Naive and adaptive adversaries	Security systems must always assume the presence of adversaries.	F2.d
Rossow et al. [42]	Check if goodware sample should be removed from datasets	Malware execution systems open to public sample submission lack control over whether specimens submitted to the system in fact consist of malware.	F1
	Balance datasets over malware families	Aggressively polymorphic malware families will often unduly dominate datasets.	F1
	Check whether training and evaluation dataset should have distinct families	Appearing in both may prove desirable for experiments that drive generic detection models for malware families by training on sample datasets.	F1
	Perform analysis with higher privileges than the malware's	Ideally, sensors are placed at a level where they cannot be modified.	F1
	Discuss and if necessary mitigate analysis artifacts and biases	Papers should explain the particular facets of the execution traces that a given model leverages.	F2.c
	Blending malware activity traces into benign background activity	Malware samples executing in dynamic analysis environments differs.	F1
	State family names of employed malware samples	Labeling the employed malware families helps the reader identify for which malware a methodology works.	F1
	List which malware was analyzed	The reader requires a summary that fully describes the malware samples.	F1
	Explain malware selection	Authors should describe how they selected the malware, and discuss any bias this induces.	F1
	Mention the system used during execution	Authors should also include version information of installed software.	F2.b
	Describe the network connectivity of the analysis environment	Malware families assign different roles of activity depending on a system's connectivity.	F1
	Analyze the reasons for false positives and false negatives	We advocate thoughtful exploration of what led to observed errors.	F2.c
	Analyze the nature/diversity of true positive	Papers should evaluate the diversity manifest in correct detections.	F2.c
	Evaluate relevant malware families	Using huge numbers of popular malware families bolsters the impact of experiments.	F1
	Perform real-world evaluations	Real-world experiment are an evaluation scenario that incorporated the behavior of a significant number of hosts in active use by people other than the authors.	F2.b
	Exercise caution generalizing from a single OS version	Papers should consider to what degree we can generalize results based on one specific OS version.	F2.c
	Choose appropriate malware stimuli	Authors should describe why the analysis duration they choose suffices for experiments.	F1
	Consider allowing Internet access to malware	In exceptional cases where experiments in simulated Internet environments are appropriate, authors need to describe the resulting limitations.	F1
	Describe containment policies	Authors should discuss the containment policies and their implications.	F1
Abrar et al. [43]	End-to-end Execution	All part of the system should be included, can be built, and run successfully.	F3
	Exact Result Reproducibility	The reproduced outcomes should match the figures presented in the paper.	**B7**
	Minor Result Difference	Relaxes the criteria for exact results reproducibility by allowing small deviations.	**B7**
	Data Availability	Records the number of datasets used in the paper that are publicly accessible.	**B6**
	Label Availability	For each publicly available dataset, the corresponding ground truth labels are publicly accessible.	**B6**
	Ready-to-run Sscripts	Evaluates how many dataset evaluations can be executed with minimal effort.	F3
	Trained model availability	Researchers may release trained models for the dataset used in their evaluations.	**B7**
	CPU/Memory discussion	Evaluates whether the paper discusses memory consumption and CPU usage.	F2.c
	Training/Testing time reporting	Assesses whether the paper reports model training and testing times.	F2.c
Bilot et al. [13]	Insufficient detection granularity	Most systems often overwhelm analysts with thousands of elements to review	F2.a
	Missing metric to measure attack detection	Most evaluations use post-threshold metrics, which overlook predictive power, misattribute poor performance, and are unsuitable for multi-attack detection.	F5
	Impractical thresholding methods	Some rely on arbitrary thresholds, while others use clustering techniques.	**A5**
	Unfair comparison with baselines	This practice leads to unfair comparison between well-tuned and untuned systems.	F4
	Not measuring instability	This phenomenon, driven by random factors, is largely overlooked in existing studies.	F4
	Featurization methods trained on test data	Some systems use text embeddings that cannot handle unseen words during inference. Mitigation involves sometime incorporating test data, leading to data snooping.	**A2, A5**
	Overly complex architectures	Authors should perform ablation studies, to validate complex architectures.	F2.a
	Insufficient scalability	Some systems have limited practicality in real-world settings.	**A3**
	Lacking real-time detection	Current systems necessitate completing graph construction prior to inference.	F2.a

```
{
    "action":"TERMINATE",\\
    "actorID":"5689402e-9c41-4fe1-aff4-a0b43a8417cf",\\
    "hostname":"SysClient0063.systemia.com",\\
    "id":"15aff6c7-a663-4244-a97a-258fc66625ed",
    "object":"PROCESS",
    "objectID":"5689402e-9c41-4fe1-aff4-a0b43a8417cf",
    "pid":4772,
    "ppid":776,
    "principal":"SYSTEMIACOM\\agilbard",
    "properties":{
        "acuity_level":"4",
        "command_line":"\\??\\C:\\Windows\\system32\\conhost.exe 0xffffffff -ForceV1",
        "image_path":"\\Device\\HarddiskVolume1\\Windows\\system32\\conhost.exe",
        "parent_image_path":"\\Device\\HarddiskVolume1\\Windows\\system32\\conhost.exe",
        "sid":"S-1-5-18",
        "user":"NT AUTHORITY\\SYSTEM"
    },
    "tid":-1,
    "timestamp":"2019-09-21T03:03:07.132-04:00"
}
```


Figure 9: Example of an eCAR system event.



(a) P lifetime encoding: $[t_c, t_t]$. (b) P lifetime encoding: $[t_c, t_c + T]$. (c) P lifetime encoding: $[t_t - T, t_t]$.

(d) P lifetime encoding: $[t_{e1} - T, t_{e3} + T]$. (e) P lifetime encoding: $[t_c, t_{e3} + T]$. (f) P lifetime encoding: $[t_{e1} - T, t_t]$.

Figure 10: Different cases of the lifetime encoding of the process P, on the graph G of duration T. The graph contains system activity between time t_0 and time $t_0 + T$. Time t_c if the creation of the process P. Time t_t is the termination of the process P. Times t_{e1}, t_{e2} and t_{e3} are events involving the process P but not the creation nor the termination.

TABLE 9: Detailed graph-level detection comparison (*thresholds are biased)

Dataset	IDS	Train-ratio	TP	FP	TN	FN	Precision	Recall	F1-score	AUC
Streamspot	GAE*	0.25	100	222.58	152.42	0	0.31	1.00	0.47±0.00	0.40±0.00
	Kairos	0.25	100	5.26	369.74	0	0.95	1.00	0.97±0.02	0.99±0.01
	GRAAL	0.25	95	7.15	367.85	5	0.93	0.95	0.94±0.03	**1.00±0.01**
	GAE*	0.80	100	58.73	41.27	0	0.63	1.00	0.77±0.00	0.40±0.00
	Flash*	0.80	95	0	100	5	1.00	0.95	0.97±0.00	0.96±0.00
	Magic*	0.80	100	0	100	0	1.00	1.00	1.00±0.00	**1.00±0.00**
	GRAAL	0.80	95	0	100	5	1.00	0.95	0.97±0.00	**1.00±0.00**
Wget	GAE*	0.80	25	25	0	0	0.50	1.00	0.67±0.00	0.56±0.03
	Flash*	0.80	9	23.14	1.86	16	0.28	0.36	0.32±0.35	0.43±0.31
	Magic*	0.80	23.5	1.5	23.5	1.5	0.94	0.94	0.94±0.01	**0.97±0.02**
	GRAAL	0.80	23.5	2.61	22.39	1.5	0.90	0.94	0.92±0.01	0.96±0.02
OpTC-6hosts	GAE*	×	128	1.1k	281	22	0.11	0.85	0.19	0.53
	Kairos*	×	47	72	1.2k	103	0.40	0.31	0.35	0.63
	Magic*	×	82	614	740	68	0.12	0.55	0.19	0.55
	GRAAL	×	60	1	1.4k	90	0.98	0.40	0.57	**0.74**

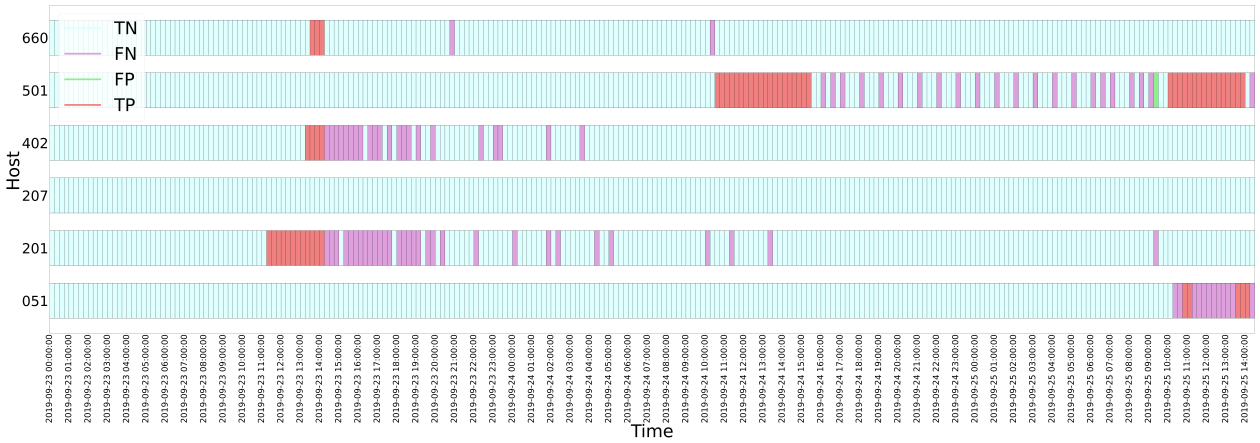
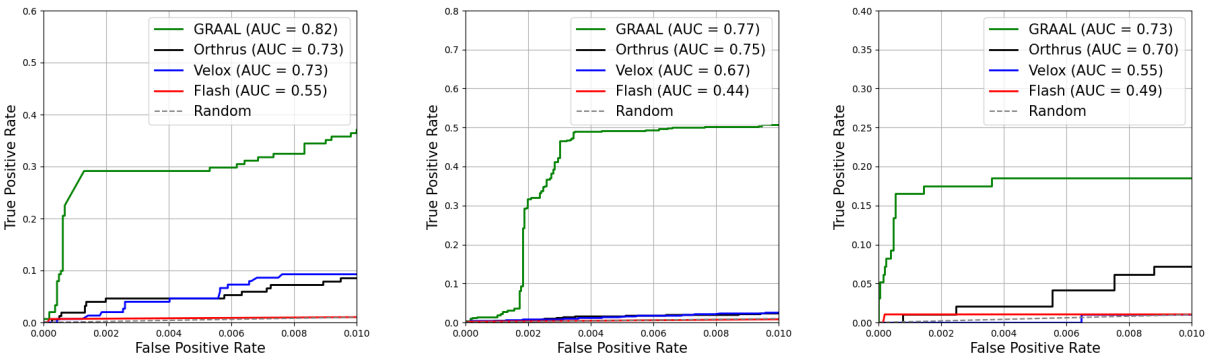


Figure 11: Detailed graph detection of GRAAL on 6 hosts of the DARPA OpTC dataset.

TABLE 10: Detailed entity-level detection comparison on three hosts of the DARPA OpTC Dataset (*threshold is biased)

Host	IDS	TP	FP	TN	FN	Prec-ision	Recall	F1-score	AUC	ADP
201	**Orthrus**	1	0	18.6k	519	**1.00**	0.00	0.00	0.75	**0.99**
	Velox	2	7	18.6k	518	0.22	0.00	0.01	0.67	0.39
	Flash*	4	143	18.5k	516	0.03	0.01	0.01	0.44	0.32
	GRAAL	2	0	18.6k	518	**1.00**	0.00	0.01	**0.77**	**0.99**
501	Orthrus	0	1	20.7k	97	0.00	0.00	0.00	0.70	0.06
	Velox	0	1	20.7k	97	0.00	0.00	0.00	0.55	0.02
	Flash*	5	884	19.9k	92	0.01	0.05	0.01	0.49	0.25
	GRAAL	16	13	20.8k	81	**0.55**	0.17	0.25	**0.73**	**1.00**
051	Orthrus	0	3	16.1k	151	0.00	0.00	0.00	0.73	0.25
	Velox	0	0	16.1k	151	0.00	0.00	0.00	0.73	0.12
	Flash*	4	140	16.1k	147	0.03	0.03	0.03	0.55	0.25
	GRAAL	34	11	16.2k	117	**0.76**	0.23	0.35	**0.82**	**1.00**

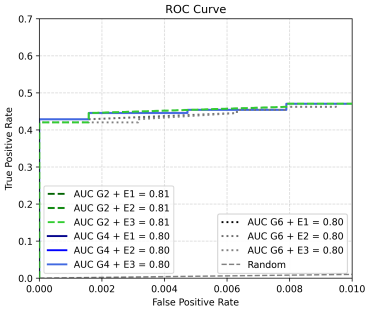


(a) ROC curve of host 051. (b) ROC curve for host 201. (c) ROC curve for host 501.

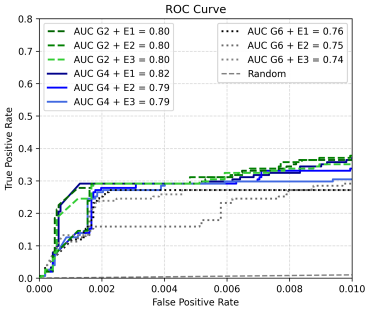
Figure 12: ROC curves limited to 0.01 false positive rate (FPR).

TABLE 11: Results on DARPA OpTC for internal GRAAL comparison

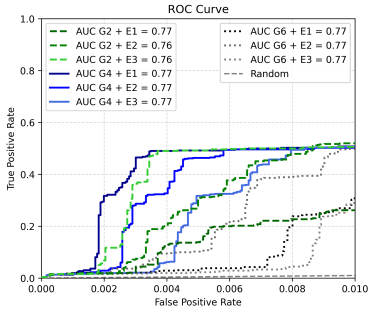
Detection	IDS	TP	FP	TN	FN	Precision	Recall	F1-score	AUC	ADP
Graph 6hosts	OCSVM	66	25	1.3k	84	0.73	0.11	0.55	0.81	×
	GRAAL$_g$	61	2	1.4k	89	0.97	0.41	0.57	0.74	×
	GRAAL	60	**1**	1.4k	90	**0.98**	0.40	0.57	0.74	×
Process host 201	OCSVM	0	0	18.6k	520	0.00	0.00	0.00	0.66	0.99
	GRAAL$_e$	2	3	18.6k	518	0.40	0.00	0.01	0.77	0.99
	GRAAL	2	**0**	18.6k	518	**1.00**	0.00	0.01	0.77	0.99
Process host 501	OCSVM	9	11	20.7k	88	0.45	0.09	0.15	0.61	0.67
	GRAAL$_e$	16	16	20.7k	81	0.50	0.17	0.25	0.73	1.00
	GRAAL	16	**13**	20.8k	81	**0.55**	0.17	0.25	0.73	1.00
Process host 051	OCSVM	3	6	16.2k	148	0.33	0.02	0.04	0.77	0.50
	GRAAL$_e$	44	21	16.2k	107	0.68	0.29	0.41	0.82	1.00
	GRAAL	34	**11**	16.2k	117	**0.76**	0.23	0.35	0.82	1.00



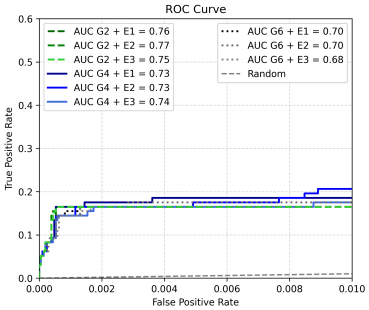
(a) Graph detection ROC curve (host 051, 201 and 501).



(b) Entity detection ROC curve of host 051.



(c) Entity detection ROC curve for host 201.



(d) Entity detection ROC curve for host 501.

Figure 13: GRAAL ROC curves limited to 0.01 false positive rate (FPR). GRAAL-g's numbers of layers vary between 2 and 6 (G2, G4 and G6). GRAAL-e's additional number of layers vary between 1 and 3 layers (E1, E2 and E3).

Towards Understanding Alerts raised by Unsupervised Network Intrusion Detection Systems

Maxime Lanvin
CentraleSupélec, Univ. Rennes, IRISA
France
maxime.lanvin@centralesupelec.fr

Pierre-François Gimenez
CentraleSupélec, Univ. Rennes, IRISA
France
pierre-francois.gimenez@centralesupelec.fr

Yufei Han
Inria, Univ. Rennes, IRISA
France
yufei.han@inria.fr

Frédéric Majorczyk
DGA-MI, Univ. Rennes, IRISA
France
frederic.majorczyk@intradef.gouv.fr

Ludovic Mé
Inria, Univ. Rennes, IRISA
France
ludovic.me@inria.fr

Eric Totel
Samovar, Télécom SudParis, Institut Polytechnique de Paris
France
eric.totel@telecom-sudparis.eu

ABSTRACT

The use of Machine Learning for anomaly detection in cyber security-critical applications, such as intrusion detection systems, has been hindered by the lack of explainability. Without understanding the reason behind anomaly alerts, it is too expensive or impossible for human analysts to verify and identify cyber-attacks. Our research addresses this challenge and focuses on unsupervised network intrusion detection, where only benign network traffic is available for training the detection model. We propose a novel post-hoc explanation method, called *AE-pvalues*, which is based on the p-values of the reconstruction errors produced by an Auto-Encoder-based anomaly detection method. Our work identifies the most informative network traffic features associated with an anomaly alert, providing interpretations for the generated alerts. We conduct an empirical study using a large-scale network intrusion dataset, CICIDS2017, to compare the proposed *AE-pvalues* method with two state-of-the-art baselines applied in the unsupervised anomaly detection task. Our experimental results show that the *AE-pvalues* method accurately identifies abnormal influential network traffic features. Furthermore, our study demonstrates that the explanation outputs can help identify different types of network attacks in the detected anomalies, enabling human security analysts to understand the root cause of the anomalies and take prompt action to strengthen security measures.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; • **Computing methodologies → Machine learning**.

KEYWORDS

intrusion detection, machine learning, explainable AI (XAI)

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
RAID '23, October 16–18, 2023, Hong Kong, Hong Kong
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0765-0/23/10…$15.00
https://doi.org/10.1145/3607199.3607247

ACM Reference Format:
Maxime Lanvin, Pierre-François Gimenez, Yufei Han, Frédéric Majorczyk, Ludovic Mé, and Eric Totel. 2023. Towards Understanding Alerts raised by Unsupervised Network Intrusion Detection Systems. In *The 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23), October 16–18, 2023, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3607199.3607247

1 INTRODUCTION

Recent years have witnessed the flourishing of the deployment of Machine Learning (ML)-based Network Intrusion Detection Systems (NIDS) [3]. ML techniques, especially end-to-end deep learning methods, can conduct automated feature engineering over the attributes of network traffics, such as application protocol, TCP flags, or payload size. Furthermore, ML-assisted NIDS can obtain prompt detection results and offer flexible detection covering various attacks, which help security operation teams (SOCs) reach fast responses to emerging new security incidents during day-to-day security practices.

Despite its advantages, ML-driven Network Intrusion Detection Systems (NIDS) are susceptible to producing a high rate of false positives in their detection output. In light of the increasing volume of network traffic in IT assets, a high false positive rate can lead to prohibitively expensive inspection efforts by human security analysts in Security Operations Centers (SOCs), who are tasked with reviewing the raised alerts. This can result in overwhelmed analysts, leading to delayed responses to potential threats, commonly known as "alert fatigue" [12]. The root cause of this issue stems from the black-box nature of ML-based detection methods. Since these methods lack human-understandable interpretations of the detection results, it becomes difficult for analysts to verify and monitor incident alarms triggered by the ML-driven model. Therefore, improving the transparency of the decision logic underlying the detection output of ML-driven NIDS is necessary.

Our research echoes the challenge of developing eXplainable Artificial Intelligence (XAI)-based solutions for Network Intrusion Detection Systems (NIDS). Specifically, our study focuses on unsupervised anomaly detection, as explored in previous works such as [10, 16], where no labeled attacks are available for training the detection model, and only benign traffic data is used to capture

the profiles of normal network activities. The core methodology of anomaly detection involves identifying traffic with significantly deviated profiles from normal traffic as abnormal activity. For instance, Leichtnam et al. [16] utilized an Auto-Encoder (AE)-based deep neural network to reconstruct the network traffic data, with any traffic that produced a large reconstruction error being considered an anomaly. Similarly, Ede et al. [10] employed a recurrent neural network model to predict the next system logs based on previously observed logs, using the prediction result to determine whether the observed log sequence contained abnormal activities.

In recent years, XAI methods, such as LIME [21] and SHAP [18], have been developed to provide post-hoc explanations of classification/detection outputs by identifying the most important features. As post-hoc methods, they are used on top of existing models and can be combined with any Machine Learning model architectures, which offers great flexibility in practices of XAI techniques. However, these methods face practical challenges when applied to security analysis. Firstly, SHAP is computationally expensive and suffers from inclusion of unrealistic data instances when features are correlated. Secondly, although LIME is computationally efficient, its linear surrogate model cannot accurately approximate complex model architectures. Additionally, neither SHAP nor LIME is adapted to a unsupervised context where only benign data is available for training. While DeepCase and ROULETTE [4] have been proposed as alternatives, they have limitations. DeepCase assumes that sequential causality exists between logs of normal system behaviors, which may not hold true in complex and environment-dependent correlations between network traffic attributes. ROULETTE is a supervised approach and uses actual classes of data for the learning task, which is a stronger assumption than our work that only uses benign traffic for learning. Therefore, in our work, we propose a novel XAI-based solution to NIDS that is free from such assumptions and specifically designed for unsupervised anomaly detection.

Our contribution can be summarised in the following perspectives:

- We propose a new XAI approach, namely *AE-pvalues*[1], which is designed to not only identify the important features responsible for unsupervised anomaly detection but also help categorize the detected anomalies into specific attack types. We instantiate our study with a state-of-the-art unsupervised Auto-Encoder-based NIDS method [16].
- We organize a comprehensive experimental study to evaluate quantitatively the usefulness of the explanation results produced by *AE-pvalues* and various state-of-the-art XAI methods on the CICIDS2017 dataset. We demonstrate that our method can offer significantly more accurate explanation results to the identified security incidents.
- We organize a use-case study which is divided into two parts. First, we show that the explanations provided can be practically used to categorize the detected security incidents into the corresponding attack types with high precision. In practice, the explanation results from our method can facilitate human experts to understand the campaigns of the detected attack behaviors. Second, we conducted a manual inspection over the explanation results generated by our method on the network attack behaviors in CICIDS2017. We show the explanations are highly consistent with the behavior of the associated attack type.

[1]We provide an implementation here https://gitlab.inria.fr/mlanvin/ae-pvalues

The remainder of this paper is organized as follows. Section 2 is a presentation of Sec2graph which is the detection method we used to obtain the alerts. The CICIDS2017 intrusion detection dataset is also presented. Section 3 presents related work on XAI approaches used in the context of intrusion detection. Section 4 presents the new XAI technique we propose, and Section 5 contains a benchmark to compare its performance to other XAI techniques. Finally, Section 6 shows how the explanations we produce can be used in practice on alerts generated from the CICIDS2017 intrusion detection dataset. Section 7 concludes the paper.

2 BACKGROUND

For this work, we used Sec2graph as the network intrusion detection system to explain. The first subsection presents this approach. We then present the CICIDS2017 dataset we use in our experiments.

2.1 Sec2graph

Sec2graph [16] is one of many unsupervised approaches [6, 19] relying on the reconstruction error of an Auto-Encoder to detect anomalies. This approach can be decomposed into three steps: the graph building, its encoding, and finally, how the detection is performed.

2.1.1 Building a graph of security objects. The raw network captures (.pcap files) are analyzed using Zeek to extract logs regarding different network protocols. Then a "security objects graph" is built from these logs to reveal the structure in the data and connect elements of different kinds. The Figure 1 shows an example of a security objects graph constructed from network logs. In this example, a client requested the IP address of the domain `google.com` via the DNS protocol, then contacted the `google.com` server with an HTTP request to obtain an SSL certificate before initiating an HTTPS connection with the same server. Since the graph is constructed from logs of these various protocols, the graph's nodes correspond to the network information in this scenario (client, DNS server, HTTPS server, network ports, network connections, SSL certificate, etc.). These nodes are directly linked when the information they represent appears in the same network event. Due to the diversity of nodes and edges, the graph is heterogeneous. The whole security objects graph model is available in the appendix A.

2.1.2 Encoding a graph of security objects. The second step of the approach is to encode a graph in the form of vectors usable by the Auto-Encoder. In this encoding, each edge of the graph is processed independently and is vectorized. This vector encodes the triplet *(source node, edge type, destination node)*. The encoding of the edge type, the source, and destination nodes attributes are concatenated to create the vector of each triplet. Remark that there is only one Auto-Encoder but several types of nodes and edges. For this reason, the vectors are sparse: every dimension of the vector related to a type not present in an edge is set to zero.

Sec2graph uses one-hot encoding, a classical technique to encode categorical variables. For each categorical variable, it creates

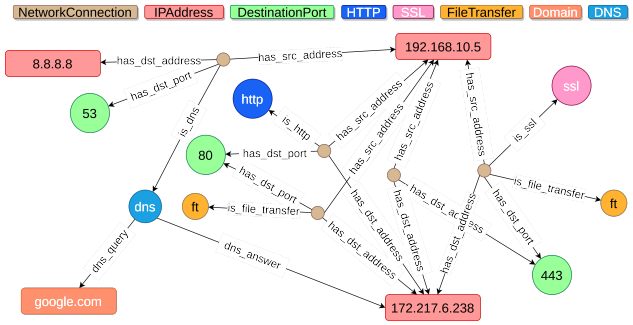


Figure 1: Example of security object graph in *Sec2graph*

a vector whose size is the number of categories. Each feature of this vector contains a 0, except the feature associated with the variable category that receives a 1. Discrete and categorical attributes are processed differently. For discrete attributes (such as port numbers or IP addresses), the number of possible categories is limited by keeping the most frequent categories and merging the rare categories into a single one. One can observe an example of the categorical encoding on the Figure 3 where the vectors represent the protocol encoding. For continuous attributes (such as flow duration or the number of packets exchanged), a clustering for each attribute is performed with a Gaussian Mixture Model (GMM). The resulting cluster membership labels are used as the categories in the one-hot representation. In the end, both categorical and continuous attributes are treated as categorical attributes.

2.1.3 Anomaly detection. In the theoretical study of machine learning, anomaly detection can be formulated as a problem of one-class learning, where only the benign data are labeled. Any testing samples deviating significantly from the benign training samples are considered as anomalies [9]. One-class learning is a branch of the family of semi-supervised learning techniques, which try to recover the classification boundary despite the lack of fully tagged training data. In our work, we follow the spirit of one-class learning methods. We train the AE-based detection model using only benign traffic. The model is supposed to cover the profile of benign traffics as much as possible. On the contrary, we do not use any attack data tagged by human experts or signature-based IDS for training. From the perspective of security practices, this is an unsupervised learning scenario, as no supervision information regarding the attack events (labeled attack events) is provided. We hence attribute our method to unsupervised detection hereafter.

For this one-class classification problem, Sec2graph uses an Auto-Encoder to learn how to accurately reconstruct the normal network traffic. Then, during the detection phase, the Auto-Encoder produces a reconstruction error of an input network traffic record. The reconstruction error is the binary cross-entropy between the input and the output of the Auto-Encoder. A lower (resp. higher) reconstruction error indicates that the corresponding input is less (resp. more) likely to represent an anomaly.

In the detection phase, the reconstruction error for each edge of the security objects graph is computed. The average reconstruction error for all the edges relative to a single network connection is then compared to a predefined threshold. Any network connection with an anomaly score above the threshold is considered as an anomaly.

2.2 CICIDS2017 dataset

CICIDS2017 [23] is a popular dataset for evaluating network intrusion detection systems. This dataset includes the network traffic of an IT infrastructure of a dozen of machines. The traffic was recorded over five days, including one day, the first one, without attack and four days with attacks. The first day is typically used for learning by one-class detection models, which only use the benign class to learn the normal behavior. Although this dataset is generally considered to have a good quality, recently, some studies [15, 17] found several mistakes, including labelling issue, in the dataset. Thus, we use the fixed version of the labels proposed by these articles. Different types of network attacks and the number of network connections belonging to each attack type are shown in the Table 1.

Attack types	Counts	Attack types	Counts
benign	1614317	heartbleed	1
botnet	736	infiltration	7
ddos	95146	infiltration - portscan	60867
dos goldeneye	7650	portscan	159043
dos hulk	162113	ssh-patator	2960
dos slowhttptest	1780	web - brute force	73
dos slowloris	2232	web - sql injection	13
ftp-patator	3972	web - xss	18

Table 1: Number of network connections per attack type and for the benign traffic in the CICIDS2017 dataset.

3 RELATED WORK

According to the taxonomy presented in [1], we can categorize XAI approaches into several main branches.

Intrinsically explainable models.

These types of XAI methods encompass simple and interpretable ML models such as linear or logistic regressions, decision trees, naive Bayes, and K-Nearest Neighbours. These models possess straightforward structures that can identify important features that trigger the decision of a given input. For instance, decision trees can rank feature importance by assessing the information gain obtained by selecting a specific feature in the decision chain. Linear and logistic regression models can evaluate the impact of different features on the decision output by utilizing sparsity-inducing constraints such as L1 regularization in Lasso/Elastic net regression methods and using the magnitudes of the coefficients. However, these models may underfit complex associations in the training data, leading to a loss of utility due to their simple model architecture. In our case, we propose producing post-hoc explanations directly from the output of the AE model. By doing so, the AE-based detection model can capture the underlying nonlinear relations between input attributes, which maintains utility for anomaly detection. Moreover, we demonstrate that the output from the AE-based reconstruction provides sufficient information to produce indicative explanations

of the contribution of each network traffic attribute in the detection task.

Model-agnostic explanation methods. These methods provide post-hoc explanations regardless of the architecture of the target ML models. Global and local surrogate function based techniques [21] learn to approximate the global classification boundary of the target ML model or local classification boundary around the testing input. The surrogate function is intrinsically explainable, e.g., a linear or decision tree-like classifier. By fitting the surrogate function to the true classification boundary, we can estimate the features' contribution to the classification output. A widely applied method in this branch is LIME [21], which can be categorized as a local surrogate function based method. Using a linear model, it approximates the target model locally around a given testing input. The magnitudes of the linear coefficients are considered as the base for the explanation. Similarly, SHAP [18] leverages the Shapley values to compute each input feature's contribution and then provide explanations.

There are also other XAI methods beyond the two major categories. For example, counterfactual analysis [11] and the influence function-based method [14] are developed to find influential data points close to the classification boundary of the target ML model. These influential data points are used to understand the importance of different features in triggering a classification output. However, most of the off-the-shelf XAI methods [8] are adopted for supervised learning tasks. They require fully labeled training data to approximate the true classification boundary and measure the usefulness of different feature dimensions. For this reason, they are not compatible with the scenario of unsupervised anomaly detection in our work, where only benign training samples are provided.

It is worth noting that there are a few XAI methods coping with the anomaly detection task [2, 4, 13, 19, 24]. [4] proposed ROULETTE to reformulate anomaly detection as a binary classification problem differentiating benign and outlier data. Then it adopts a self-attention module in the classification model to provide explanations. [13, 19] used the gradient of an Auto-Encoder-based anomaly detection model to deliver axiomatic feature importance evaluation. However, these methods consider continuous attributes as inputs to the detection model. In our study, the network traffic attributes are mostly categorical features, such as the user agent string or TCP flags. It is thus infeasible to use gradients as the indicator to feature-wise contribution to the detection output. In contrast, our proposed explanation methods is developed to adapt to unsupervised anomaly detection and handle both categorical and numerical inputs.

4 THE ALGORITHM DESIGN OF AE-PVALUES

4.1 Notations

Let b denote one categorical network traffic feature in our study. To facilitate the training of the model, we transform b into a one-hot encoded vector. The categorical feature b is unfolded into a k-dimensional binary vector $\{b_j\}$ $(j = 1, 2, 3, ..., k)$, where b has k possible category values. For example, the feature *protocol* has three possible category values *tcp*, *udp*, and *icmp*. Therefore the protocol feature is encoded into a 3-bit binary vector, one bit per category value. The bit is valued as 1 if the protocol feature carries the corresponding category value. In the end, the one-hot encoded vectors of each network traffic feature are concatenated into a high-dimensional feature vector x as input to the Sec2Graph-based detection model. Each x_i denotes the binary status of one category value of a network traffic feature. The corresponding reconstruction output from the Sec2Graph model is denoted correspondingly as $\hat{x}$.

4.2 Explanation Algorithm Design

Following the design of Sec2Graph, we can derive the dimension-wise reconstruction error regarding each feature dimension x_i. As we observe, the empirical distributions of the reconstruction errors per feature dimension x_i vary significantly. There exist feature dimensions that exhibit significantly higher variance of reconstruction errors, even in benign network traffic. Such feature dimensions are inherently more challenging to distinguish from truly abnormal feature values based solely on the absolute values of dimension-wise reconstruction errors. Specifically, in cases where feature dimensions exhibit large variances of reconstruction error, the magnitude of the dimension-wise reconstruction error may not necessarily imply the presence of abnormal activities.

As an echo, we consider using the p-value of the empirical distribution of the dimension-wise reconstruction error to flag abnormal feature values. Let H_0 and H_1 be the null and alternative hypotheses. H_0 and H_1 assume the reconstruction error of x_i is within the normal range or not respectively. We use r_i to denote the reconstruction error variable corresponding to x_i in theory. e_i denotes the empirically observed reconstruction error for x_i given an input x to Sec2Graph. e_i is given by the difference between the reconstructed variable $\hat{x}_i$ and x_i its input value. The p-value is then defined as the probability $p_i = \mathbb{P}(r_i > e_i)$. In our study, we take the empirical distribution $\hat{\mathbb{P}}$ of the reconstruction error of x_i as the reference distribution in H_0. We then compare the observed e_i to $\hat{\mathbb{P}}$ to compute the p-value of e_i with respect to $\hat{\mathbb{P}}$. A smaller p-value means that the H_0 hypothesis is more likely to be rejected, i.e. x_i has a large reconstruction error with respect to $\hat{\mathbb{P}}$ and vice versa. Given the feature dimension-wise p-value, we can rank the dimensions based on their probability of respecting the null hypothesis and obtain the explanation list.

Using the p-value to flag abnormal feature values alleviates the bottleneck of the L1-distance-based criterion for tagging anomalies. Computing p-values takes the variance of the per-dimension empirical reconstruction error $\hat{\mathbb{P}}$ into consideration. Therefore, whether or not the variance of $\hat{\mathbb{P}}$ is large does not change the derived p-value. In practice, the dimension-wise $\hat{\mathbb{P}}$ is a discrete empirical distribution, as the values of reconstruction error observations are discrete. As a result, we adopt the following computation in Eq 1 to obtain the p-value and produce the explanation results, which gives:

$$p_i = \frac{\#\{r_i \geq e_i\}}{\#\{r_i\}} \tag{1}$$

Improving the sensitivity of explanation. For the feature dimensions contributing extremely large reconstruction error, the derived p-value can become arbitrarily small, approaching to 0. In that case, the p-value is not differentiable enough to compare the usefulness of the features, i.e., these feature dimensions have equally small vanishing p-values. To increase the sensitivity of our explanation, we complement the proposed *AE-pvalues* method in

this extreme situation by inspecting the difference between the empirical quantiles of the dimension-wise reconstruction error, which gives as below:

$$\alpha_i = \frac{e_i - m_i}{p_i^{99} - p_i^1} \tag{2}$$

where m_i is the median value (the 50-th percentile) of the empirical reconstruction error. p_i^{99} and p_i^1 are the 99-th and 1-st percentiles of the empirical reconstruction error. According to Eq 2, we take the difference between the 1-st and 99-th percentiles of the empirical reconstruction error of x_i. This difference is considered as the estimate of the variance range of the reconstruction error. After that, we perform the normalization of e_i by centering it with m_i and scaling the error by dividing the difference between percentiles to further remove the influence of the variance. The derived normalized reconstruction error value, noted as α_i, is particularly used to evaluate the anomaly level of the corresponding feature x_i in the case when e_i is extremely large, and p_i is equal to 0. More generally, the α_i are useful for ranking the feature dimensions when several share the same p-values score.

We raise a toy example in Figure 2 to demonstrate the benefit of using p-value instead of L1 distance. We create two toy variables following Gaussian distributions. The mean values and variances of the two Gaussian distributions are -0.1 and 0.1, and 0.2 and 0.15 respectively. These two variables correspond to the distributions of reconstruction error of the two toy features (feature dimensions 1 and 2, respectively, in the figure). The variance of the reconstruction error of feature dimension 1 is less dispersed than that of feature dimension 2. The two vertical lines denote the p-value-determined threshold to trigger anomaly alerts for the two features. Without loss of generality, we set $p = 0.25$ in this example. The p-values are represented by the green and the blue areas in the Figure 2. As seen, no unique L1 distance-based threshold value can be applied to both features 1 and 2. For example, the threshold used for feature dimension 1 may produce a high false positive rate for feature dimension 2, represented by the grey area on the Figure 2. Instead, we can use a fixed p-value in this example to adaptively set the threshold to detect anomalies for both features, regardless of the reconstruction error variance.

Besides, we involve two explanation baselines to organize the comparative study. The first method noted as *AE-abs*, is a variant of the proposed *AE-pvalues*. It ranks feature dimensions x_i by the L1 distance-based reconstruction error per x_i, namely by the $|\hat{x}_i - x_i|$ scores. We introduce *AE-abs* to show the benefit of using p-values instead of the magnitude of the reconstruction error, given the varying variance of the per-dimension reconstruction error.

The second method adopts SHAP for Auto-Encoders [5], noted as *SHAP_AE*. The core idea of *SHAP_AE* is to use Shapley values to measure the contribution of each input dimension x_i to the highest dimension-wise L2 distance-based reconstruction errors among Sec2Graph's output. We then rank the Shapley values of each x_i to prioritize the feature dimensions contributing the most to the reconstruction error.

5 EXPERIMENTAL EVALUATIONS

We first explain the empirical protocol used to set up the comparison between different explanation techniques in Section 5.1. Then, we present the results of the comparative evaluation on the CICIDS2017 intrusion detection dataset in Section 5.2.

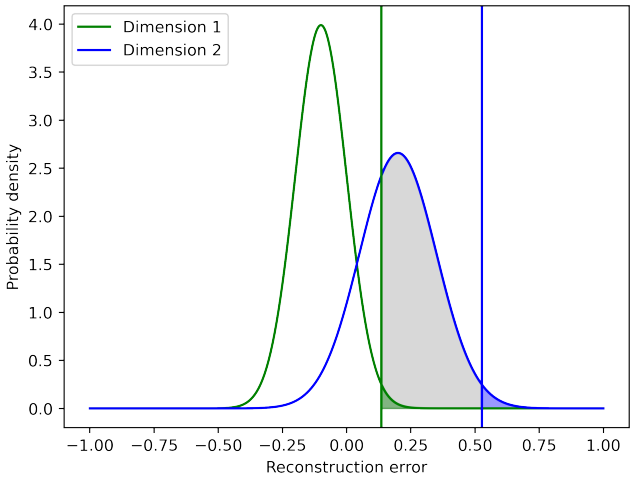


Figure 2: The reconstruction error distributions of two feature dimensions.

5.1 Experimental protocol

5.1.1 Problem statement. The aim of the explaining methods is to identify correctly where the anomaly is located in the vectors. Each **vector** represents an edge in the graph as reminded in the Subsection 2.1.2. The vector is a representation of all the **network traffic features** of the nodes that are associated with the edge. These network traffic features are, for instance, the protocol, the HTTP method, and the browser. A comprehensive list of them is available in the appendix A which is the Sec2graph model description. Each of these network traffic features has several **category values**. For example, (icmp, udp and tcp) are the category values for the network feature protocol, (GET, POST, PUT, DELETE, other) would be those of HTTP method. In the rest of the article, we named **feature dimensions** the whole list of the category values.

5.1.2 Protocol. **Evaluating the accuracy of the explanation results:**

Our approach involves evaluating different explanation methods by measuring their accuracy in identifying both the network traffic feature and the perturbed category values responsible for producing anomalies. We evaluate the identification at two levels, and our evaluation methodology is based on the widely used one-factor-at-a-time sensitivity analysis method [7]. To ensure that no anomalies are present, we use vectors from days without attacks. For each network traffic feature such as *protocol*, *conn_state* or *file-transfer_depth* and the others , we select 100 feature vectors without attacks and introduce artificial perturbations to the chosen feature of each vector. In each of the 100 vectors we create a perturbation in the chosen network feature by changing the position of the selected feature dimension as shown in Figure 3. To perturb categorical network attributes, such as the public/private status of an IP address, we replace the original category value with a randomly selected value. For binary features, we simply flip the original values. Next,

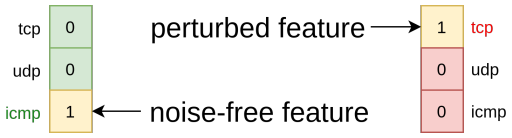


Figure 3: Example of noise insertion in a network feature (protocol). This network feature has three category values (tcp, udp, icmp), thus encoded to a 3-dimensional one-hot encoded representation. The one-hot dimension containing originally a 1 is named "noise-free feature", and the dimension containing, in the end, the 1 is called "perturbed feature"

we evaluate whether the explaining methods can identify the perturbed category values in each feature vector where artificial noise is injected. Each explanation method ranks the feature dimensions corresponding to all category values according to their anomaly scores and identifies the top-K feature dimensions as the most likely to present abnormal values. We refer to this list as the top-K list.

As explained in Section 2.1.1, different kind of vectors exists in the graph because of the heterogeneous nature of the edges in the Sec2graph representation. The use of certain features may vary depending on the type of vectors that are selected. To guarantee the effectiveness of the evaluation, we decided to only inject noise into vectors for which the feature-to-perturb was already used. This prevents the introduction of artificial embeddings that do not exist in the data because all the network traffic features are not used for all of the edge types. These kinds of anomalies are too easy to detect and unrealistic. The explanations are edge-specific. Indeed, each vector corresponds to an edge in the graph and has a type. We only keep the explanations that are meaningful with the edge type. For instance, when assessing the explanations regarding a vector that corresponds to an edge that is linking an HTTP object and a NetworkConnection object, we do not take into account the feature dimensions that are associated with DNS, SSH, or any other non-related objects. Thus all the dimensions non-related to the two objects of the edge are discarded.

In the comparative study, we involve *AE-pvalues* and *AE-Abs* explaining methods. A *Random* strategy is also compared with the other methods and consists in ranking the feature dimensions randomly. In addition to these methods, we also include *SHAP_AE* [5] in the comparative study. *SHAP_AE* uses the model-agnostic approximation of SHAP values, a.k.a. Kernel SHAP [21]. It needs a background set to approximate the model locally with another linear model. This background set is made up with 200 vectors from the training set of the Auto-Encoder used in Sec2graph. Similarly, *AE-pvalues* also needs a background set to estimate the normal distributions of the reconstruction errors for each feature dimension. We provided the whole train set of the Auto-Encoder to have as much diversity as possible.

To set up the ground truth to measure the accuracy of the explanation results, we include the feature dimensions manually perturbed in the ground truth of the explanation evaluation. Beyond that, we also include the other feature dimensions statistically significantly correlated to the manually perturbed ones (with a significance level of less than 0.05). For example, if the perturbation is injected into the *http_code_200* category value and the first returned explanation is the category value associated with the *http_msg_OK* attribute, which is confirmed to have the same physical significance with *http_code_200*. Therefore, if the feature dimension manually perturbed or any other feature dimensions highly correlated with the perturbed ones exist in the top-K list, the explanation results are considered to be effective. In the rest of the paper, when the "_corr" suffix is added to any method name, it denotes that we match not only the perturbed dimensions but also the significantly correlated ones to the top-K features ranked by the explanation output. We define two metrics by finding the matches feature dimensions in the top-K list.

The mean rank (noted as Rank): we first compute the ranking index of the matched features in the top-K list produced by each explanation method. We then compute the mean rank of the matched features among all the vectors selected in the test. A lower value of the rank denotes a more accurate identification of the features containing anomalies.

The top-K accuracy (Accuracy − K): we compute the fraction of the matched features in the top-K list. The higher the accuracy score is obtained, the more accurate the explanation method is in identifying features containing anomalies.

5.2 Experimental results

Results of the mean rank. In Figure 4, we present the average rank of both the noise-free feature and the perturbed feature, along with the rank of the network feature itself. In the example of the Figure 3, *icmp* is the noise-free feature, *tcp* is the perturbed feature, and *protocol* is the network feature. The Table 2 lists the mean ranks provided by different explaining methods. All the involved explanation methods are evaluated with and without taking into account the correlated network traffic features. As shown, *AE-pvalues* provides consistently lower mean rank values compared to the other methods, no matter whether the correlated features are taken into consideration. The results indicate that *AE-pvalues* can better prioritise the features that contain anomalies in the explanation results than the other opponents.

Results of top-K accuracy. Figure 5 illustrates the top-K accuracy obtained by different explanation methods. We conduct the analysis in two steps. We first compare the explanation methods by taking into account the correlations. Then we revisit the comparison between different methods without taking into account the correlations. In the first case, where correlations are taken into account, we can find that *AE-pvalues* obtains the highest accuracy. In contrast to the other two methods, *SHAP_AE* (evaluated with or without considering the correlations) is always less accurate than the others except for $K = 1$. Even increasing the number of explanations, *SHAP_AE* still have more difficulties to identify the relevant feature than the other approaches. When comparing the different methods without taking into account the correlations, we can observe that *AE-abs* and *SHAP_AE* perform better for the top-1 accuracy. *AE-pvalues* certainly provides a correlated attribute in the first position more often than the two other methods. However, when we take the top-K with $K > 2$, *AE-pvalues* becomes the most accurate method. That is we are more likely to obtain the abnormal feature in the provided top-K explanations. As we can find, *SHAP_AE* delivers less accurate explanations compared

explaining method	Mean rank of the noise-free feature dimension	Mean rank of the perturbed feature dimension	Mean rank of the network feature
AE-pvalues_corr	**1.96**	**0.63**	**0.02**
AE-abs_corr	2.89	0.61	0.07
SHAP_AE_corr	3.71	3.44	0.26
AE-pvalues	**3.61**	**2.07**	**0.39**
AE-abs	4.78	3.78	0.49
Random_corr	4.68	15.3	0.85
SHAP_AE	17.96	6.18	1.15
Random	25.93	26.13	6.8

Table 2: Table of mean ranks of the noise-free feature, the perturbed feature, and the network feature where the noise is inserted. See Figure 3 for more precise column name definitions.

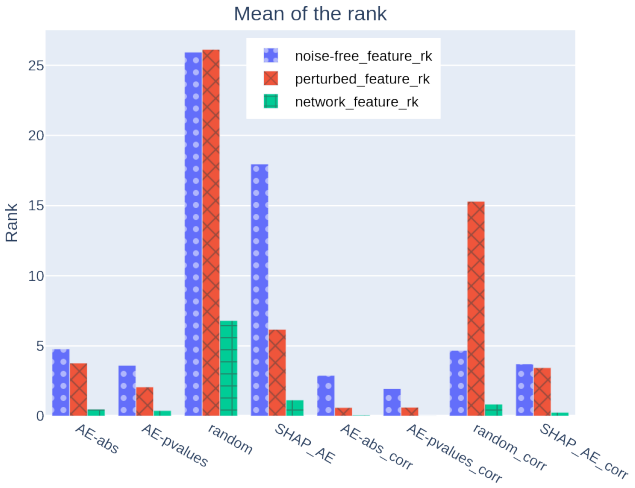


Figure 4: Mean rank comparison between the different explaining methods.

to *AE-pvalues* and *AE-abs*. We further demonstrate the superior accuracy of the explanation results produced by *AE-pvalues* in Figure 6. We first aggregate the variance of reconstruction error per feature dimension to obtain the averaged variance of reconstruction error for each network traffic feature. We then split all the network traffic features into two separate subsets. One set contains the features with an averaged variance larger than 10^{-6} (noted as the large variance subset), and the other set contains those with an averaged variance less than 10^{-6} (noted as the small variance subset). Figure 6 illustrates the top-K accuracy of all the involved explanation methods over the larger and small variance feature subsets. As shown in the right side plot of Figure 6, the proposed *AE-pvalues* gives significantly higher top-K accuracy than *AE-abs* and *SHAP_AE*, especially with K less than 4. By comparison, the top-K accuracy of *AE-pvalues*, *AE-abs* and *SHAP_AE* are close when the average variance of network traffic features is small according to the left side plot. The observation echos the motivation of using p-values instead of *AE-abs* and *SHAP_AE*. The limit of *AE-abs* and *SHAP_AE* is rooted in the use of L1 distance based reconstruction error as the measurement to evaluate feature informativeness. The L1 distance only indicates the magnitude of the reconstruction error. It does not consider the baseline variance of the reconstruction error under the noise-free scenario. In case the variance of the reconstruction error is large in the noise-free case, a large magnitude of L1 distance-based reconstruction error does not necessarily indicate the existence of anomalies. By incorporating p-value selection, the explanation output of *AE-pvalues* is more robust to variations in the reconstruction error's variance. Consistently, the average ranks of the good explaining features and category values are better using *AE-pvalues* method as shown in the Table 2.

5.3 Performance considerations

Beyond the relevance of the explanations brought by the different explaining methods, we also measured the scalability of the different methods. In the Table 3, we present the time to process the explanations for a single sample. The measurements were done on 100 vectors (samples) to explain, and the values represent the average processing time. Given the table, on the one hand, we clearly see that *SHAP_AE* does not scale well since the duration is far too long to process many vectors. On the other hand, *AE-pvalues* method is much faster and thus can be used to process many vectors. It is important to mention that *AE-pvalues* method takes approximately 150 additional seconds as a pre-processing step to compute the distributions associated with the "normal" behaviors. This duration only depends on the size of one vector. Here the size of each vector is 400 dimensions. It is only computed once for all and does not depend on the number of samples to explain. Moreover, the implementation of *AE-pvalues* can be easily parallelized and thus provide even much more efficient performances for many samples to explain.

Method	Processing time per sample
SHAP_AE	28 s
AE-pvalues	1.9 ms
AE-abs	1.0 ms

Table 3: Processing time for one sample for each explaining method

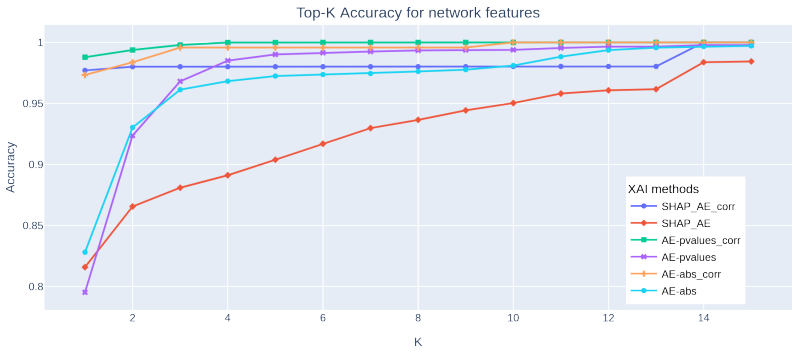


Figure 5: Comparison of the top-K accuracy of the explanations of the different XAI methods

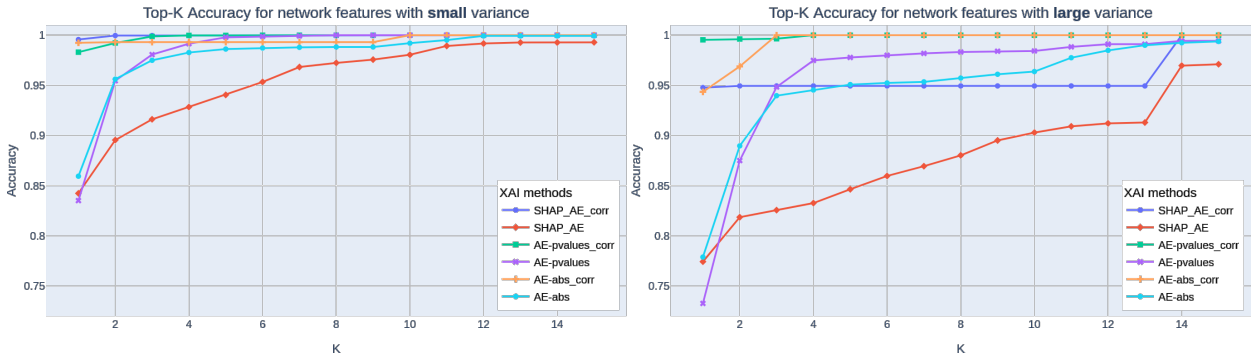


Figure 6: Comparison of the top-K accuracy of the explanations of the different XAI methods. The left Figure is for small variance network features, and the right Figure is for the high variance ones.

6 EXPLANATION IN THE CONTEXT OF ANOMALY DETECTION

Through Section 5, we analyzed different explaining methods. From the benchmark, we know *AE-pvalues* is the most accurate explaining method. Moreover, *AE-pvalues* is scalable. For all these reasons, the explanations obtained hereafter will be only based on *AE-pvalues*. In the following, we demonstrate how human analysts can benefit from these explanations to analyze alerts raised by IDS.

In Section 5, we focused on the explanations for a single vector that represents a single edge in the graph, see Section 2.1.2 for details. In practice, the Sec2graph method represents a single network connection with several vectors (several edges). We hence define how to combine explanations produced from different vectors. The goal is to achieve the categorization of different attack types in the network connections.

Using the anomaly scores aggregated per network connection either of each category value or of each network feature, we show that we can perform clustering on them and group by attack type the different network connections that are identified as positives by the detector.

6.1 Clustering explanation outputs to identify network attack types

We perform clustering over the explanation outputs from *AE-pvalues*. We aim to unveil different attack types in the detected anomalies. The clustering results can help demonstrate the expressiveness of the explanations to capture differences among attack events. Thus, when a new security alert is generated, the security operator can obtain insights into the type of attack the alert belongs to by leveraging the previously labeled alerts and associated clusters. To this end, we constructed vectors by extracting the explanations and their corresponding anomaly scores from a network connection, as detailed in Section 6.2. These vectors have dimensions equal to the number of possible features, and their values represent the anomaly scores of each feature. We experiment with both the category value scores and the network feature scores. Each vector thus captures the abnormality scores of either the category values or the network features for a given network connection. To ensure sufficient representation of each attack type, we sampled 1000 network connections of each type. We used all the available attack types existing in the CICIDS2017. As presented in Table 1, some types of attacks, such as heartbleed, infiltration, and web attacks, are significantly more rarely witnessed than others. For the attack types with less than 1000 network connections, we used all the available network connections relative to this attack.

We employ an unsupervised hierarchical clustering algorithm to detect clusters among the vectors, where we specify the desired number of clusters to be formed. In this study, we aim to have 7 clusters corresponding to the major classes of attacks, including patator, Denial Of Service (DoS), heartbleed, portscan, web, infiltration, and botnet attacks. The algorithm then attempts to group together vectors that share similarities in the explanations of each network connection. The method constructs a hierarchical tree of clustering outcomes, starting from a single cluster encompassing all the vectors and then repeatedly partitioning it until the desired number of clusters is obtained. The complexity of the hierarchical algorithm is $O(n^3)$, where n is the number of network connections involved in the clustering. The clustering analysis was performed on a Linux machine with an Intel(R) Core(TM) i7 CPU processor (2.70GHz, 12 cores), and 32Gb of RAM. The execution took 17 seconds to cluster 2×9848 data points with respective dimensions $(389, 109)$ for (feature dimensions, network features) .

We proceed to calculate the Normalized Mutual Information (NMI) between the predicted clusters and the actual labels. In the case of a random strategy, the NMI obtained is 0.0011. However, when using the clustering method based on the anomaly scores of network features obtained from *AE-pvalues*, the NMI value of the clustering results significantly increases to 0.64. This is much better than the random guess baseline, as shown in Table 4.

The attack types captured by each of the seven clusters are illustrated in Figure 7. It can be observed that the SSH-patator attack and port scans are effectively captured in a single cluster, namely clusters 4 and 5, respectively. The network connections associated with the SlowHttpTest attack, which belong to cluster 5 are rejected network connection attempts that behave similarly to port scans. This explains why they are grouped together in the same cluster. However, for DoS attacks, they are divided among clusters 0, 2, and 6. The ftp-patator and botnet attacks are contained in cluster 1. Detailed information regarding the composition of each cluster is available in the appendix B.

	AE-pvalues NMI scores	**Random**
feature dimensions	**0.638**	0.0011
network features	0.554	

Table 4: Comparison of the Normalized Mutual Information scores using *AE-pvalues* to obtain explaining scores per feature dimension and per network feature and a *Random* strategy.

The results demonstrate the feasibility of categorizing network connections based on their associated attack types. The methods used to calculate the anomaly scores at the network connection level will be elaborated in Section 6.2.

6.2 Aggregating explanations of each network connection

We utilize the p-values obtained from the explanation list of each network feature vector to rank the feature dimensions and network features. The p-values indicate the abnormality of each feature dimension and their ranking order. Our proposed ranking scheme takes into account the global reconstruction score of each vector j, which is computed using BinaryCrossEntropy (BCE) as bce_j. We assign a higher weight to vectors with higher reconstruction error (higher bce). The p-value p_i of the i-th dimension indicates its abnormality; smaller p-values indicate a higher abnormality. As the variation of the bce_j score is the opposite of the variation of the p-value p_i, we use $(1 - p_{ij})$ which we call β scores to make the two quantities evolve in the same way. The score of the i-th dimension is the average of all the β scores, weighted by their respective bce, for all vectors related to the network connection. The formula for computing the score of the i-th dimension is given by Eq 3.

$$score\_dimension_i = \frac{1}{nb\_edges_i} \sum_{j=1}^{nb\_edges_i} bce'_j \times (1 - p_{ij}) \quad (3)$$

We also aim to aggregate the explanations at the network feature level and propose Eq 4 to achieve this. The basic principle is similar to the per feature dimension case described in Eq 3. However, since we have multiple category values for a single network feature, we needed to devise a strategy to combine the abnormality scores of the different category values into a single score. We decided to use the highest abnormality score among all the category values of the network feature. Therefore, we use the max function over the β scores for each category value that belongs to the network feature.

$$score\_feature_i = \frac{1}{nb\_edges_i} \sum_{j=1}^{nb\_edges_i} bce'_j \times \max_{l \in dim\_feat_i} (1 - p_{lj}) \quad (4)$$

In both Eq 3 and Eq 4, nb_edges_i is the number of edges that are compatible with the i-th network feature or category value and bce'_j is the bce_j score normalized as expressed in Eq 5.

$$bce'_j = \frac{bce_j}{\sum_{k=1}^{nb\_edges} bce_k} \quad (5)$$

6.3 Explaining alerts raised by a ML-based NIDS

To figure out if the explanations were relevant for each attack type, we took all the True Positive (TP) samples obtained with the Sec2graph approach. Like in the clustering section, we analyzed the explanations of 1000 network connections of each attack type. These network connections were obtained using the following sampling scheme: we took the first 300 network connections and the last 300 network connections, and then we sampled 400 network connections uniformly in the rest of the available network connections. We chose this approach to get connections related to the beginning, the middle, and the end of each attack. Indeed, the behavior of the attack may change over time, as shown afterward. We include a manual inspection of several attacks in CICIDS2017 and verify whether the explanations produced by *AE-pvalues* are consistent with the unveiled attack mechanisms in the network traffic data. In addition, we also performed the manual investigation for some False Positive (FP) samples.

In practice, a security operator is likely to be able to handle between three and five explanations per network connection. If we provide more than five explanations, the operator might be

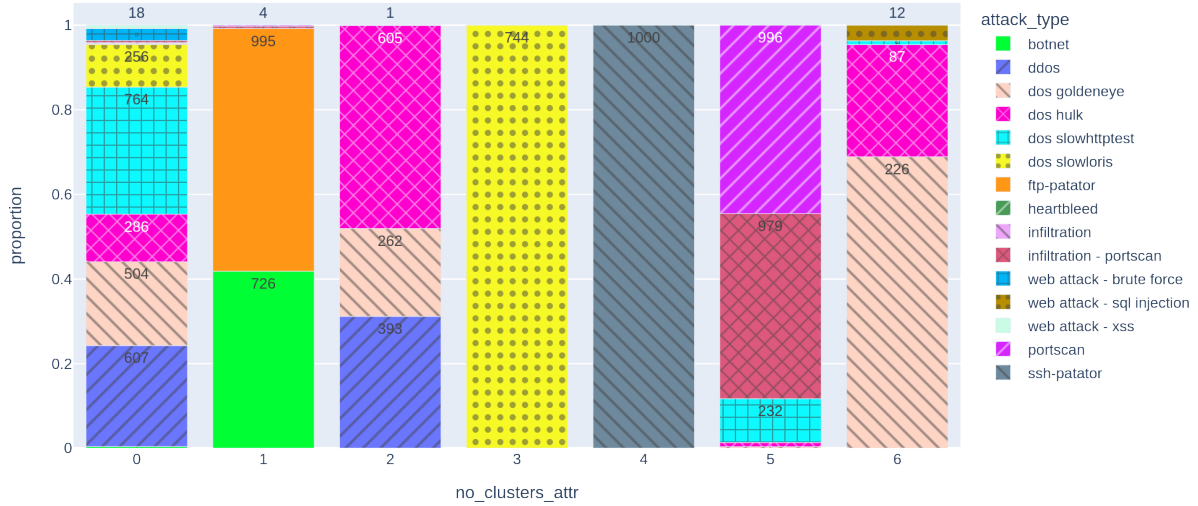


Figure 7: Proportion of each attack type in the different clusters

unable to analyze them thoroughly. If we provide less than three explanations, the explanations might not be discriminating enough to identify the attack type. In this section we decided to always work with the top-5 feature dimensions or network features.

The Figure 8 proposes a general overview by summarizing the features contributing by at least 10% in an attack's explanation. To compute these values, we collected the first five explaining network features for each network connection and measured the appearance frequency of these features for each attack type. The darker the color, the higher frequency has the feature for the corresponding attack. The maximum score per network feature is 20%, which means the network feature always occurs among the top-5 explanations. We can observe some common patterns based on the different attack types. For instance, web attacks (highlighted in green) are characterized by the user agent (browser and operating system). Several attacks are characterized by the *port_value*, which is involved in attacks such as the port scan and botnet attacks. SSH-related features are specific to the ssh-patator attack. From this high-level perspective, the explanations seem reasonable and consistent with the class of attack. In this section, we present a manual analysis of the DoS Slowloris, the brute force belonging to the web attacks, the SSH-patator, the botnet attack, and the DoS GoldenEye.

In the following, we use both network features and feature dimension explanations. Each feature dimension can be described as a tuple (feature, category value). For clarity, we write such a tuple as *feature/category*. For example, the explanation *ua_browser/Firefox* means that the value *Firefox* of the feature *ua_browser* is important to explain the abnormality of the network connection. As seen in Section 2.1.2, continuous features are categorized using GMM. Thus the name of these features ends with *_gmm*, and their associated category value is the number of the Gaussian in which the value falls. Again for clarity's sake, some very long category values are shortened with "[…]".

DoS Slowloris With this attack, the attacker seeks to consume all the processes or threads allocated by the web server to process queries. They do so by maintaining the TCP connections open for a very long time. To maintain these connections open, the tool does not end the HTTP header part of the request and regularly sends (before the server timeout) a new HTTP header. The most frequent explanations provided at the category values level are: *http_status_code/∅*, *http_info_code/∅*, *http_status_msg/other*, *http_info_msg/∅*, and *http_status_code/other*. They represent 55% of the samples, i.e., of 1000 network connections of this attack type. These explanations describe the situation when the server is down. Thus, it is not responding anymore. Therefore we can observe the missing status code and message since the server never replied before the network connection was stopped.

Then 40% of the network connections were explained by the following feature dimensions *ua_browser/Firefox*, *ua_os/Mac OS X*, *ua_browser/other*, *filetransfer_mime_type/image/gif*, and *filetransfer_mime_type/application/xml*. These explanations are related to the abnormal user-agent of the attacker. Indeed, the user-agent seen during this attack has never been used on the training day (the day without attacks). The Auto-Encoder detects this anomaly, and the explanation method correctly highlights the user-agent (containing the browser and the operating system).

Brute Force (web) With this attack, the attacker makes many authentication attempts on a web page with a machine under the Kali Linux operating system. Thus the attacker has a very discriminating user agent, and the detection model uses it to detect this attack. We find it again in the explanations. 96% of the 73 network connections associated with this attack are explained at the feature dimensions level by the following attributes: *ua_browser/other*, *ua_browser/Firefox*, *ua_os/Mac OS X*, *ua_browser/Safari* and *http_status_msg/Found*. The HTTP message "Found" is consistent with the fact that the attacker requests the `login.php` page to perform the brute force, and for each query, the server returns the code 302 and the message "Found". The explanations at the network feature level are *ua_browser*, *ua_os*, *http_status_msg*, *http_status_code*, *http_trans_depth*, are also valuable for the expert since they relate the fact that there are a large number of HTTP requests within the same network connection as shown in the *http_trans_depth*

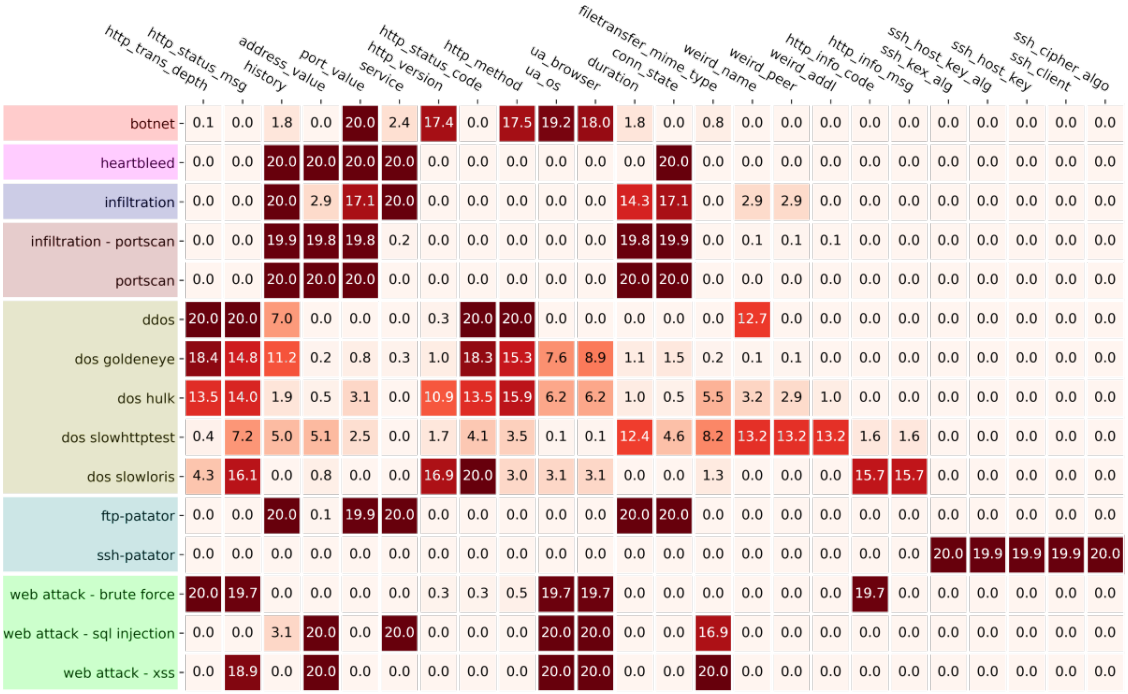


Figure 8: Visualisation of the explanations per attack type. The scores are the percentages indicating how often the features occur in the top-5 explanations for the respective attack type.

category. At the network feature level, we find the user-agent and the HTTP status code related features again.

SSH-Patator The attack consists in brute forcing an SSH server to try to obtain access to the server. The attacker uses a tool named Patator, developed in Python. That tool uses the paramiko library to establish the ssh connection and attempts some login password couples. Zeek logs four authentication attempts for each network connection. None of the attempts are successful.

99.6% of the alerts are explained by two groups of explanations. 77.5% of the alerts linked to SSH-Patator are explained by the following feature dimensions: *ssh_host_key/b5:61:[…], ssh_mac_alg/hmac-sha1, ssh_server/SSH-2.0-OpenSSH[…], ssh_compression_alg/none, ssh_host_key_alg/ssh-rsa.* Their corresponding network features are: *ssh_host_key, ssh_mac_alg, ssh_server, ssh_compression_alg, ssh_host_key_alg*. Then the rest (22.1%) of the alerts are explained by the following feature dimensions: *ssh_host_key/b5:61:[…], ssh_mac_alg/hmac-sha1, ssh_server/SSH-2.0-OpenSSH[…], ssh_host_key_alg/ssh-rsa* and *ssh_compression_alg/none*, and the associated network features explanations are *ssh_host_key, ssh_mac_alg, ssh_server, ssh_host_key_alg, and ssh_compression_alg*.

In fact, on the training day, only one SSH server is accessed, and the clients used to access the SSH server are all the same. It implies that all the attributes linked to the SSH security object are correlated. As the attacker uses the Patator tool, which uses the paramiko library, only two network features are different in the SSH security objects related to the attack: the *ssh_client* (linked to paramiko) and the *ssh_mac_alg* (certainly depending on the use of the paramiko library). The explanations support the fact that, for the detection model, the network features of the SSH security object are inconsistent.

Botnet. The bots are certainly preinstalled on five Windows machines of the experimental setup of CICIDS2017. The bots installed are related to an open-source botnet developed in Python named Ares[2]. The bots communicate with the C2 server by HTTP on the port 8080. They send regular beacons to the C2 and sometimes receive commands from the server, such as taking a screenshot of the desktop or listing a directory.

86.5% of the alerts are explained by the following feature dimensions: *duration_gmm/2, history/w, conn_state/RSTO, conn_state/SF, proto/udp*. The network features linked to those alerts are: *duration, history, conn_state, proto, orig_ip_bytes*.

The HTTP connections related to these alerts are linked to the beaconing, and they last less than 10ms. That is due to the fact that the C2 server responds to those beacons only with HTTP headers (no HTML page is sent by the server) and to the geolocation of the C2 server which is in the same setup as the other machines (only one router is between the computers infected by the botnet and the C2 server). The explanations point to the fact that the duration value should be higher and that, for very short-lived connections, it is abnormal to have a full HTTP connection on tcp (conn_state/RSTO, conn_state/SF, proto/udp). So the explanations are corroborated by the actual analysis.

[2]https://github.com/sweetsoftware/Ares

DoS GoldenEye. The GoldenEye tool[3] is a DoS tool that aims to take down a server by preventing the cache mechanism, maintaining connections with the keep-alive header (timeout option) and sending a large number of requests. In order to prevent easy filtering of the requests, all requests have a random argument with a random value; the referrer is also random, and the user-agent is taken randomly from a list of user agents. It must be noted that the Apache server attacked by that DoS does not care about the timeout value specified in the request and sets it to five seconds. After those five seconds, if the client sends no packets, the server closes the TCP connection.

For this attack, we observed two cases. For the first case, 25.9% of the alerts are explained by the following feature dimensions: *http_method/other, http_trans_depth_gmm/2, http_status_msg/other, http_status_code/304, http_trans_depth_gmm/1*. The network feature-level explanations related to those alerts are the following: *http_method, http_trans_depth, http_status_msg, http_status_code, history*.

Secondly, another case includes 16.5% of the alerts and the alerts are explained by the following feature dimensions: *http_status_msg/other, http_trans_depth_gmm/3, http_status_msg/Found, history/R, history/t*. The explanations at the network feature level related to those alerts are the following: *http_status_msg, http_trans_depth, history, conn_state, duration*.

The network feature *http_trans_depth* represents the number of HTTP requests sent in one TCP connection (thanks to the keep-alive mechanism). For all those alerts, only one request is sent, but the connection duration is around five seconds in the first case and around five minutes in the second case (due to a RST packet sent by the attacker five minutes after the FIN packet sent by the server). For those kinds of duration, more HTTP requests should have been sent in the TCP connection. The network feature *history* is also interesting because the network connections related to the attack are closed in an abnormal way (the attacker does not answer with a FIN/ACK packet to the FIN packet sent by the server and, in the second case, responds with a RST packet five minutes later). The explanations are, therefore, consistent with the expert analysis.

False Positives analysis. The Sec2Graph NIDS raised about 17,000 false positive alerts on the four days with attacks (containing almost 1,700,000 connections on the whole). Given the results of Table 4, the explanations for each dimension of every network feature produce a more fine-grained and informative understanding of the detection result. Therefore, we adopt this level of explanation to investigate the FP examples. The alerts can be grouped into 3,000 different explanation tuples. Each tuple corresponds to a particular case to be investigated. One such tuple is composed of the top 5 feature dimensions ranked by the proposed *AE-pvalues* method, like in the TP analysis, and they are considered to contribute the most to the corresponding alerts. We provide the analysis of three different FP alerts with the highest anomaly scores. For these samples, there is no ground truth regarding the explanations. We can only rely on the understanding of the network captures.

The following 5 feature dimensions are considered to explain the first FP example:*dcerpc_endpoint/other, dcerpc_operation/other, dcerpc_named_pipe/other, dcerpc_endpoint/lsarpc, dcerpc_named_pipe/135*. Given the explanations, this network connection was very likely to be related to using the DCE RPC protocol. In the corresponding network capture, we indeed observed the use of the DCE RPC protocol for the remote administration of the machine. We also note that, as suggested by the explanations, the endpoint, the named pipe, and some operations used are never seen in the benign training data of the AE. Moreover, this protocol is rarely used in the benign network traffics and represents only 0.05% of the network connections. The scarcity of the protocol and the unseen values may explain why this connection triggers the FP. A network administrator should know the mechanisms used for the administration of its network. In such a situation, he or she should know whether DCE RPC is legitimately used. Therefore from the provided explanations, he or she should be able to eliminate this FP alert quickly.

Another interesting FP sample among the most abnormal ones is a network connection with the following explanations: *port_value/other, port_value/80, port_value/443, history/S, conn_state/RSTR*. In this example, we can expect a problem related to the value of the port used. The history of the network connection seemed to be abnormal, as suggested by the SYN flag (*history/S*), and finally, the connection state flags also seemed to indicate an issue (RSTR: Responder sent a RST). When we looked at the network capture, the port value did not seem abnormal (the port value is 80). However, the two other explanations were very indicative. The network connection was observed at the beginning of Wednesday's capture. In fact, the beginning of the connection was missing, so it missed the SYN packet of the emitter at the beginning of the connection. At the end of this connection, the server ended the connection via the issuance of a RST packet. This connection corresponded to an update of a Windows client's malware signatures on Wednesday morning in the dataset. It was, therefore, not abnormal. But it was also reasonable to return an alert regarding an initiated connection without a complete TCP handshake and then aborted by the server. For this FP, even though the explanations well indicated how the low-level network event occurs, we can find that a manual inspection would be required to verify the true cause. The explanations give initial clues about the anomaly to the analyst and should accelerate the analysis.

For the third FP sample, the explanations include *ssh_version/other, ssh_server/other, ssh_kex_alg/other, ssh_cipher_algo/other, ssh_host_key/other*. We observe many ssh-related explanations. We can thus expect an issue with this ssh protocol. The network connection was initiated by a Windows client with the IP address 192.168.10.5 toward the Ubuntu web server with the IP address 192.168.10.50. The client did not launch any attack against the web server. However, in parallel to this connection, an SSH-Patator attack was ongoing on the web server. When we checked the fields of network features highlighted by the explanations, we observed that all of them except the *ssh_version* were left empty, which is never witnessed in benign traffics. These fields were empty because when the client initiated the connection, the server had no more available resources due to the attack. The server thus sent a FIN packet right after the TCP handshake. The client continued to issue data, but the server terminated the connection by issuing a RST packet. No information about the versions of cipher and key exchange algorithms was exchanged. This network connection was not malicious but had unusual behavior due to the ongoing attack. In the above paragraph on the SSH-Patator attack, we saw that the explanations on the true

[3]A version of the tool can be found here: https://github.com/jseidl/GoldenEye

positive alerts allowed the analyst to understand that the paramiko tool was used, revealing an ongoing ssh-related attack. Similarly, the explanations of this alert also highlighted ssh-related issues, so the analyst could quickly understand that it was a consequence of the attack.

Discussion. The explanations are very precise at the network level, pointing to the feature dimensions and the network features that differentiate the network connections of the attacks from normal network connections. However, they are generally too low-level to give an immediate insight to the analyst so that he can classify the alert without further analysis. The clustering of the alerts is a way to give this insight and ease the classification (see Section 6.1). However, since the clustering is unsupervised, clusters are not labeled and thus require analysts to have previously labeled some samples of the different clusters to infer which attack type it is.

The lack of high-level explanations is mainly due to four reasons: high correlations between some feature dimensions or some network features, many biases in the CICIDS2017 dataset, the fact that the detection model is only based on a single network connection and the fact that the explanations only point out a feature dimension or a network feature without telling whether the value should be present or absent. This last issue can be illustrated as follow. If an explanation is *history_R*, the analyst does not know if the detection model expected that the client sent an RST packet, but this event did not happen, or if it is the opposite and the explanation highlights the presence of such a history flag. We could easily overcome this issue by confronting the explanations at the feature dimension level with the actual category value existing in the vector. This information is more difficult to obtain at the network features level.

About the biases of the CICIDS2017 dataset, we discovered many of them by analyzing the explanations given by our method. Several works already tackled CICIDS2017 dataset quality and performed sometimes a manual analysis of the network captures such as [17], [22] or [15] however, they highlighted labeling issues, flow extraction issues, traffic capture issues, but none of them is highlighting the biases that exist in the dataset. To the best of our knowledge, the biases we identified are novel and were not previously reported. For example, we can see in Figure 8 that the alerts for the web attacks and the botnet attack are mainly explained by network features related to the user-agent. In fact, in the CICIDS2017 dataset, the user agents are the default ones specified in the tools used to implement those attacks. Script-kiddies would certainly keep those values, while a more advanced attacker would change them. Thanks to the explanations, we realized that for those attacks, the detection is more related to an artifact of the attacker's tool than the network behavior of the attack type. The experimental setup used to generate the dataset also affects network features such as the duration of connections (for the botnet attack, the C2 server is in the experimental setup while all the other web servers are on the Internet). In the false positive analysis, we even saw how cutting network captures can artificially make benign connections have abnormal behavior. These examples point out an issue shared by several security datasets: the evaluated detectors can learn to correctly identify attacks by relying on experimental artifacts, leading to a potential overestimation of the detection performances of these detectors on such datasets without being relevant in practice.

7 CONCLUSION

In this article, we proposed *AE-pvalues*, a new unsupervised method based on p-values to provide explanations that could be used on top of any AE-based anomaly detection method. We compared this new method to existing ones and showed that it performs better at finding the right feature responsible for the anomaly. It also obtained better performances when it comes to clustering the network connections by attack type based on the explanations. The method is scalable which was not the case for *SHAP_AE*. Besides, we used this p-value-based method to analyze practical attacks of the CICIDS2017 dataset and show its relevance on real data. On this occasion, we could observe the limitation of the dataset regarding the quality of the attacks. They are often not stealthy enough, and thanks to the explanations, we were able to highlight that the detector often relies on easy clues to detect them and not especially on the "real" malicious behavior. This is not an issue due to the learning of the ML model but a problem related to data quality. The explanations are an excellent tool to improve detection by checking the proper functioning of the (N)IDS. Lastly, we also used the anomaly scores associated with the explanations provided by our explaining method to cluster the network connections by the attack type they belong to and obtained good clustering results.

In our study, we did not discuss the role of evading techniques in attacker behaviors in detecting anomalies. Our explanation technique is positioned as a post-hoc method. Therefore, it will act as a lens to help human experts understand the logic triggering the detection. It nevertheless does not change the results of detection. That said, the explanations produced by our method could be used by adversaries to evade the ML-driven IDS because they highlight the network characteristics on which the IDS is relying for the detection. The adversaries may change these attributes to avoid detection. Beyond that, the defenders/practitioners of ML-driven IDS may use the explanations to predict potential evading efforts and take proactive actions to harden the detection system.

All these contributions are aimed at facilitating the forensic investigation and the alert analysis done by security operators. As a future work, we plan to leverage the concept of p-values not only to provide explanations but also for the detection step. In addition, it would be interesting to investigate how the explanations could automatically help reduce FP. We believe that the explanations provided by our AE-pvalue method can be used to triage the different alerts and identify FP. First of all, we have demonstrated in Section 6.1 that the explanations can be used to perform clustering of different attack behaviors and offer a good clustering accuracy. Second, we can combine the clustering results and a few labels of attack types provided by the human analysts' manual investigations. We can then propagate the attack class memberships across the clustering results based on semi-supervised learning techniques, e.g., label propagation [20] or even supervised learning methods, which provide the classification confidence of each network traffic data. The estimated confidence can help differentiate TP from FP. For example, FP tend to locate in the ambiguous area close to the classification boundary separating benign and attack data. Therefore, FP may have low classification confidence in the label propagation process, which could be used as an indicator to identify them.

REFERENCES

[1] Amina Adadi and Mohammed Berrada. 2018. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access* 6 (2018), 52138–52160. https://doi.org/10.1109/ACCESS.2018.2870052

[2] Diana Laura Aguilar, Miguel Angel Medina-Pérez, Octavio Loyola-Gonzalez, Kim-Kwang Raymond Choo, and Edoardo Bucheli-Susarrey. 2022. Towards an interpretable autoencoder: A decision-tree-based autoencoder and its application in anomaly detection. *IEEE transactions on dependable and secure computing* 20, 2 (2022), 1048–1059.

[3] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, and Farhan Ahmad. 2021. Network Intrusion Detection System: A Systematic Study of Machine Learning and Deep Learning Approaches. *Trans. Emerg. Telecommun. Technol.* 32, 1 (jan 2021), 30 pages. https://doi.org/10.1002/ett.4150

[4] Giuseppina Andresini, Annalisa Appice, Francesco Paolo Caforio, Donato Malerba, and Gennaro Vessio. 2022. ROULETTE: A neural attention multi-output model for explainable network intrusion detection. *Expert Systems with Applications* 201 (2022), 117144.

[5] Liat Antwarg, Ronnie Mindlin Miller, Bracha Shapira, and Lior Rokach. 2022. Explaining Anomalies Detected by Autoencoders Using Shapley Additive Explanations. *Expert Syst. Appl.* 186, C (dec 2022), 14 pages. https://doi.org/10.1016/j.eswa.2021.115736

[6] Ons Aouedi, Kandaraj Piamrat, and Dhruvjyoti Bagadthey. 2020. A Semi-supervised Stacked Autoencoder Approach for Network Traffic Classification. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*. 1–6. https://doi.org/10.1109/ICNP49622.2020.9259390

[7] J. E. Campbell, G. R. Carmichael, T. Chai, M. Mena-Carrasco, Y. Tang, D. R. Blake, N. J. Blake, S. A. Vay, G. J. Collatz, I. Baker, J. A. Berry, S. A. Montzka, C. Sweeney, J. L. Schnoor, and C. O. Stanier. 2008. Photosynthetic Control of Atmospheric Carbonyl Sulfide during the Growing Season. *Science* 322, 5904 (2008), 1085–1088. http://www.jstor.org/stable/20145274

[8] Fabien Charmet, Harry Chandra Tanuwidjaja, Solayman Ayoubi, Pierre-François Gimenez, Yufei Han, Houda Jmila, Grégory Blanc, Takeshi Takahashi, and Zonghua Zhang. 2022. Explainable artificial intelligence for cybersecurity: a literature survey. *Annals of Telecommunications* 77 (2022), 789 – 812.

[9] Koby Crammer and Gal Chechik. 2004. A Needle in a Haystack: Local One-Class Optimization. In *Proceedings of the Twenty-First International Conference on Machine Learning* (Banff, Alberta, Canada) *(ICML '04)*. Association for Computing Machinery, New York, NY, USA, 26. https://doi.org/10.1145/1015330.1015399

[10] Thijs van Ede, Hojjat Aghakhani, Noah Spahn, Riccardo Bortolameotti, Marco Cova, Andrea Continella, Maarten van Steen, Andreas Peter, Christopher Kruegel, and Giovanni Vigna. 2022. DEEPCASE: Semi-Supervised Contextual Analysis of Security Events. In *2022 IEEE Symposium on Security and Privacy (SP)*. 522–539. https://doi.org/10.1109/SP46214.2022.9833671

[11] Swastik Haldar, Philips George John, and Diptikalyan Saha. 2021. Reliable counterfactual explanations for autoencoder based anomalies. In *Proceedings of the 3rd ACM India Joint International Conference on Data Science & Management of Data (8th ACM IKDD CODS & 26th COMAD)*. 83–91.

[12] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. *Proceedings 2019 Network and Distributed System Security Symposium* (2019).

[13] Nguyen Xuan Hoang, Nguyen Viet Hoang, Nguyen Huu Du, Truong Thu Huong, Kim Phuc Tran, et al. 2022. Explainable anomaly detection for industrial control system cybersecurity. *IFAC-PapersOnLine* 55, 10 (2022), 1183–1188.

[14] Pang Wei Koh and Percy Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1885–1894. https://proceedings.mlr.press/v70/koh17a.html

[15] Maxime Lanvin, Pierre-François Gimenez, Yufei Han, Frédéric Majorczyk, Ludovic Mé, and Éric Totel. 2023. Errors in the CICIDS2017 Dataset and the Significant Differences in Detection Performances It Makes. In *Risks and Security of Internet and Systems*, Slim Kallel, Mohamed Jmaiel, Mohammad Zulkernine, Ahmed Hadj Kacem, Frédéric Cuppens, and Nora Cuppens (Eds.). Springer Nature Switzerland, Cham, 18–33.

[16] Laetitia Leichtnam, Eric Totel, Nicolas Prigent, and Ludovic Mé. 2020. Sec2graph: Network Attack Detection Based on Novelty Detection on Graph Structured Data. In *DIMVA*. 238–258.

[17] Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, and Wouter Joosen. 2022. Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018. In *2022 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 254–262.

[18] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *NIPS*.

[19] Quoc Phong Nguyen, Kar Wai Lim, Dinil Mon Divakaran, Kian Hsiang Low, and Mun Choon Chan. 2019. GEE: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection. In *2019 IEEE Conference on Communications and Network Security (CNS)*. 91–99. https://doi.org/10.1109/CNS.2019.8802833

[20] Rattana Pukdee, Dylan Sam, Maria-Florina Balcan, and Pradeep Ravikumar. 2023. Label Propagation with Weak Supervision. arXiv:2210.03594 [cs.LG]

[21] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1135–1144. https://doi.org/10.1145/2939672.2939778

[22] Arnaud Rosay, Eloïse Cheval, Florent Carlier, and Pascal Leroux. 2022. Network Intrusion Detection: A Comprehensive Analysis of CIC-IDS2017. In *International Conference on Information Systems Security and Privacy*.

[23] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *ICISSP*.

[24] Véronne Yepmo, Grégory Smits, and Olivier Pivert. 2022. Anomaly explanation: A review. *Data & Knowledge Engineering* 137 (2022), 101946.

A SEC2GRAPH MODEL

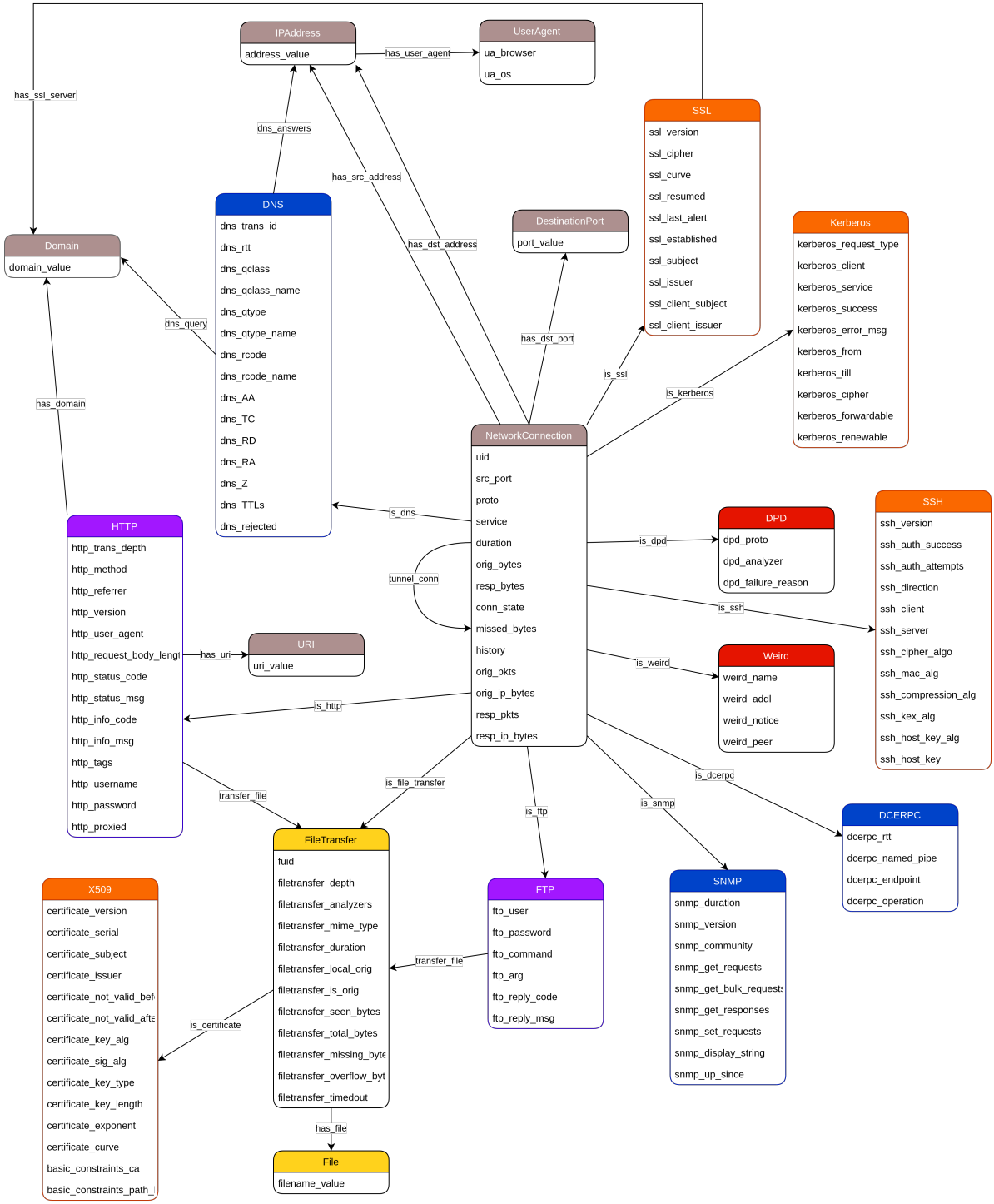


Figure 9: Sec2graph: security object graph definition

B CLUSTERING RESULTS DETAILS

no_clusters_attr	attack_type	counts	cluster_proportion (%)
0	dos slowhttptest	764	30.0
0	ddos	607	23.9
0	dos goldeneye	504	19.8
0	dos hulk	286	11.2
0	dos slowloris	256	10.1
0	web attack - brute force	73	2.9
0	web attack - xss	18	0.7
0	infiltration - portscan	12	0.5
0	botnet	10	0.4
0	ftp-patator	5	0.2
0	infiltration	6	0.2
0	heartbleed	1	0.0
0	web attack - sql injection	1	0.0
1	ftp-patator	995	57.3
1	botnet	726	41.8
1	infiltration - portscan	9	0.5
1	portscan	4	0.2
1	infiltration	1	0.1
2	dos hulk	605	48.0
2	ddos	393	31.2
2	dos goldeneye	262	20.8
2	dos slowhttptest	1	0.1
3	dos slowloris	744	100.0
4	ssh-patator	1000	100.0
5	portscan	996	44.5
5	infiltration - portscan	979	43.8
5	dos slowhttptest	232	10.4
5	dos hulk	22	1.0
5	dos goldeneye	8	0.4
6	dos goldeneye	226	68.9
6	dos hulk	87	26.5
6	web attack - sql injection	12	3.7
6	dos slowhttptest	3	0.9

Table 5: Details of the cluster content.

TADAM: Learning Timed Automata from Noisy Observations

Lénaïg Cornanguer[1], Pierre-François Gimenez[2]

Abstract

Timed Automata (TA) are formal models capable of representing regular languages with timing constraints, making them well-suited for modeling systems where behavior is driven by events occurring over time. Most existing work on TA learning relies on active learning, where access to a teacher is assumed to answer membership queries and provide counterexamples. While this framework offers strong theoretical guarantees, it is impractical for many real-world applications where such a teacher is unavailable. In contrast, passive learning approaches aim to infer TA solely from sequences accepted by the target automaton. However, current methods struggle to handle noise in the data, such as symbol omissions, insertions, or permutations, often resulting in excessively large and inaccurate automata. In this paper, we introduce TADAM, a novel approach that leverages the Minimum Description Length (MDL) principle to balance model complexity and data fit, allowing it to distinguish between meaningful patterns and noise. We show that TADAM is significantly more robust to noisy data than existing techniques, less prone to overfitting, and produces concise models that can be manually audited. We further demonstrate its practical utility through experiments on real-world tasks, such as network flow classification and anomaly detection.

1 Introduction

The inference of system behavior models from observational data allows gaining insight into complex systems without requiring expert knowledge and a time consuming process. Such models can then be used to automatize various tasks such as monitoring, diagnostics, or scheduling [15, 1]. Timing of events is particularly crucial in many domains, including real-time systems and communication protocols. To capture this aspect, these systems can be modeled using timed automata (TA) [2], a well-established formalism for event sequence data that includes temporal information. For instance, Fig. 1 illustrates an automaton modeling in a simplified way the establishment of a TCP network connection between two devices.

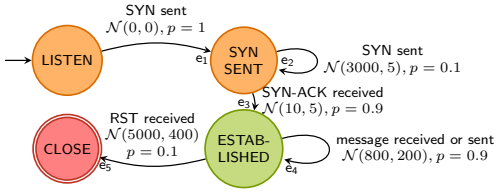


Figure 1: A timed automaton extended with probabilistic parameters as model of a simplified TCP connection.

The current state of the system can be mapped to a *location* in the automaton (LISTEN, SYN SENT, etc.) at a given time. The state transitions triggered by an event are modeled by *transitions* between locations that are labeled with a corresponding *symbol* (SYN sent, SYN-ACK received, etc.). A *guard* on each transition describes the temporal delay associated with an event. For example, SYN messages can be regularly sent from the sender until SYN-ACK message is received, triggering the transition from the location SYN SENT to ESTABLISHED. Such sequences of events occurring during an execution of the system are called *words*.

Several methods have been proposed to learn timed automata from observational data. However, the issue of noise in the data remains unaddressed, despite the fact that available data are often incomplete and noisy, making it harder to infer accurate models. Noise can arise from various sources: limited measurement accuracy, probe configuration errors [8], incorrect labelling, etc. The current methods are therefore unsuited for a large proportion of real-world applications.

A second challenge in inferring timed automata is to find the good level of generalization between two extremes: an automaton with a single location and only self-loop transitions which accepts every possible words, and an automaton with one branch per event sequence which only accepts the learning data.

To address both these challenges, we present an approach based on the Minimum Description Length (MDL) principle [7], a model selection criterion that favors simple models that most efficiently compress the data. By balancing complexity and goodness of fit, the

[1]CISPA Helmholtz Center for Information Security
lenaig.cornanguer@cispa.de
[2]CentraleSupélec, Inria, Univ. Rennes, IRISA
pierre-francois.gimenez@centralesupelec.fr
Both authors contributed equally.

Accepted at SIAM SDM 2025

MDL principle is particularly relevant for noisy data as it discourages overly complex models that might capture random noise rather than true underlying patterns.

This paper presents the following contributions:

(1) An MDL encoding for timed automata (Sec. 4.2);

(2) A word correction strategy for data encoding (Sec. 4.3);

(3) TADAM, the first algorithm to learn TA from noisy data (Sec. 5).

2 Related Work

The vast majority of timed automata learning literature relies on active learning, with access to a teacher that can provide counterexample to an equivalence query or the answer to the membership query of one word [3, 17].

Learning TA solely from observational data (passive learning) has received much less attention due to the complexity of the absence of counter-examples to guide the process. Cornanguer et al. [4] have proposed the TAG algorithm that controls the model generalization level by factorizing the TA on sub-sequence of events of fixed length. Verwer et al. [16] have proposed the RTI+ algorithm which transforms the learned automaton by performing operations that improve the likelihood of the data with the model. Yet, these algorithms are not designed to be robust to noise in the data.

In the broader literature on model inference without temporal consideration, various strategies have been proposed to tackle the challenge of noise. Wiegand, Klakow, and Dietrich [19] proposed a greedy MDL-based approach to learn an event pattern graph. To handle noise, they compute *covers* to indicate how to read the data with a given pattern graph. There is no notion of time and the graph edges are not weighted, making the search problem simpler. Wallner, Aichernig, and Burghard [18] learn Moore machines using SAT solving by looking for the deterministic model that is consistent with the largest proportion of the input sample. In addition to have no consideration of time, their algorithm requires to specify the number of state of the model, and there are no clear model selection criterion, only guidelines.

3 Preliminaries

We first present our modeling formalism based on timed automata, and then introduce the MDL principle.

3.1 Timed Automata Timed automata (TA) [2] are finite-state automata extended with temporal parameters that can model systems governed not only by the occurrence of events but also by time. In this paper, we consider a sub-class of TA, called Real-Time Automata (RTA) [5], with a single type of temporal parameter: the delay between two events.

Definition 1. (Real-Time Automaton) *A real-time automaton is a tuple* $\mathcal{A} = (\mathcal{Q}, \Sigma, \mathcal{E}, q_0, \mathcal{F})$ *where* $\mathcal{Q}$ *is a finite set of locations,* Σ *is a finite set of symbols (or events) called alphabet,* $\mathcal{E}$ *is a finite set of transitions,* $q_0 \in \mathcal{Q}$ *is the initial location, and* $\mathcal{F}$ *is a finite set of accepting locations.*
$\mathcal{E} \subseteq \mathcal{Q} \times \Sigma \times \mathbb{N}^2 \times \mathcal{Q}$ *is a finite set of transitions of the form* (q, a, g, q') *where* q *and* q' *are respectively the source location and destination location,* $a \in \Sigma$ *is a symbol,* g *is a guard in the form of an interval constraining the delay.*

Motivated by the noisy learning environment setup, we introduce a probabilistic variation of the RTA denoted pRTA. A probabilistic transition has a probability to be triggered from its source location and the interval guard is replaced by a probability distribution. The TA displayed in Fig. 1 is a pRTA.

Definition 2. (Probabilistic transition) *A probabilistic transition* $e = (q, a, G, p, q')$ *is a transition where* G *is a normal distribution* $\mathcal{N}(\mu, \sigma^2)$ *and* p *is the probability* $p(e|q)$ *of the transition from* q.

We now present the concepts of timed words (informally referred to as event sequences in the introduction) and timed automata language.

Definition 3. (Timed word) *A timed word* $w = (a_0, t_0)(a_1, t_1)... \in (\Sigma \times \mathbb{N})^*$ *is a finite sequence of symbols and non-decreasing timestamps.*
In the context of pRTA, we can re-write a timed word as a sequence of symbol and delay $w = (a_0, d_0)(a_1, d_1)...$ *where* $d_0 = 0$ *and* $d_i = t_i - t_{i-1}$.

A (timed) word w is consistent with an automaton $\mathcal{A}$ (or accepted by it) if there exists a *path* from the initial location to an accepting location, i.e., a sequence of transitions $q_0 \xrightarrow{e_0} q_1 \xrightarrow{e_1} ... \xrightarrow{e_p} q_f$. For a timed automaton $\mathcal{A}$, the *(timed) language* denotes the set of timed words accepted by $\mathcal{A}$. In a RTA, all words in its language have the same importance, while in a pRTA, the probability of occurrence of the timed words is no longer uniform thanks to the probabilistic transitions.

On a final note, to ease the computations, we consider a unique ending symbol to identify the end of the words. In the automaton, the transitions labeled by this ending symbol all lead to a unique final location q_f.

3.2 The MDL Principle The Minimum Description Length (MDL) principle [13, 7] is a model selection criterion that states that the best description for

some observed data is the shortest one. This principle is rooted in algorithmic information theory, where the amount of information conveyed by data or its description can be quantified in bits. In this work, we focus on the two-part MDL, where, among a model class $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one minimizing $L(M) + L(\mathcal{D}|M)$, with $L(M)$ the length of model M in bits and $L(\mathcal{D}|M)$ the length of data $\mathcal{D}$ encoded with this model M. This description in two-part allows us to balance model complexity with the accuracy of data representation. To apply the MDL principle, an encoding for the data and the model must be defined.

4 MDL for Timed Automata Learning

This section presents how to leverage MDL to learn pRTA from noisy timed words.

4.1 Noise Model The noise in a timed word can take different forms depending on the application. We consider four different noise types, illustrated with an initial timed word $w = (a_0, d_0)(a_1, d_1)(a_2, d_2)$: deletion ($w = (a_0, d_0)(a_2, d_2)$), insertion ($w = (a_0, d_0)(a_3, d_3)(a_1, d_1)(a_2, d_2)$), transposition ($w = (a_0, d_0)(a_2, d_2)(a_1, d_1)$), and symbol repetition ($w = (a_0, d_0)(a_1, d_1)(a_1, d_3)(a_2, d_2)$) (note that we allow the repeated symbol to be associated with a different delay). TADAM can also be parameterised with a subset of noise types depending on the domain's specificity.

4.2 Model Encoding We define an MDL encoding for the pRTA model. The more complex the pRTA, the more expensive it should be to encode. The description length of a pRTA $\mathcal{A}$ corresponds to the sum of the cost of encoding its locations, its alphabet, for each transition the source and destination locations, the symbol, the guards' normal distributions parameters, and its initial and accepting locations, and is given by

$$L(\mathcal{A}) = L_{\mathbb{N}}(|\mathcal{Q}|) + L_{\mathbb{N}}(|\Sigma|) + \sum_{e \in \mathcal{E}} \Big(2\log_2(|\mathcal{Q}|) + \log_2(|\Sigma|) + L_{\mathbb{N}}(\lfloor \mu_e \rfloor) + L_{\mathbb{N}}(\lfloor \sigma_e^2 \rfloor) \Big) + 2\log_2(|\mathcal{Q}|) \,,$$

where $L_{\mathbb{N}}$ is the MDL-optimal encoding for integers defined by [14].

4.3 Data Encoding We now define how to encode the data $\mathcal{D}$ given an automaton $\mathcal{A}$. Due to the presence of noise, many words in $\mathcal{D}$ might be not accepted by the automaton even if $\mathcal{A}$ is equivalent to the true generating process of $\mathcal{D}$. For this reason, we do not use the classical data encoding that relies on the word probability [7], because all the noisy words rejected by $\mathcal{A}$ would have a null probability and would have to be encoded explicitly. Instead, we integrate into the encoding the notion that such words could be accepted by $\mathcal{A}$ with some light changes. To do so, we associate the timed words with an *edit operation sequence* that describes how to read and correct them such that they are accepted by the pRTA. We propose several edit operations that mirror the noise types presented in Sec. 4.1:

- Add (ad): add a pair (a, d) to the word;
- Skip (sk): ignore a pair (a, d);
- Transpose (tr): permute two consecutive pairs;
- Deduplicate (de): remove a pair (a, d) following a pair with an identical symbol
- Follow (fo): read the pair (a, d) as it is.

Note that the follow edit operation allows us to handle the already accepted words in the same way as the words needing to be corrected.

More formally, an edit operation sequence R is a sequence of tuple in $\{add, skip, transpose, deduplicate, follow\} \times \mathcal{E} \cup \{\epsilon\} \times \mathbb{N}$, with ϵ a dummy transition that is not in $\mathcal{E}$, used when no transtition is to be associated.

For example, given the automaton in Fig. 1, the timed word (SYN sent, 0),(ACK sent, 500),(SYN sent, 2000),(RST received, 6500) is not recognized. We can associate it with the following edit operation sequence R: (follow, e_1, 0), (skip, ϵ, 500), (follow, e_2, 2000), (add, e_3, 10), (follow, e_5, 6500), resulting in the corrected timed word (SYN sent, 0),(SYN sent, 2000),(SYN-ACK received, 10),(RST received, 6500) that is accepted. Note that there can be multiple edit operation sequences possible for a same word. We propose in Sec. 5.1 an algorithm to find the edit operation sequence associated with a timed word that minimizes its encoding cost.

Let us now define the data encoding based on this notion. We denote $\mathcal{D}_r$ the subset of words in $\mathcal{D}$ that are accepted by a pRTA $\mathcal{A}$ or for which there exists an edit operation sequence R that results in a word accepted by $\mathcal{A}$. Encoding $\mathcal{D}_r$ with $\mathcal{A}$ consists in encoding R, the flattened edit operation sequence over all words. Let R_O with $O \subseteq \{\text{fo}, \text{sk}, \text{de}, \text{tr}, \text{ad}\}$ denotes a subsequence of R consisting of all tuples (o, e, d) where $o \in O$. We define the description length of $\mathcal{D}_r$ given $\mathcal{A}$ as

$$\begin{aligned} L(\mathcal{D}_r|\mathcal{A}) = & \sum_{(o,e,d) \in R} -\log_2(p(o)) \\ & + \sum_{(o,e,d) \in R_{\{\text{tr,fo,ad}\}}} -\log_2 p(e \mid q_s(e)) \\ & + \sum_{(o,e,d) \in R_{\{\text{tr,fo,de}\}}} -\log_2 p(d \mid e) \\ & + \sum_{(o,e,d) \in R_{\{\text{sk}\}}} (L_{\mathbb{N}}(d) + \log_2(|\Sigma|)) \end{aligned}$$

where $q_s(e)$ is the source location of e and $\log_2(p(o))$ is a fixed cost for each edit operation. The second sum term corresponds to the probability of the triggered transition given its source location, the third sum term corresponds to the probability of the delay given the triggered edge, and the fourth sum term corresponds to the explicit encoding of the skipped pairs using an uniform distribution to encode the symbol and a universal encoding for the delay. Remark that we only encode information necessary to retrieve the original word: for example, there is no need to encode the delay of a pair introduced by the "add" operation.

This formula can be partially rewritten using the notion of entropy, which will be useful in a following section. To this end, we introduce E, the random variable denoting which edge is triggered. We obtain

$$\sum_{\substack{(o,e,d)\in \\ R_{\{\mathrm{tr,fo,ad}\}}}} -\log(p(e \mid q_s(e))) = |R_{\{\mathrm{tr,fo,ad}\}}| \cdot H(E \mid q_s(E)) \ .$$

Similarly, we can show that

$$\sum_{(o,e,d)\in R_{\{\mathrm{tr,fo,de}\}}} -\log(p(d \mid e)) = |R_{\{\mathrm{tr,fo,de}\}}| \cdot H(D \mid E) \ ,$$

where D denotes the random variable of the delay, and that encoding the edit operation corresponds to

$$\sum_{(o,e,d)\in R} -\log_2(p(o)) = |R| \cdot H(O) \ .$$

As there may also exist timed words that cannot be associated with any edit operation sequence, we must also define an encoding for the uncorrectable words $\mathcal{D}_u \in \mathcal{D}$. This situation can happen when the noise model is narrowed and some of the edit operations are forbidden. Similarly as in the case of the skip operation, we define the description length of $\mathcal{D}_u$ given $\mathcal{A}$ as

$$L(\mathcal{D}_u|\mathcal{A}) = \sum_{(s,d)\in\mathcal{D}_u} (L_{\mathbb{N}}(d) + log_2(|\Sigma|)) \ .$$

Besides, we need to indicate whether a timed word is encoded as corrected or uncorrectable. We use a random variable X_r denoting to which set, $\mathcal{D}_r$ or $\mathcal{D}_u$, a timed word belongs to. The resulting description length of $\mathcal{D}$ given $\mathcal{A}$ is given by

$$L(\mathcal{D}|\mathcal{A}) = L(\mathcal{D}_u|\mathcal{A}) + L(\mathcal{D}_r|\mathcal{A}) + |\mathcal{D}| \cdot H(X_r) \ .$$

5 Algorithm

This section first describes the word correction algorithm and then the actual TA learning algorithm.

5.1 Word Correction Algorithm As explained in Sec. 4, our data encoding relies in some part on a word correction strategy that we present here. Our goal is to find the sequence of edit operations that leads to the minimal description length of the words corrected to be accepted by $\mathcal{A}$. To this end, we adapt the algorithm proposed by Oommen and Loke [11] to compute the optimal string alignment distance between two words to the problem of aligning a timed word with a pRTA (i.e., to the words from its language). Since the main novelty lies in the cost function, we focus on its description here.

Let $\mathcal{A}$ be a pRTA and $w = (w_0, w_1, \ldots, w_k)$ a timed word with $w_l = (a_l, d_l)$. Let $p(w, q, q')$ be the probability of a (sub-)timed word from location q to q', or 0 if no such path exist. We initialize the cost function as $cost(0, q) = \min_{w\in(\Sigma\times\mathbb{N})^*} -\log_2(p(w, q_0, q)) - |w| \cdot \log_2(p(\mathrm{ad}))\ \forall q \in \mathcal{Q}$. It indicates, for empty timed words, the most probable (i.e., with minimal cost) sequence of pairs (a, d) to add to reach q from q_0. Let us define an intermediate cost $cost'(l+1, q)$ to consider the different edit operation, given by the minimum of

$$\begin{cases} cost(l, q') - \log_2(p(\mathrm{fo})) - \log_2(p((w_{l+1}), q', q) \text{ if} \\ \quad \text{transition } q' \to q \text{ is labelled with } s_{l+1} \textit{ (follow)} \\ cost(l, q) - \log_2(p(\mathrm{de})) - \log_2(p(d_{l+1} \mid e_l)) \text{ if} \\ \quad s_{l+1} = s_l, \text{ where } e_l \text{ is the edge triggered by } w_l \\ \quad \textit{(deduplicate)} \\ cost(l-1, q') - \log_2(p(\mathrm{tr})) \\ \quad + \min_{q',q''\in\mathcal{Q}} \big(-\log_2(p((w_{l+1}), q', q'') \\ \quad - \log_2(p((w_l), q'', q))\big) \textit{ (transpose)} \\ cost(l, q) + L_{\mathbb{N}}(d_{l+1}) + \log_2(|\Sigma|) \textit{ (skip)} \end{cases}$$

The cost of reaching a location q from position $l+1$ in the word is given by $cost(l+1, q) = \min_{q'\in\mathcal{Q}}(cost'(l+1, q) + \min_{w\in(\Sigma\times\mathbb{N})^*} -\log_2(p(w, q, q')) - |w| \cdot \log_2(p(\mathrm{ad})))$, i.e., the minimal cost of reaching some intermediate location q' and then to reach location q through additions.

5.2 Transformation Operations Given $\mathcal{D}$, the learning process aims to find the automaton $\mathcal{A}^* \in$ pRTA such that $\mathcal{A}^* = \arg\min_{\mathcal{A}\in\mathrm{pRTA}} L(\mathcal{A}) + L(\mathcal{D}|\mathcal{A})$. Enumerating all possible automata where each transition is used by at least one word $w \in \mathcal{D}$ is impractical, and allowing some parts of $\mathcal{D}$ to remain unrecognized by the automaton further expands the search space. Therefore, we opt for a greedy algorithm that iteratively transforms the automaton to reduce the MDL cost. We define multiple transformations that can be applied to a given pRTA $\mathcal{A}$. To lighten the notations, we write $\mathcal{E}_{l,out}$ to refer to the outgoing transitions of location l, i.e., the set of transitions $\{(q, a, G, p, q') \in \mathcal{E} | q = l\}$, and similarly $\mathcal{E}_{l,in}$ to refer to the incoming transitions of l.

5.2.1 Location Merging The location merge simplifies $\mathcal{A}$ by merging two locations. The merge of q and q' in $\mathcal{A}$ results in their replacement in $\mathcal{Q}$ by q'' with $\mathcal{E}_{q'',out} = \mathcal{E}_{q,out} \cup \mathcal{E}_{q',out}$ and $\mathcal{E}_{q'',in} = \mathcal{E}_{q,in} \cup \mathcal{E}_{q',in}$.

5.2.2 Subpart Deletion The subpart deletion reduces the size of $\mathcal{A}$ and the timed language it accepts. Let $e = (q, a, G, p, q')$ be a probabilistic transition in $\mathcal{A}$. Deleting e results in $\mathcal{E} \leftarrow \mathcal{E} \setminus \{e\}$ and in the deletion of the subpart of A that is no more reachable from q_0, or from which q_f is no more reachable.

5.2.3 Location Split The location split replaces a location $q \neq q_0$ by a set of new locations Q_{new}, changes the destination of the transitions in $\mathcal{E}_{q,in}$ to locations in Q_{new}, and recreates the transitions in $\mathcal{E}_{q,out}$ with a location in Q_{new} as source whenever it is needed for a timed word, as shown in Fig. 2. This transformation is especially interesting with our MDL-based score because it affects this score very locally, meaning that despite the high number of new possible partitions P (exponential in $\mathcal{E}_{q,in}$), it is straightforward to estimate the gain or loss induced by it. We present in the following how to estimate the gain given a new partition P, and then how to find a good partition P for a given location q.

We start by showing how we can estimate the MDL data gain brought from a split. Let $\mathcal{A}_1$ be a pRTA and $\mathcal{A}_2$ the pRTA resulting from the split of q in $\mathcal{A}_1$. Let $f : \mathcal{E} \to \mathcal{Q}$ be the destination assignment function that takes a transition from $\mathcal{A}_1$ and return its destination location in $\mathcal{A}_2$. This function allows us to map the transitions in $\mathcal{E}_{q,in}$ to their new destination in Q_{new} after the split. To make our notation clearer in the following, we provide an example in Fig. 2, but the explanation holds for any $e_i \in \mathcal{E}_{q,in}$ (resp. $e_j \in \mathcal{E}_{q,out}$) with source location q' (resp. destination location q''). By construction, the transitions $q' \xrightarrow{e_i} q \xrightarrow{e_j} q''$ are

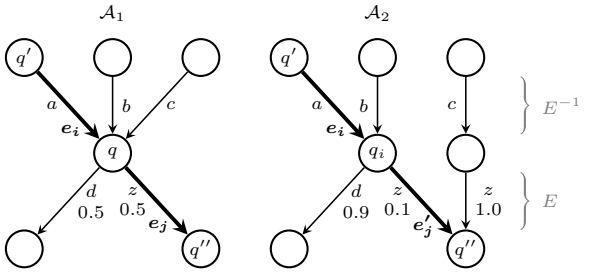


Figure 2: An example of the split of a location q ($\mathcal{A}_1$ is before and $\mathcal{A}_2$ after). Guards are omitted.

triggered in $\mathcal{A}_1$ if and only if the transitions $q' \xrightarrow{e_i} f(e_i) \xrightarrow{e'_j} q''$ ($f(e_i)$ corresponding to q_i) are triggered in $\mathcal{A}_2$. This allow us to express the probability of e_j in $\mathcal{A}_1$ using the knowledge in $\mathcal{A}_2$, which results in $p_{\mathcal{A}_1}(e_j \mid f(e^{-1})) = p_{\mathcal{A}_2}(e'_j \mid q_i)$, where e^{-1} denotes the transition triggered before e_j in $\mathcal{A}_2$. To express this in terms of entropy, we introduce the random variable E (denoting the triggered transition outgoing from a location in Q_{new}) and E^{-1} (the transition triggered before E). Then, $H_{\mathcal{A}_1}(E \mid q_s(E), f(E^{-1})) = H_{\mathcal{A}_2}(E \mid q_s(E))$ because there exists a trivial bijection between $q_s(E), f(E^{-1})$ in $\mathcal{A}_1$ and $q_s(E)$ in $\mathcal{A}_2$. This way, we can estimate the impact of the split operation on the cost of encoding the edges as

$$\begin{aligned}
&|R_{\{\text{tr,fo}\}}|(H_{\mathcal{A}_2}(E \mid q_s(E)) - H_{\mathcal{A}_1}(E \mid q_s(E))) \\
&= |R^q_{\{\text{tr,fo}\}}|(H_{\mathcal{A}_1}(E \mid q_s(E) = q, f(E^{-1})) \\
&\qquad - H_{\mathcal{A}_1}(E \mid q_s(E) = q)) \\
&= -|R^q_{\{\text{tr,fo}\}}| I_{\mathcal{A}_1}(E; f(E^{-1}) \mid q_s(E) = q) \leq 0
\end{aligned}$$

where $|R^q_{\{\text{tr,fo}\}}|$ is the number of transpose and follow operations using an edge whose source is q, and $I_{\mathcal{A}_1}(E; f(E^{-1}) \mid q_s(E))$ is the mutual information between E and $f(E^{-1})$ given $q_s(E)$ in $\mathcal{A}_1$.

The intuition behind this formula can be explained with Fig. 2. Assume that q in $\mathcal{A}_1$ can be entered with either a, b, or c, with equal proportion, and that the next symbol is either z or d in equal proportion, which makes the encoding cost of the next symbol high. Consider now the split of this location, revealing that c is always followed by z. Now, the cost of z following c is zero because it must happen with probability equal to 1. From the location reached by a and b, the encoding cost of d is much lower in $\mathcal{A}_2$ than it was in $\mathcal{A}_1$, leading to an overall lower encoding cost. This is captured in the formula by the mutual information that is high between two consecutive symbols. The same reasoning can show that $|R_{\{\text{tr,fo,de}\}}| \cdot (H_{\mathcal{A}_2}(D \mid E) - H_{\mathcal{A}_1}(D \mid E)) = -|R^q_{\{\text{tr,fo,de}\}}| \cdot I_{\mathcal{A}_1}(D; f(E^{-1}) \mid E, q_s(E) = q)$.

Location split comes however at a cost in model description because it adds new locations and transitions. Therefore, there is an optimal trade-off in the number of new locations. We propose to find a good partition with a greedy method presented in Algorithm 1. In practice, we first start by splitting edges in $\mathcal{E}_{q,in}$ by using a Gaussian mixture model on the delays associated with them, creating one edge per Gaussian mixture model, to allow for more refined partitioning.

5.3 Learning Algorithm We now present TADAM (*Timed Automata Discovery Applying MDL*), an algorithm that integrates the transformation operations, the word correction strategy, and the MDL cost computa-

Algorithm 1 Split search

Require: A location q, a timed automaton $\mathcal{A}$
Ensure: A partition of $\mathcal{E}_{q,in}$

split incoming edges of q using Gaussian mixtures
$P \leftarrow [\mathcal{E}_{q,in}]$; $mean\_edge\_cost \leftarrow L(\mathcal{A})/|\mathcal{E}|$
$best\_partition \leftarrow P$
while $|P[0]| > 1$ **do**
  for all edge e of $P[0]$ **do**
    for $i = 0$ **to** $|P|$ **do**
      $P' \leftarrow P$
      remove e from $P'[0]$
      if i = 0 **then**
        append new set $\{e\}$ to P'
      else
        add e to $P'[i]$
      $score \leftarrow |R^q_{\{tr,fo\}}| \cdot I(E; f_{P'}(E^{-1}) \mid q_s(E)) - (|P'| - 1) \times |\mathcal{E}_{q,out}| \times mean\_edge\_cost$
      if $score > score(best\_partition)$ **then**
        $best\_partition \leftarrow P'$
  if $P = best\_partition$ **then**
    break
  $P \leftarrow best\_partition$
return $best\_partition$

Algorithm 2 TADAM

Require: Input sample of timed sequences $\mathcal{D}$
Ensure: A pRTA $\widehat{\mathcal{A}}$ partially consistent with $\mathcal{D}$

$\widehat{\mathcal{A}} \leftarrow$ initTA($\mathcal{D}$)
repeat
  $candidates \leftarrow \emptyset$
  for all operation **do**
    for all target in $\widehat{\mathcal{A}}$ **do**
      $\mathcal{A} \leftarrow$ transform($\widehat{\mathcal{A}}$, operation, target)
      $candidates \leftarrow candidates \cup \{\mathcal{A}\}$
  $\mathcal{A}' \leftarrow \arg\min_{\mathcal{A} \in \text{candidates}} L(\mathcal{A}) + L(\mathcal{D}|\mathcal{A})$
  $gain \leftarrow L(\widehat{\mathcal{A}}) + L(\mathcal{D}|\widehat{\mathcal{A}})$ - $L(\mathcal{A}') - L(\mathcal{D}|\mathcal{A}')$
  if $gain > 0$ **then**
    $\widehat{\mathcal{A}} \leftarrow \mathcal{A}'$
until $gain \leq 0$
return $\widehat{\mathcal{A}}$

tion. We provide its pseudo-code in Alg. 2. An open-source implementation is available[1].

The first step is the creation of an initial automaton accepting all words $w \in \mathcal{D}$. Most passive learners of TA start by a tree-shaped automaton, called prefix tree, where each word correspond to a branch [16, 4, 12]. We propose a *Markov initialization*, where each symbol in Σ is associated with a location in the initial automaton, and there is a transition between two locations if their corresponding symbols ever appear consecutively in $\mathcal{D}$.

From this initial automaton, we iteratively perform transformation operations to improve the MDL score. At each iteration, we test all possible transformation operations (merge, deletion, and split) at every position in the automaton. The words are corrected given the new model, and then the MDL score is computed. The transformed automaton leading to the greatest gain in MDL is kept as new base model for the next iteration. If no operation improves the MDL score, the algorithm stops and outputs the automaton with maximal score.

6 Experiments

Throughout the experiments, we are interested in the following questions: to which extend is TADAM robust to noise in the data; how well does it compare to state-of-the-art TA learners; how competitive is it to other methods on real-world problems. We address the two first questions with an experiment on synthetic data. We then present applications on real data with classification and anomaly detection tasks.

6.1 Experimental Setup

We first describe how we generated synthetic data, and then present the setup for the real-world experiments.

6.1.1 Synthetic Data

We generated 35 random target timed automata as data generating process. To ensure a diversity in the target automata, we considered different values of number of locations (from 5 to 50), of alphabet size (5 to 20), and of outdegree (from 1.25 to 2). For each target automaton, we generated a set of timed words. We then injected noise in the timed words, affecting up to 50% of the pairs (a, d) according to the noise model defined in Sec. 4.1 to constitute the training sample $\mathcal{D}_{train}$ for TADAM and its competitors. We considered the following levels of noise n: 0, 0.001, 0.005, 0.01, 0.025, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, and 0.5. We set the default training sample size to 500, nevertheless, we also considered different training sample sizes (from 100 to 1000) for 5 of the target automata (with the following default characteristics: 14 locations, 10 symbols, and an outdegree of 1.25). For the evaluation, we also generated test sets of timed words $\mathcal{D}^+_{test}$ and $\mathcal{D}^-_{test}$ with, respectively, words accepted by the target automaton, and words from the target but with added noise and thus not accepted anymore. Note that words from $\mathcal{D}^-_{test}$ may also belong to $\mathcal{D}_{train}$ where $n \geq 0$.

To evaluate the 600 timed automata learned by the TADAM (and its competitors), we computed the recall, precision, and F1-score using $\mathcal{D}_{test}$. We also computed

[1] https://github.com/Fos-R/TADAM

the Jaccard distance between the untimed language of the target and the learned automata, which measures their similarity without taking into account the word probabilities. We also computed the Jensen-Shannon divergence between the untimed word distribution in the languages of the target and learned automata.

We also conducted an ablation study on TADAM using the automata generated with default values. We tested different initialization strategies, and removed the transformation operations one by one.

6.1.2 Classification We performed a classification task on network traffic, based on the ISCXVPN2016 dataset [6]. In the raw network captures, there are four classes of network traffic: chat, email, streaming, and VoIP. A usual method to classify such data relies on network flow descriptive features, such as number of packets, data rate, etc. We relied on the NTLFlowLyzer software to extract these features. Due to issues with it, we had to leave out some of the original dataset classes. We split the traffic into a training set (with 80% of data) and testing set (with 20% of data). The symbols are formed from the TCP flags, the message directions, and an indicator of the payload, leading to an alphabet with 23 symbols. We created one word per communication flow between two IP addresses, each packet corresponding to a pair (symbol, payload size).

We used a Random Forest on the network flow features as a domain-specific baseline. For the TA learning methods, TADAM, TAG and RTI+, we learned one automaton per class. To predict a class with an automaton, we consider the most likely automaton given a timed word.

6.1.3 Anomaly Detection We conducted an anomaly detection experiment in a collection of HDFS message logs [20]. The HDFS (Hadoop Distributed File System) is a storage system designed to distribute and manage large datasets across multiple machines in a Hadoop cluster. The dataset contains 29 different kind of messages (received data, failed to transfer, etc.). To obtain timed words, we used the message type as symbol and the delay between two messages as time value. We randomly selected 1,000 traces labelled as normal among the 575,061 available traces to constitute the training sample $\mathcal{D}$. Using the learned timed automaton, we then computed the probability of the remaining traces (557,223 normal and 16,838 abnormal) to perform the anomaly detection.

6.2 Validation on Synthetic Data We report in Table 1 the average F1-score, precision, recall and execution time over all data. TADAM significantly outperforms TAG and RTI+ by reaching a high recall (the TA accept most true words from the target model) while keeping a high precision (they reject most words where noise was injected). These performances come at a price of a longer execution time (TADAM and TAG are implemented in Python while RTI+ in C++).

Learner	F1	Precision	Recall	Time (s)
TADAM	**0.884**	**0.849**	**0.938**	1649
TAG	0.803	0.756	0.878	267
RTI+	0.783	0.837	0.763	**0.4**

Table 1: Global learning performance of our method and its competitors.

6.2.1 Noise Robustness We show the impact of the noise ratio on the learned models in Fig. 3. We provide the full results for the other metrics in the appendix. While TAG and RTI+ are immediately impacted by the presence of noise even in a small proportion, TADAM shows more robustness. The Jaccard distance computed between the untimed languages of the learned model and the target increases more progressively for TADAM than its competitors, showing that it has successfully rejected noisy words. The number of edges, which is an indicator of the size of the learned models, remains small. The recall drop of TAG and RTI+ is a symptom of their inability to generalize over the available learning data due to the noise. For TADAM, there could also be a risk to recognize less true words due to inopportune deletions. The empirical

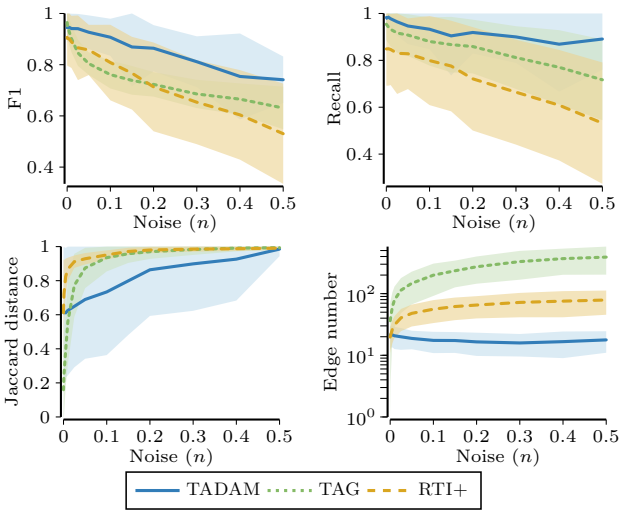


Figure 3: Impact of the noise proportion in the training data on the learned models.

results suggests that this phenomenon is limited.

6.2.2 Ablation Study We tested different initialization strategies: the tree-shaped prefix automaton, the universal automaton, or our Markov initialization. The results confirm the latter is the most appropriate for TADAM. When initialized with the prefix automaton, the runtime quickly skyrockets due to the number of operations to test and of merges needed. The universal automaton is not ideal either, because splitting this single location becomes increasingly complicated as n grows, due to the lack of clear structure in the data.

Running TADAM by removing the transformation operations one by one shows that the most important is the deletion. It is not surprising as this operation brings robustness to noise and induces the word correction process. The other operations appear less critical as our initialization ensures a compact automaton with high mutual information between successive symbols.

6.3 Real-world Applications We now present the results of using the learned timed automata to classify network data and detect anomalies.

6.3.1 Classification We report the classification results in Table 2. TADAM performs slightly better than TAG, though both fall short compared to the domain-specific baseline that achieved a F1-score of 0.720. As in the previous experiments, RTI+ shows significantly lower performance, close to random guess (that would have about 20% accuracy). The feature importance analysis from the Random Forest reveals that high-level metrics, such as average rate of data or the number of certain events, are important for classification. Automata lack the capacity to express this kind of metrics. Finally, TADAM exhibits very little overfitting, achieving similar results on train set and test set, unlike TAG.

	Train set	**Test set**	
Learner	**Accuracy**	**Accuracy**	**F1**
TADAM	0.607	**0.617**	**0.494**
TAG	**0.672**	0.585	0.486
RTI+	0.260	0.255	0.197

Table 2: *Classification performance of TA learners on ISCXVPN2016.* We report the accuracy on the train and test sets, and the mean F1-score on test set.

6.3.2 Anomaly Detection We present the results on the anomaly detection task in Table 3. Although we do not achieve the performance levels of state-of-the-art techniques for log-based anomaly detection, in particular deep learning techniques such as CNN or LogRobust for which F1-score over 0.98 are reported [9], our results are promising. Indeed, the rate of false alarms (FPR) of our learned model is low and its rate of detection (TPR) very high. The learned automaton still demonstrates a strong ability to detect anomalies, despite that learning data was both noisy and limited, and that no adaptation was made for this task. The advantage of the noise robustness becomes evident when we compare our performances with the other TA learners. We also learned an Hidden Markov Model (HMM) as this family of probabilistic models is also used for log structure learning and anomaly detection [10]. However, HMMs clearly lack the expressiveness needed for effective discrimination in this task.

Learner	**AU-ROC**	**TPR**	**FPR**	**F1**
TADAM	**0.982**	0.998	**0.025**	**0.705**
TAG	0.891	**1**	0.142	0.298
RTI+	0.790	**1**	0.292	0.171
HMM	0.608	0.640	0.085	0.288

Table 3: *Anomaly detection performance on HDFS_v1 dataset.* We report the TPR, FPR and F1-score for the threshold maximizing TPR-FPR.

7 Conclusion

We present TADAM, a TA learning algorithm from noisy observational data. Traditional passive TA learners struggle with noise, as they attempt to construct models that strictly align with all observations, resulting in overly complex models with poor generalization. We address this issue by introducing a data correction strategy that selects the most appropriate edit operations to modify a timed word such that it is accepted by a given timed automaton. Based on this strategy, we propose a two-part MDL encoding and a greedy algorithm to learn a TA by balancing the complexity of the model with the fit to noisy data. Experiments on synthetic data demonstrate that TADAM is significantly more robust to noise than existing methods, producing compact and interpretable models that remain faithful to the true data generation process. In addition, real-world experiments highlight the practical utility of the learned automata for tasks such as classification and anomaly detection. This work open new application perspectives for timed automata that can now be more easily inferred from real data. This learning approach could also be extended to other formalisms, such as pushdown automata that can model different phenomena.

8 Acknowledgment

This work has been partially supported by Inria under the SecGen collaboration between Inria, CentraleSupélec and CISPA Helmholtz Center for Information Security.

References

[1] Y. Abdeddaïm, E. Asarin, and O. Maler, *Scheduling with timed automata*, Theoretical Computer Science, 354 (2006), pp. 272–300. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003).

[2] R. Alur and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science, 126 (1994), pp. 183–235.

[3] J. An, M. Chen, B. Zhan, N. Zhan, and M. Zhang, *Learning one-clock timed automata*, in International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2020, pp. 444–462.

[4] L. Cornanguer, C. Largouët, L. Rozé, and A. Termier, *TAG: Learning timed automata from logs*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 3949–3958. Issue: 4.

[5] C. Dima, *Real-Time Automata*, Journal of Automata, Languages and Combinatorics, 6 (2001), pp. 3–24.

[6] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, *Characterization of encrypted and vpn traffic using time-related*, in Proceedings of the 2nd international conference on information systems security and privacy (ICISSP), 2016, pp. 407–414.

[7] P. Grünwald, *The Minimum Description Length Principle*, MIT Press, 2007.

[8] M. Lanvin, P.-F. Gimenez, Y. Han, F. Majorczyk, L. Mé, and E. Totel, *Errors in the cicids2017 dataset and the significant differences in detection performances it makes*, in International Conference on Risks and Security of Internet and Systems, Springer, 2022, pp. 18–33.

[9] V.-H. Le and H. Zhang, *Log-based anomaly detection with deep learning: how far are we?*, in Proceedings of the 44th International Conference on Software Engineering, ICSE '22, New York, NY, USA, 2022, Association for Computing Machinery, p. 1356–1367.

[10] B. Mor, S. Garhwal, and A. Kumar, *A systematic review of hidden markov models and their applications*, Archives of computational methods in engineering, 28 (2021), pp. 1429–1448.

[11] B. J. Oommen and R. K. Loke, *Pattern recognition of strings with substitutions, insertions, deletions and generalized transpositions*, Pattern Recognition, 30 (1997), pp. 789–800.

[12] F. Pastore, D. Micucci, and L. Mariani, *Timed k-tail: Automatic inference of timed automata*, in IEEE International conference on software testing, verification and validation (ICST), 2017, pp. 401–411.

[13] J. Rissanen, *Modeling by shortest data description*, 14 (1978), pp. 465–471.

[14] ——, *A universal prior for integers and estimation by minimum description length*, 11 (1983), pp. 416–431.

[15] S. Tripakis, *Fault diagnosis for timed automata*, in Formal Techniques in Real-Time and Fault-Tolerant Systems, W. Damm and E. R. Olderog, eds., Berlin, Heidelberg, 2002, Springer Berlin Heidelberg, pp. 205–221.

[16] S. Verwer, M. de Weerdt, and C. Witteveen, *A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data*, in Grammatical Inference: Theoretical Results and Applications: 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings 10, Springer, 2010, pp. 203–216.

[17] M. Waga, *Active learning of deterministic timed automata with myhill-nerode style characterization*, in International Conference on Computer Aided Verification, Springer, 2023, pp. 3–26.

[18] F. Wallner, B. K. Aichernig, and C. Burghard, *It's not a feature, it's a bug: Fault-tolerant model mining from noisy data*, in Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE '24, New York, NY, USA, 2024, Association for Computing Machinery.

[19] B. Wiegand, D. Klakow, and J. Vreeken, *Mining Easily Understandable Models from Complex Event Logs*, in Proceedings of the 2021 SIAM International Conference on Data Mining (SDM), 2021, pp. 244 – 252.

[20] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, *Detecting large-scale system problems by mining console logs*, in Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, SOSP '09, New York, NY, USA, 2009, Association for Computing Machinery, p. 117–132.

A Proofs

In Section 4.3, we state that:

$$\sum_{\substack{(o,e,d)\in \\ R_{\{\text{tr,fo,ad}\}}}} -\log(p(e \mid q_s(e))) = |R_{\{\text{tr,fo,ad}\}}| \cdot H(E \mid q_s(E))$$

Here is the proof of that claim:

$$\begin{aligned}
\sum_{(o,e,d)\in R_{\{\text{tr,fo,ad}\}}} -\log(p(e \mid q_s(e))) &= -\sum_{e'\in\mathcal{E}} \sum_{(o,e,d)\in R_{\{\text{tr,fo,ad}\}}} \mathbb{1}[e'=e]\log(p(e' \mid q_s(e'))) \\
&= -\sum_{e'\in\mathcal{E}} \log(p(e' \mid q_s(e'))) \sum_{(o,e,d)\in R_{\{\text{tr,fo,ad}\}}} \mathbb{1}[e'=e] \\
&= -\sum_{e'\in\mathcal{E}} \log(p(e' \mid q_s(e')))|R_{\{\text{tr,fo,ad}\}}|p(e') \\
&= -\sum_{e'\in\mathcal{E}} \log(p(e' \mid q_s(e')))|R_{\{\text{tr,fo,ad}\}}|\frac{p(q_s(e'),e')}{p(q_s(e') \mid e')} \\
&= -\sum_{e'\in\mathcal{E}} \log(p(e' \mid q_s(e')))|R_{\{\text{tr,fo,ad}\}}|p(q_s(e'),e') \\
&= |R_{\{\text{tr,fo,ad}\}}|H(E \mid q_s(E))
\end{aligned}$$

In Section 5.2.3, we state that:

$$|R_{\{\text{tr,fo,de}\}}| \cdot (H_{\mathcal{A}_2}(D \mid E) - H_{\mathcal{A}_1}(D \mid E)) = -|R^q_{\{\text{tr,fo,de}\}}| \cdot I_{\mathcal{A}_1}(D; f(E^{-1}) \mid E)$$

Here is the proof of that claim:

$$\begin{aligned}
|R_{\{\text{tr,fo,de}\}}| \cdot (H_{\mathcal{A}_2}(D \mid E) - H_{\mathcal{A}_1}(D \mid E)) &= |R_{\{\text{tr,fo,de}\}}| \cdot (H_{\mathcal{A}_2}(D) - I_{\mathcal{A}_2}(D;E) - H_{\mathcal{A}_1}(D) + I_{\mathcal{A}_2}(D;E)) \\
&= |R_{\{\text{tr,fo,de}\}}| \cdot (I_{\mathcal{A}_1}(D;E) - I_{\mathcal{A}_2}(D;E)) \\
&= |R^q_{\{\text{tr,fo,de}\}}| \cdot (I_{\mathcal{A}_1}(D;E|q_s(E)=q) - I_{\mathcal{A}_2}(D;E|q_s(E)=q)) \\
&= |R^q_{\{\text{tr,fo,de}\}}| \cdot (I_{\mathcal{A}_1}(D;E|q_s(E)=q) - I_{\mathcal{A}_1}(D;E,f(E^{-1})|q_s(E)=q)) \\
&= -|R^q_{\{\text{tr,fo,de}\}}| \cdot I_{\mathcal{A}_1}(D;f(E^{-1})|E,q_s(E)=q)
\end{aligned}$$

We exploit the fact that $H_{\mathcal{A}_1}(D) = H_{\mathcal{A}_2}(D)$ (because the definitions of $\mathcal{A}_1$ and $\mathcal{A}_2$ do not change the observation of D). The passage from line 2 to 3 is motivated by the fact that $I_{\mathcal{A}_1}(D;E) = I_{\mathcal{A}_2}(D;E)$ for all sequences of symbols and delays except those whose source locations of the triggered edges are q, because $\mathcal{A}_1$ and $\mathcal{A}_2$ only differ for these edges.

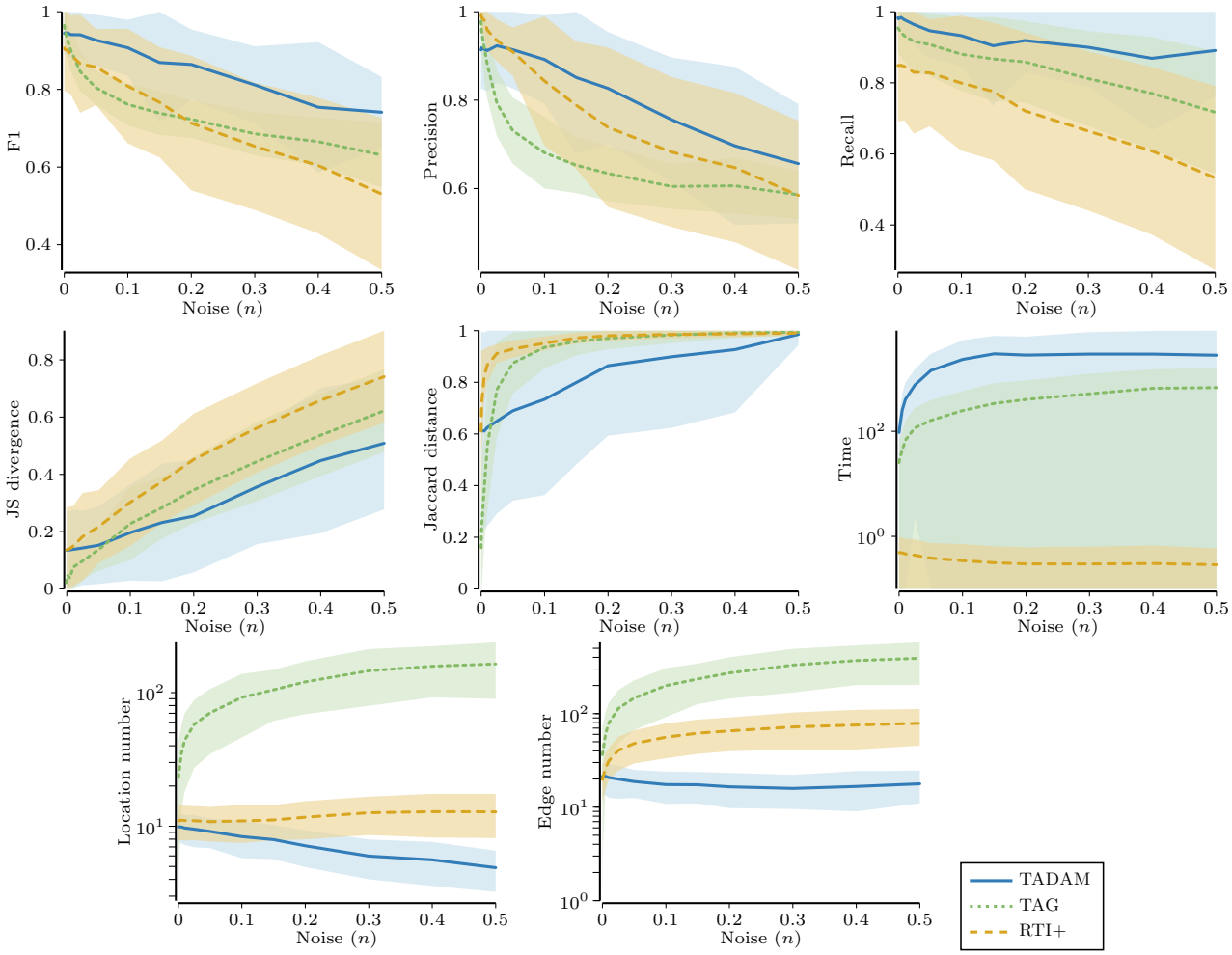


Figure 4: Full results of the synthetic data experiment showing the impact of the noise proportion in the training data on TADAM and its competitors.

B Synthetic Data Experiment

B.1 Robustness to Noise We present in Fig. 4 the full results of the synthetic data experiment.

B.2 Ablation study We first present in Fig. 5 the experimental results for TADAM with different initialization strategies. We either used the classical tree-shaped prefix tree strategy, the universal automaton (a single location with one self-transition per symbol in Σ), or our Markov initialization (one location per symbol). When initialized with the prefix automaton, the runtime quickly skyrockets. From a noise proportion n of 0.025, it starts to exceed 6 hours (we stopped the learning process after these 21600 seconds) due to the number of operations to test and of merges needed. There is also an increased risk of inopportune deletions because deleting a whole subtree will greatly reduce the model cost and it might exceed the increase in data cost on small datasets. In one of the cases, it happened from $n = 0.05$. When initialized with the universal automaton, the only possible operations are the deletion of a transition (improbable at this stage as it would affect all the words have the corresponding symbol), or a split. As the noise proportion rises, less splits appear to be interesting, and no transformation is performed. From $n = 0.2$, the performances dropped in comparison to our proposed initialization. In conclusion, the automaton with a single location per symbol seems to be the best initialization strategy for TADAM.

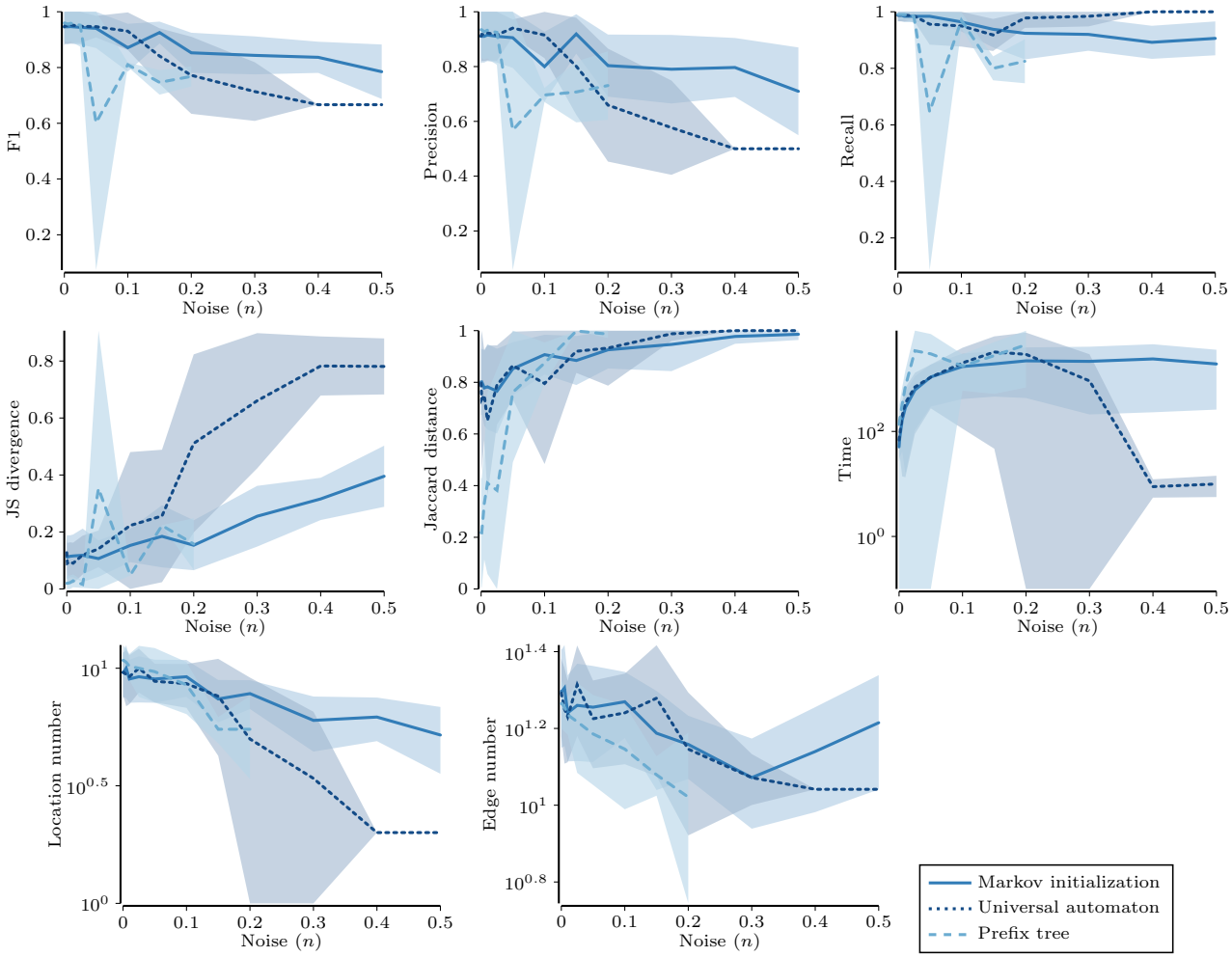


Figure 5: Results of the ablation study on TADAM's transformation operation. The prefix tree initialization caused the learning process to fail due to a learning time exceeding 6 hours (failed cases: 0/20 for $n \in [0, 0.01]$, 1/5 for $n = 0.025$, 2/5 for $n = 0.05$, 9/15 for $n \in [0.1, 0.2]$, 15/15 for $n \geq 0.3$).

We also run TADAM without the possibility to perform either merges, splits, or deletions and present the results in Fig 6. Without the merges, the performances in terms of word acceptance are not significantly different. However, the learned automata are bigger (up to twice as much edges), resulting in a higher runtime (more transformation targets). Without deletions, TADAM unsurprisingly looses its robustness to noise. The average F1-score falls below 0.7 from $n = 0.05$. Without splits, we get slightly worst results. The low difference might be explained by the moderate complexity of the target automata.

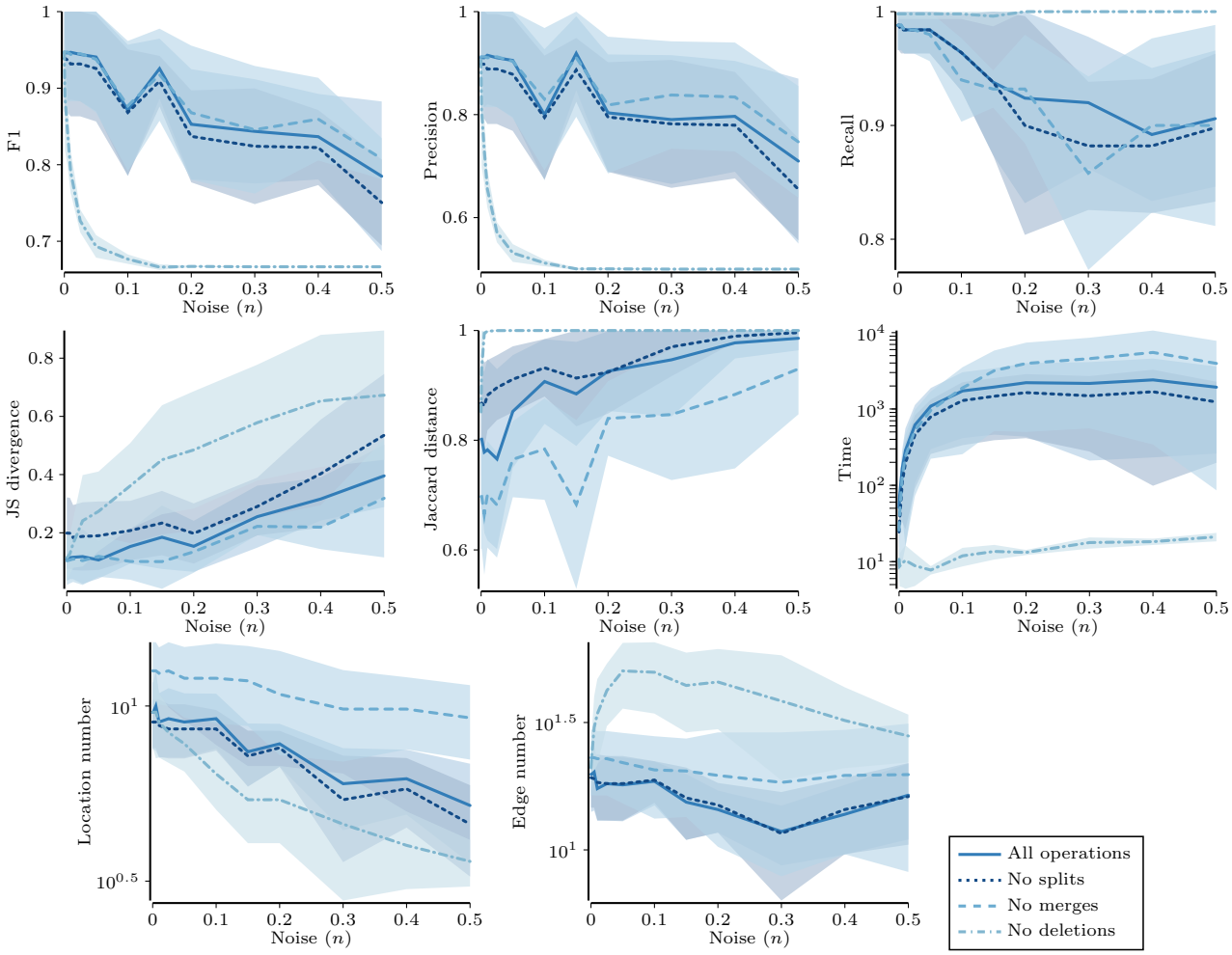


Figure 6: Results of the ablation study on TADAM's transformation operation.