Computer Networks 287 (2026) 112570

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

cAPTure dataset: How fast can you detect APT threats?

Tommaso Puccetti [a,*], Simona De Vivo [b], Davide Zhang [a], Pietro Liguori [b], Roberto Natella [c], Andrea Ceccarelli [a]

[a] University of Florence, Florence, Italy
[b] Federico II University of Naples, Naples, Italy
[c] Gran Sasso Science Institute, L'Aquila, Italy

ARTICLE INFO

Keywords:
Cyber-physical systems
Advanced persistent threats
Attack paths
Intrusion detection
Dataset
Cyber-attacks

ABSTRACT

High-quality datasets are essential for machine learning-based intrusion detection systems, which are considered a promising defense for cyber-physical systems against Advanced Persistent Threats (APTs). However, existing datasets often are not built to capture the long, multi-stage nature of real APT campaigns, and they are labeled as general cyber-attacks rather than explicitly as APTs. To address this gap, we propose a methodology for creating a semi-synthetic, labeled dataset that reflects the complex attack paths typical of APTs targeting cyber-physical environments. Our approach integrates realistic network traffic gathered from a real testbed with multi-step APT attack scenarios modeled on the well-established MITRE ATT&CK framework and CVE exploits repository. The cAPTure dataset provides a rich basis for evaluating intrusion detection systems, enabling an evaluation methodology that relates false positive rate and time-to-detection, two metrics that are crucial for practical, real-world NIDS deployment.

1. Introduction

Industrial processes are increasingly executed or supported by automated cyber-physical systems (CPSs) to reduce operational costs and enhance productivity. Because system performance directly influences stakeholder profits, these environments have become attractive targets for adversaries aiming to exfiltrate sensitive information or disrupt system availability. In more severe scenarios, threats extend to safety-critical applications and infrastructure, where successful attacks may impact human well-being as well as cause economic damage. Recent years have seen increased interest and investment in cyber warfare and cyberattacks [1]. This trend has contributed to the emergence of Advanced Persistent Threats (APTs) [2,3], which are complex attacks executed over a prolonged period against a specific target, and sustained by actors with strategic or financial motivations. APTs integrate conventional intelligence-gathering approaches with advanced cyber-attack techniques to exploit system vulnerabilities. After gaining initial access to a target system, attackers aim to remain undetected to enable continued reconnaissance, lateral movement to escalate control, data exfiltration, and the gradual degradation of system performance. Examples of APT campaigns include the Stuxnet malware [2], which sabotaged Iranian nuclear facilities, and the Ukrainian smart grid attack, which caused a prolonged power outage affecting 225,000 customers [3]. In both incidents, the attacks leveraged prior espionage activities and, in some cases, insider assistance to uncover system vulnerabilities that would otherwise be inaccessible. For this reason, APTs are particularly challenging to detect as attackers tailor their actions to mimic normal system behavior and avoid conventional detection mechanisms.

Anomaly-based intrusion detection systems represent a promising and extensively researched approach [4–7] for defending against sophisticated attacks. These systems analyze network traffic to learn patterns that distinguish normal operations from attacks. However, it is widely acknowledged that the performance of any anomaly-based intrusion detector, being fundamentally a machine learning model, depends heavily on the quality of the datasets used for training and testing. These datasets are essential for selecting, configuring, and validating detection algorithms, which in turn are crucial to building effective defenses. Therefore, constructing high-quality datasets for training these detectors against APTs is a major challenge. In the literature, only a limited number of datasets capture the full complexity of APT campaigns, often limited to isolated attack steps or techniques. Moreover, developing APT datasets through monitoring of real systems is extremely demanding, and data from operational environments, particularly industrial CPSs, is rarely available due to security and

* Corresponding author.
E-mail address: tommaso.puccetti@unifi.it (T. Puccetti).

https://doi.org/10.1016/j.comnet.2026.112570
Received 10 December 2025; Received in revised form 30 April 2026; Accepted 7 July 2026
Available online 8 July 2026

privacy constraints.

Furthermore, most available datasets simulate corporate and enterprise networks and services, typically including a broad range of application-level web services [8,9], whereas only a few studies simulate industrial cyber-physical systems. Moreover, state-of-the-art datasets fail to describe the concurrent interplay between normal system operation and attacker activities that naturally occur in real-world scenarios. They also fall short of representing a realistic APT campaign. As a result, such datasets are inadequate for evaluating a detector's ability to disrupt an attack progression, which is an essential requirement for APT detection, where multiple techniques and stages unfold.

We present a methodology and a toolset to build a semi-synthetic intrusion detection dataset specifically designed for APTs detection, and its instantiation to obtain the cAPTure dataset. More specifically, in this instantiation, we simulate APT campaigns targeting a CPS industrial network operating on a publish/subscribe architecture. Unlike most studies, the dataset reproduces publish/subscribe communication traffic from a real industrial network. This approach ensures that the resulting traces reflect the behavior of the underlying industrial application, rather than depending solely on fully simulated environments or abstract user profiles. cAPTure includes APT attacks based on the MITRE ATT&CK framework [10], combining multiple techniques and covering the APT lifecycle. In many existing network intrusion detection datasets, attack techniques are selected with reference to MITRE ATT&CK; however, they are typically executed in isolation, without following a structured attack graph whose paths define a precise and coherent attack scenario. As a result, attacks are often injected as independent events or in batches, rather than as logically connected steps of a multi-stage campaign. In contrast, cAPTure follows predefined attack paths derived from an attack graph, each representing a specific APT scenario. The overall dataset is composed of a concatenation of such scenarios, each executed according to its corresponding path. Moreover, attacks are not performed in isolated bursts; instead, they reproduce alternation between normal activity and malicious actions. Each attack step is represented by a sequence of network actions corresponding to the individual operations required to achieve that step, thereby preserving the temporal and logical structure of a multi-stage attack lifecycle.

As a result, the cAPTure structure is organized as a time series, alternating between periods of normal system communication and the execution of APT steps. Unlike most network intrusion detection datasets, which aggregate activity into flows representing complete communication sessions, cAPTure records every packet exchanged within the system and preserves the indices of those generated during each attack step. This enhanced granularity enables precise identification of malicious sequences embedded within normal traffic. It also allows the computation of attack latency, defined as the time to detection, which is the interval between the first malicious packet generated by an APT step and the moment the detector flags it. This metric supports the evaluation of intrusion detection systems based on their ability to interrupt an attack path by recognizing intermediate steps or sequences, as well as their speed in detecting an ongoing attack.

We assess the dataset by training several Network Intrusion Detection Systems (NIDSs), illustrating how its structure can be leveraged to measure both the latency of individual attack steps and APT detectability. Our experiments include algorithms that operate on individual data points as well as models tailored for time-series analysis and feature-engineering approaches that exploit the temporal characteristics of the traffic. This dataset design offers a new perspective on NIDS evaluation, revealing how temporal information influences detection latency and, ultimately, the effectiveness of attack detection.

The cAPTure dataset comprises over 25 million data points and 103 features, monitored across seven days of operation, resulting in a total size of 25 GB. We also provide a simplified 1.5 GB version of the dataset, produced by reducing the feature set and applying row subsampling.

All scripts and detailed instructions for reproducing the dataset composition methodology, along with the raw monitored network data, the full and reduced datasets, and the tools for training and evaluating detectors, are publicly available at [11].

This paper extends our previously published conference paper [12], which appeared in the proceedings of the 44th International Conference on Computer Safety, Reliability and Security. The conference paper introduced the methodology for composing the dataset and provided a preliminary raw sample. In contrast, this extension presents the final refined dataset, which introduces several data quality enhancements aside from delivering a complete dataset. First, we substantially improve the simulation environment by doubling the number of attack exploits and the volume of normal traffic, expanding the APT simulations to 7 days. We also improve the dataset composition methodology, particularly for raw traffic conversion, where we use an enhanced script to ensure precise encoding of MQTT packets. Furthermore, we improve labeling and merging, refining our scripts to avoid mislabeling.

Additionally, this paper includes a new comprehensive Data Record section that details the dataset, including data distributions and features. Furthermore, we present an extensive experimental campaign that evaluates the dataset using various intrusion detection algorithms. The aim is to guide practitioners and demonstrate how the unique structure of cAPTure can be leveraged to compute latency-related metrics and assess the ability to interrupt APT attacks.

The remainder of the paper is organized as follows: Section 2 presents background knowledge. Section 3 presents related work and positions our dataset against the state-of-the-art. Section 4 describes the methodology for creating the dataset. Section 5 describes the cAPTure dataset. Section 6 details how to use the dataset and the experimental campaign. Section 7 discusses the experimental results. Section 8 concludes the paper, specifies its limitations, and outlines future work.

2. Background

2.1. APTs

Advanced Persistent Threats (APTs) are attacks carried out by skilled, well-organized groups that target the computing systems of a victim organization. *Advanced* refers to the use of a full spectrum of intelligence-gathering and exploitation techniques. They are *persistent* since they span long periods of time and silently spread across the victim's computer network until specific objectives are achieved [12]. *Threat* highlights their focus on extracting strategic and sensitive information, leading to large-scale damage [2].

APTs are modeled in different phases that compose the attack's lifecycle, with each phase comprising different *attack steps*. This paper adopts the *attack lifecycle* defined by Engebretson [13] as it best fits the characteristics of industrial CPS environments, where the objective may include disrupting services and damaging facilities, in addition to exfiltrating sensitive data, which is the primary goal in enterprise networks [12]. In this scenario, techniques such as spear phishing and social engineering, while still applicable to system users, are usually not appropriate for CPS. Further, an attacker may leverage web services in a typical enterprise network (e.g., email servers or VoIP applications) that simply should not be expected to exist in CPSs. Briefly, the attack lifecycle includes: i) *Reconnaissance*, in which the attacker gathers information through traffic analysis and social engineering techniques (where applicable), to learn about devices and hosts connected to the network; ii) *Scanning*, in which the attacker, once inside the system, applies additional techniques to vulnerable services and accounts; iii) *Exploitation*, where the attacker performs malicious activities on target devices; and iv) *Maintaining Access*, in which the attacker establishes persistent access to the system.

A variety of resources and frameworks have been developed to document, classify, and describe cyberattacks, providing structured knowledge for understanding adversarial behavior. The MITRE ATT&CK [10] is a comprehensive framework for understanding, documenting, and developing defensive strategies against the tactics, techniques, and

procedures (TTPs) of advanced cyber adversaries. Each step of the attack life cycle provides techniques that can be applied to execute the attack, along with literature examples, possible target assets, the impact of the attack, and detection strategies.

The Common Vulnerabilities and Exposures (CVE) [14] scheme provides detailed information about software vulnerabilities. It assigns unique identifiers to publicly disclosed vulnerabilities and describes how they can be leveraged for attacks, making it an insightful source about real exploits.

2.2. Datasets generation approaches for intrusion detection

A promising solution to intrusion detection [4–7] is training a classifier relying on monitored system indicators (also referred to as features) encoded as structured data. Attacks are suspected whenever a deviation from normal behavior, i.e., an anomaly, is suspected. The detection of anomalies is then formulated as a binary classification problem: the class *(normal or attack)* is predicted for each input data point fed to the detector. The training process can be performed either in a supervised or unsupervised way, depending on the availability of labels in the training set.

The interest in anomaly-based intrusion detectors has led to huge efforts to compose datasets suitable for training. Three main approaches are exercised: i) synthetic approach: data is crafted by applying different data generation techniques, ii) semi-synthetic approach: the system and network are simulated in a virtual environment, or iii) monitoring a real system. The synthetic generation has the advantage of completely controlling the data-generated properties. Collaterally, no realistic noise is introduced in the data, causing a clear distinction between normal and under-attack behavior and, ultimately, leading to a simplified representation of the system behavior. Instead, monitoring a real system implies attacking an actual operational system, possibly degrading the quality of service and availability, when not escalating into cascading failures. Further, it provides low repeatability and low scalability in terms of user scenarios, and the collected data may present privacy issues that prevent their publication [12]. A semi-synthetic approach provides reasonable pros and cons between purely synthetic data and real monitored data [15]. Data generated with a semi-synthetic approach is more realistic than synthetic data, as it includes network noise while being easier to produce. However, they can be unrealistic when using a simplified user model [16].

2.3. Evaluating attack latency

In the scientific literature [4,17,18], intrusion detectors are evaluated mostly in terms of the correctness of the detection using state-of-the-art classification metrics such as f1-score, precision, recall, and accuracy. However, achieving high performance in terms of data point classification alone, without consideration of the time elapsed since the attacker initiated its attack until detection, i.e., the attack latency, may provide an insufficient understanding of the detector's abilities. Considering the temporal dimension is especially important for APTs, where intrusion detection needs to analyze events over long periods of time, and must be timely to mitigate the spread of the attack.

We summarize the procedure and metrics to evaluate latency as proposed in previous work [19]. Given S an *attack step*, composed of all the network packets corresponding to the execution of an attack, time instant t_i is the starting time of the attack, and instant t_d is the time when detection is performed ($t_d \geq t_i$), we have that *attack latency* is $\Delta l = t_d - t_i$. When the attack step is undetected, t_d, is unavailable, meaning the attack succeeds and the latency corresponds to the full duration of the step. Additionally, the *sequence detection rate SDR = detected attack steps / total attack steps* is computed to quantify the ability to detect the ongoing attack before it succeeds. We can measure the average latency ΔL over the N attack steps, that is:

$$\Delta L = \sum_{i=0}^{N} \Delta l_i \Big/ N$$

In Fig. 1, we represent 3 attack steps, 2 of which are detected, achieving SDR=0.67. The average latency $\Delta L = (\Delta l_1 + \Delta l_2 + \Delta l_3)/3$, where Δl_3 is the duration of the undetected step.

3. Related work and positioning of our approach

3.1. Network intrusion detection and APT datasets

In Table 1, we report notable public datasets that are commonly used to train intrusion detectors and should be taken as a comparison.

The datasets KDD / NSL-KDD [20], ADFA LD/WD [21], ISCX [22], CICIDS17 [23], CICIDS18 [23], InSDN [24], IoT-IDS [25], LANL Dataset [26], Modbus [27], SWaT [28], VASTs [29], OpTC [30], TON IoT [31], X-IIoTID [32], CICIoT2023 [33] describe simulated or real networks in which different attacks were executed. Compared to our proposed dataset, they execute isolated attacks that fail to resemble a realistic APT where different techniques and phases alternate to reach specific objectives. Attacks are executed consecutively in batches, without alternating with the normal behavior, and usually report communication flows, summarizing the behavior of the entire interaction rather than reporting the packets [34]. This flow-level representation aggregates multiple packets into a single record and therefore removes fine-grained temporal information, such as packet ordering and inter-arrival times, which are required to analyze the evolution. For these reasons, they are not suitable for computing attack latencies. Differently, ROSPaCe [35], Edge-IIoTset [36], and BATADAL [37] support the evaluation of attack latency, but they do not describe APTs.

The CICAPT-IIoT [38], Unraveled [8], DAPT2020 [39], and Linux-APT [9] have significantly contributed to the realism of APT datasets. These works incorporate multi-stage attacks mapped to the MITRE ATT&CK framework and include advanced features such as provenance tracking, enterprise behavior emulation, and detailed host-level telemetry. However, compared to them, the cAPTure dataset presents distinctive characteristics and addresses several complementary gaps.

DAPT2020 [39], Unraveled [8], and Linux-APT [9] model realistic APT attacks that reproduce the APT lifecycle and target simulated enterprise infrastructures and web services. In contrast, cAPTure simulates a CPS industrial network in which device communications rely on an industrial publish/subscribe protocol rather than enterprise services. Enterprise environments, however, are characterized by heterogeneous, user-dependent interactions spanning multiple services and protocols. Consequently, modeling realistic enterprise traffic requires

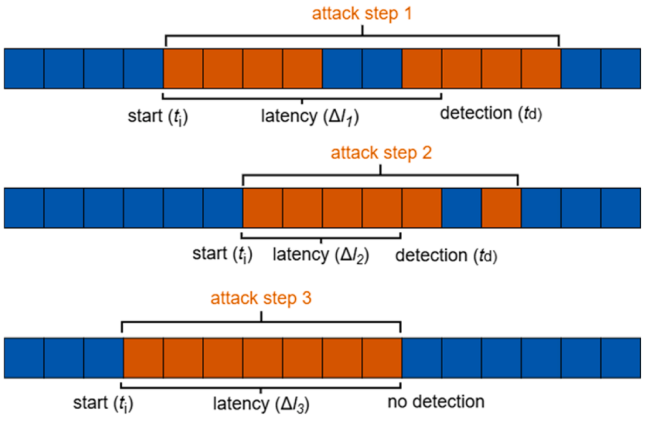


Fig. 1. Overview of the proposed approach for latency measurement, where blue packets correspond to normal traffic and red packets indicate attack steps.

Table 1
Comparison of state-of-the-art IDS and APT datasets. Column dom (domain) specifies whether the monitored system is an enterprise network or a CPS network. The apt column indicates if the attack APT lifecycle is reproduced in the attack model. The type column specifies whether the dataset is generated synthetically (synth), simulating network interactions (semi), or monitoring a real network (real). The lat. (latency) column specifies if the dataset is suitable for computing attack latency.

dataset	year	dom	apt	type	data	duration	timestamp	lat
KDD/NSL-KDD	1999	ent	no	real	net	7 weeks	yes	no
ADFA LD/WD	2014	ent	no	real	log	not reported	no	no
ISCX	2012	ent	no	real	log	7 days	yes	no
CICIDS17	2017	ent	no	real	net	5 days	yes	no
CICIDS18	2018	ent	no	semi	net	5 days	yes	no
InSDN	2020	ent	no	semi	net	18 hours	yes	no
IoT-IDS	2019	iot	no	real	net	45 minutes	yes	no
LANL Dataset	2019	iot	no	real	net	90 days	yes	no
ROSPaCe	2024	cps	no	real	net, log	4 days	yes	yes
Modbus	2016	cps	no	real	net	6.5 hours	yes	no
SWaT	2020	cps	no	semi	net, log	11h	yes	no
BATADAL	2018	cps	no	synth	log	1 year	yes	yes
VASTs	2018	ent	no	semi	net, log	3 days	yes	no
OpTC	2019	ent	yes	semi	net, log	2 week	yes	no
DAPT2020	2020	ent	yes	semi	net, log	5 days	yes	no
TON IoT	2020	cps	no	semi	net, log	not reported	yes	no
X-IIoTID	2021	cps	no	semi	net, log	not reported	yes	no
Edge-IIoTset	2022	cps	no	real	net	not reported	yes	no
Unraveled	2023	ent	yes	semi	net	10 weeks	yes	no
CICIoT2023	2023	cps	no	real	net	16 hours	yes	no
CICAPT-IIoT	2024	cps	yes	semi	net, log	7 days	yes	no
Linux-APT	2024	ent	yes	semi	net, log	3 months	yes	no
This paper	2025	cps	yes	semi	net	4.5 days	yes	yes

sophisticated behavioral emulation of users and services, which increases complexity and may introduce abstraction artifacts not present in protocol-driven industrial communications.

A second key difference concerns data granularity. Cited APT datasets are primarily composed of network flows and host-level logs, and do not preserve fine-grained packet-level and temporal information, such as inter-arrival times and packet sequencing. Flow aggregation abstracts temporal details that may be crucial for identifying attacks. cAPTure employs a different architectural design, providing a packet-level representation of traffic that preserves communication semantics and dynamics. This enables detection approaches based on packet-level observability and fine-grained communication behavior.

Third, while cited APT datasets provide tactic, technique, or stage-level labels, they do not explicitly encode ordered attack-step progression within traffic streams. In these datasets, labels are typically associated with events, flows, or logs, but are not explicitly linked to a temporally ordered sequence of attack steps at the packet level. cAPTure explicitly identifies and tracks individual attack steps, associating each step with the exact set of packets generated during its execution. As a result, detection can be evaluated based on attack evolution rather than coarse binary classification, enabling detectors to be assessed by how quickly they detect malicious activity along the attack path.

Existing APT datasets typically assess detection performance using standard classification metrics such as accuracy or f1-score. While informative, these metrics do not account for a crucial operational factor: detection latency, which refers to the time taken to identify an ongoing APT step. In cyber-physical environments, delayed detection can directly impact operational safety and system integrity.

None of the datasets examined provides the combination of packet-level traces, explicit attack-step indexing, and fine-grained temporal information required to systematically measure detection latency across attack stages. cAPTure introduces latency-aware evaluation features, including step detection rate and detection latency analysis.

More broadly, while existing datasets are well-suited for offline analysis, forensic reconstruction, and classification benchmarking, they are not explicitly designed for operational benchmarking under real-time constraints. cAPTure is structured to support evaluation of detection performance in timing-sensitive CPS environments, bridging the gap between offline research and holistic deployment-oriented intrusion detection evaluation.

4. Methodology

We propose a methodology, together with a toolset, to compose a semi-synthetic APT detection dataset that leverages pre-existing network traces from a real industrial network. Briefly, we reproduce the normal behavior of a CPS network thanks to virtual devices that replay real network traffic. In such a controlled, monitored environment, we can inject APTs and suspend or rewind execution whenever needed. This approach provides scalability and flexibility to define and simulate multiple APT scenarios. The methodology is organized following nine steps, summarized below. Steps 1 to 4 are detailed in Section 4.1, step 5 is detailed in Section 4.2, and steps 6 to 9 are detailed in Section 4.3. Steps are:

1. Select a dataset representing the network communication of a real system. The dataset should contain all the packets exchanged between nodes, e.g., as *pcap* files logged by Wireshark.
2. Analyze the selected dataset to understand how communication is organized. We need to identify the IP addresses of devices, device roles (e.g., as in the publish/subscribe protocol), and the network topology.
3. Simulate multiple network devices and channels to replicate the dataset topology and the interactions by replaying logged packets. This step can be done with the support of existing tools; for example, we will use DDoShield [40] to create and configure a simulated device for each IP address in the dataset.
4. Monitor the simulated network during normal operations, capturing network traffic. Again, with the support of a tool like DDoShield, we replay the packets from the dataset, respecting the timing specified by their timestamp. The simulated components react accordingly following the network protocol.
5. Define APT attacks to exploit the simulated network and protocols. To design APTs suitable for the target system and network, we refer to the MITRE ATT&CK [10] tactics and techniques. To implement specific stages of APT attacks, we refer to the CVE [14] list of vulnerabilities.
6. Run the APT attacks in a dedicated run, while capturing network traffic. The execution of the normal traffic (as can be generated by replaying the dataset) is halted, so that we can identify attack packets and the related responses, obtaining precise labeling.

7. For each attack step of the APT path, record the timestamps corresponding to periods of attacker activity, and label the related network traffic with a specific identifier. Each attack step should be repeated in different iterations to increase the representation of each phase. For each iteration, the duration of individual attacks is randomized, and a random idle period is introduced between attack steps, during which the attacker remains inactive. Packets collected during these idle periods are irrelevant for characterizing the attack and are therefore discarded.
8. Combine the normal and attack traffic to create a unique dataset. Select a timestamp from the normal traffic as the starting point to merge the APT packets. The first packet of the APT is assigned this timestamp, and all subsequent packets have their timestamps adjusted accordingly while preserving the original time intervals between them. The merged dataset is then sorted chronologically by timestamp.
9. We retain the labels for both attack and normal traffic, enabling clear differentiation between anomalous and benign activity. As a result, the dataset is well-suited for training intrusion detectors using supervised learning techniques. Additionally, by recording the timestamps of the first and last packets of each attack step iteration, we can assess both the detector's ability and the speed in identifying APT steps.

4.1. Semi-synthetic normal behavior

We simulate the communication channels and interactions of a real CPS network, illustrated in Fig. 2, that implements a publish/subscribe paradigm. In publish/subscribe, only publish messages are caused by specific system tasks (e.g., a sensor read value at time t that has to be published on a topic), while the other messages are generated collaterally, either to send published messages to subscribers or to provide communication acknowledgments. Publish/subscribe facilitates the control flow within a network and is a good simplifier for our purpose of replicating the system's behavior. We selected the DoS/DDoS-MQTT-IoT [41] dataset, which provides the network-collected traffic of a physical testbed. The available *.pcap* files contain the traffic produced by various physical IoT sensors located in different locations of an industrial facility. Data is communicated using Raspberry Pi (publishers) through the Message Queuing Telemetry Transport (MQTT) protocol, sending the collected data to the MQTT Broker, and subsequently to Raspberry Pi's (subscribers). The dataset collects traffic for about 4 days. We analyze the pcap files to i) identify the IP addresses of the devices in the network, and ii) the device's roles (i.e., publishers, subscribers, and brokers).

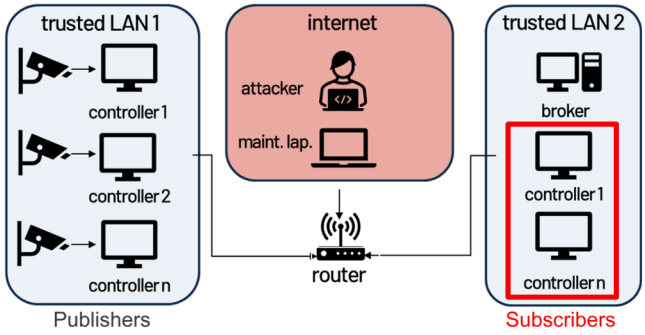


Fig. 2. Simulated CPS network composed of two trusted LANs connected through a router. Trusted LAN 1 contains Raspberry Pi controllers that collect sensor data and act as MQTT publishers. Trusted LAN 2 hosts the broker and the MQTT subscribers. The external Internet, shown in red, represents the environment through which both the operator's laptop and the attacker access the system.

We configure the MQTT network using the DDoShield tool that leverages Docker containers and the NS-3 network simulator to accurately mimic IoT environments and traffic [40]. We create the same number of MQTT devices to match the ones in the dataset. We map each device in real traffic with one of the devices in the simulated network, assigning the role accordingly (e.g., the publisher with IP 192.168.1.14 is mapped to the 10.0.0.14 publisher device in the simulated network). We filter the traffic by only selecting publish messages. We apply the defined mapping to substitute the source and destination IP addresses with the simulated device's IP addresses. We extract header and payload information from the dataset packets, and we use it to generate and send MQTT packets via the simulated MQTT publishers. We assign the resulting messages to the simulated publishers based on the source IP.

Further, we configure the simulated MQTT broker to expose the same topics as the real network and the subscriber device to subscribe to the same topics as in real traffic. The publisher's devices will send the real message by following the timing given by the dataset timestamps. The broker reacts to published messages, orchestrating traffic to subscribers. Additionally, we set up a maintenance operator laptop that accesses the system from outside the network. The operator accesses the devices via SSH to execute shell commands in a programmed time window.

We configure a total of 23 machines with IP addresses ranging from 10.0.0.1 to 10.0.0.23. The 10.0.0.1 machine executes the MQTT broker, 10.0.0.2 is the operator or the attacker machine, and 10.0.0.3 logs the network traffic using Wireshark. All three are implemented using Ubuntu 20.04. The 10.0.0.20-23 machines are MQTT subscribers, and the remaining devices are MQTT publishers. We implement both publishers and subscribers using Alpine Linux v3.17 machines.

4.2. APTs threat model

We adopt the MITRE ATT&CK [10] framework to define realistic APT scenarios, selecting one or more techniques (attack steps) for each APT stage. Since we aim to use the dataset for training and evaluating network intrusion detectors, we focus on attack techniques that can be potentially detected through network analysis. In this Section, we detail attackers' goals (Section 4.2.1), capabilities (Section 4.2.2), knowledge (Section 4.2.3), and constraints (Section 4.2.4). Section 4.2.5 presents the selected attack techniques and describes how the attacker orchestrates them.

4.2.1. Attacker goals

We model an attacker targeting an industrial cyber-physical system network. The model considers multiple attack scenarios in which several techniques are combined to achieve the attacker's goals. These goals include discovering system topology and exposed services, obtaining access to devices, establishing and maintaining persistence within the infrastructure, exfiltrating operational data, and degrading system performance. Such objectives align with documented APT campaigns targeting industrial environments and are consistent with the multi-stage attack lifecycle described in the MITRE ATT&CK framework.

4.2.2. Attacker capabilities

We assume that the attacker has access to the LAN network and possesses advanced technical capabilities to orchestrate multi-step attacks via network vectors. Specifically, the attacker can perform network reconnaissance and scanning, exploit known vulnerabilities, misconfigurations, and weak credentials, deploy malicious scripts, and execute DoS attacks. These capabilities translate to the attack steps described in Section 4.2.5 and summarized in Table 2.

4.2.3. Attacker knowledge

We assume the attacker to have only partial initial knowledge of the system. Information about network topology, services, and devices is acquired by executing reconnaissance and scanning scripts during the early stages of the attack.

Table 2
Summary of the dataset attack steps. Indicate for each attack step name of the attack step, the corresponding MITRE techniques, the CVE used for the exploit, and the reference phase of the APT lifecycle.

attack step	MITRE technique	CVE ID	lifecycle phase
nmap_10_T4, nmap_10_T5	T1595.001 Scanning IP blocks		Reconnaissance
mqtt_cat, nmap_banner, nmap_mqtt, nmap_sub, nmap_10_T4, nmap_10_T5,	T0840 Network Connection Enumeration, T0888 Remote System Information Discovery, T0842 Network Sniffing		Scanning
brute_force_m, brute_force_t	T0866 Exploit Remote Services, T0886 Remote Services, T0859 Valid Accounts	CVE-2018 15473	Maintaining Access
scp_inst, sftp_inst	T0853 Scripting, T0869 Standard Application Layer Protocol		Installation
slash_char	T0853 Scripting, T0869 Standard Application Layer Protocol, T0826 Loss of Availability	CVE-2019 11779	Exploitation
scp_exf, sftp_exf	T0853 Scripting, T0869 Standard Application Layer Protocol, T0882 Theft of Operational Information		Exploitation
pub_exf sub_exf	T0853 Scripting, T0869 Standard Application Layer Protocol T0882 Theft of Operational Information	CVE-2017 7650	Exploitation
dollar_char	T0853 Scripting, T0869 Standard Application Layer Protocol, T0826 Loss of Availability	CVE-2018 12543	Exploitation
user_prop_ddos	T0853 Scripting, T0869 Standard Application Layer Protocol, T0828 Loss of Productivity and Revenue	CVE-2021 41039	Exploitation
empty_conn, empty_conn_ddos	T0853 Scripting, T0869 Standard Application Layer Protocol, T0828 Loss of Productivity and Revenue	CVE-2023 5632	Exploitation
qos_mid, qos_mid_ddos	T0853 Scripting, T0869 Standard Application Layer Protocol, T0828 Loss of Productivity and Revenue, T0826 Loss of Availability	CVE-2023 28366	Exploitation

4.2.4. Attacker constraints

We assume that the attacker is connected to the local area network (LAN) and can execute *attacks exclusively through this network vector*. No physical access or host-level manipulation is considered.

Since the dataset is derived from a real system with limited normal network traffic, the attacks must be executed within restricted time windows so they can be coherently merged with the baseline traffic while preserving realistic network behavior.

4.2.5. Selected attacks

We consider attack techniques that specify *DS0029-Network* [10] as a possible detection mechanism. The only exception is the *sub_exf* attack step, which assumes an already exploitable vulnerability in the subscriber's configuration. As a result, this exploit does not involve any network traffic to be triggered and is therefore not included as a detectable attack step in the final dataset. We identify a total of 21 attack steps that cover all the phases of the attack lifecycle [13]. The scripts and the entire simulation environment are available in our repository [11] to enable full reproducibility.

To implement attack steps and put them into action, we inject exploitable vulnerabilities obtained from the CVE database [14]. We report in Table 2, for each attack step, the CVE used for the exploit, and the reference phase of the lifecycle.

Reconnaissance. Search for active IP addresses and ports using *nmap* network and port scanning. The scripts are categorized based on the scan velocity and the range of IP addresses targeted. The *nmap_10_T4* script employs a slower packet transmission rate, whereas *nmap_10_T5* represents a faster variant of the same script.
Scanning. Discover network topology and services using network enumeration scripts and command-line tools. This stage includes *nmap_banner, nmap_*sub, *nmap_mqtt, and mqtt_cat*. The *nmap_banner* script uses nmap to scan network open ports and identify the active services and their version. The *nmap_sub* script connects to the MQTT broker and subscribes to the default topics to retrieve information about exposed topics and clients involved. The *nmap_mqtt* script collects information about MQTT services by scanning default protocol ports. Last, *mqtt_cat* searches for known configuration files on target machines. Additionally, in this phase, we apply the *nmap_10_T4* and *nmap_10_T5* scripts that are used on the first attack contact with the target but share the same goal as the reconnaissance steps.
Maintaining Access. Apply brute force to steal SSH access credentials to network machines. We execute the CVE-2018-15473 exploit using *Metasploit*. The script enumerates SSH users on the target publisher or subscriber machine and runs the *Hydra* brute force attack based on malformed packets to crack the users' passwords. We execute the attack with basic settings (*brute_force_m*) or with a dedicated Metasploit configuration applying a slower request rate (*brute_force_t*).
Installation. Upload malicious scripts on the target machines using the scp protocol (*scp_inst*) or using *sftp* server pre-installed on the target (*sftp_inst*).
Exploit. Exploit the target machine's known vulnerabilities. This phase includes the following attack steps. Each attack step reproduces the relevant CVE exploit, using the following dedicated scripts:
empty_conn: exploits the CVE-2023-5632 MQTT vulnerability by establishing empty connections with the broker from one or multiple exploited publishers, resulting in memory overload and loss of productivity.
empty_conn_ddos: the attack represents a DDoS variant of the *empty_conn* attack step, relying on coordination among multiple infected systems.
slash_char: exploits the CVE-2019-11779 vulnerability to crash the broker. The attacker subscribes to a topic with a username consisting of 65400 or more "/" characters.
dollar_char: exploits the CVE-2018-12543 vulnerability by publishing messages prefixed with $ to exposed MQTT topics, to crash the target broker.
pub_exf: exploits CVE-2017-7650 to subscribe to restricted topics by using Mosquitto credentials with special usernames ('#' or '+').
user_prop: exploits CVE-2021-41039 by connecting to the broker with many user-defined properties, causing excessive CPU usage and loss of performance.
user_prop_ddos: it is the distributed version of the *user_prop* attack step.

qos_mid: it exploits CVE-2023-28366 by sending messages with QoS 2 and duplicate MID while not replying to subsequent PUBRECs to overload the broker's memory with unsent packages.
qos_mid_ddos: it implements the distributed version of the *qos_mid* attack step.
sftp_exf and *scp_exf*: they exfiltrate sensitive information from a compromised machine via, respectively, sftp or scp channels.

We present in Fig. 3 the attack graph that combines the various attack steps and defines the possible ways of composing the APT. We selected 7 attack paths from the attack graph shown in Fig. 3 to define distinct simulations of the complete APT lifecycle. Each path includes one or more attack steps from the blue box corresponding to the *Reconnaissance, Scanning, Maintaining Access*, and *Installation* phases, and one or more attack steps from exactly one of the blue boxes associated with the *Exploitation* phase. The DDoS variants of attack steps are executed within the same path as the corresponding non-DDoS step. Table 3 provides a brief example of how we use the attack graph shown in Fig. 3 to construct an APT path.

Table 3
Example of an APT scenario composed from the APT graph in Fig. 3.

attack phase	attack step
Reconnaissance	nmap_10_T4
	pause
	nmap_10_T4
	pause
	nmap_10_T5
	pause
Maintaining Access	brute_force_m
	pause
Scanning	mqtt_disc
	pause
	mqtt_disc
	pause
Exploitation	dollar_char
	pause
	dollar_char
Scanning	nmap_banner
	pause
…	…

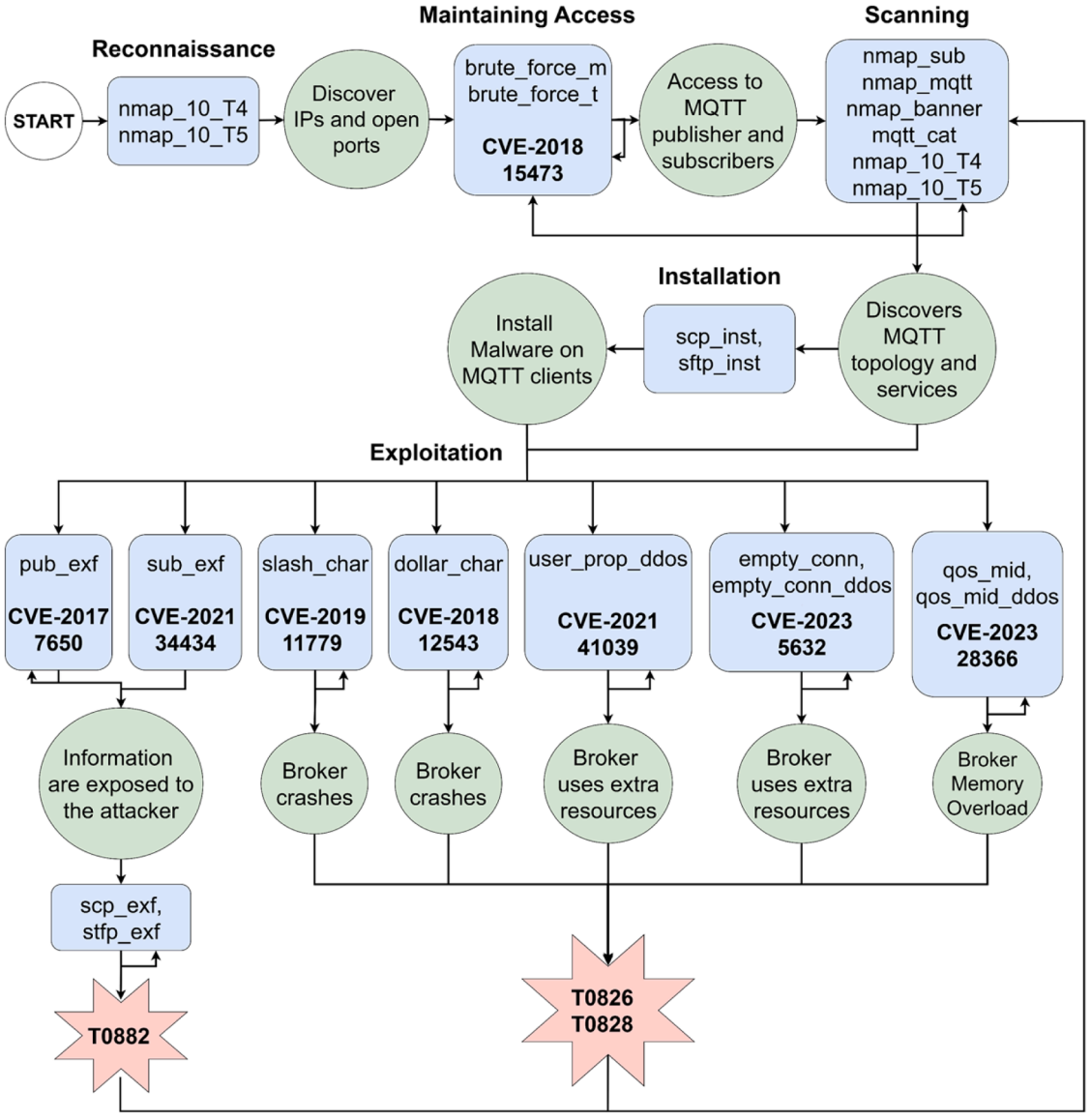


Fig. 3. Attack graph composing the APT we consider in this paper. In the blue boxes, we report the attack steps indicating the corresponding lifecycle phase, in green the effect on the system, and in red the impact as identified by MITRE.

4.3. Composing and labeling the dataset

We simulated normal traffic by replaying the selected MQTT dataset from three distinct days of system operation. Two of these days cover approximately 12.5 hours each, while the third covers 10 hours, resulting in a total of 35 hours of system execution.

Then, we simulate the 7 APT scenarios defined in Section 4.2 in isolation, ensuring that the only non-attack traffic consists of initial subscriber connections and low-level protocol messages generated by the network infrastructure, which represents background network noise. Each scenario is executed twice, once for the training set and once for the test set. We randomize both the durations of the attack steps and the pauses between them, using different non-overlapping randomization intervals for the two sets. As a result, the training and test simulations differ in the combination and timing of attack steps, as well as in the temporal distribution of packets.

At the end of this process, we obtained 3 normal traffic simulations and 14 attack traffic simulations, 7 for training and 7 for testing, each saved as a separate PCAP file. These files are then converted into csv format. The APT path simulations are labeled to identify the packet sequence for each of the executed attack steps, along with the corresponding phase and technique. This approach ensures that both normal and attack traffic can be labeled before merging, resulting in accurate ground truth. Details about the ground-truth labels in the dataset are provided in Section 5.2.

The normal and APT path simulations are then merged to create the final dataset. We illustrate in Fig. 4 how to compose training and test sets starting from these csv files. To create the training set, we merge the traffic from each of the 7 training APT simulations with 2 days of normal traffic. For each APT scenario, one of the 2 days of normal traffic is randomly selected for the merging process. We repeat the same procedure for the test set: each of the 7 testing APT path simulations is concatenated with the normal traffic from the third day. We specify in Fig. 5 the procedure to merge normal and attack simulations. We select a random packet from the normal traffic and merge the first packet of the attack traffic at that point. The timestamps of the remaining attack packets are then modified to be successive to the modified first packet timestamp, while maintaining the original relative timing. The packet indexes composing each attack step are retained. This allows each attack step to be easily accessed among normal traffic for computing its detection latency.

At the end of this process, we obtained 7 complete APT training scenarios, including normal and attack traffic, and 7 for testing. The training APT scenarios are then concatenated into a single sequence. The same concatenation procedure is applied to the test scenarios.

We then perform a feature selection process. First, all features related to time, except the timestamp, are removed to prevent inconsistencies arising from the merging of different simulated traffic streams. Next, we compute the correlation matrix for the remaining features and remove any features that are correlated above 0.999. Finally, we apply the mutual information algorithm on the training set to identify the features that best distinguish attack-labeled packets from normal traffic. From the resulting ranking, the top 100 features were selected for inclusion in the dataset.

4.4. Generation of alternative test scenarios using the attack graph

While the provided test sets allow controlled, repeatable measurement of SDR and latency across multiple iterations of identical attack steps, more realistic, diverse scenarios can be generated. Specifically, new scenarios can be constructed as follows:

1. Select an attack path from the attack graph.
2. For each step in the path, randomly select a packet sequence corresponding to one iteration of that attack step.
3. Modify the inter-arrival times between packets in the selected sequence to control the packet rate, enabling the simulation of more or less stealthy attack behavior.
4. Insert each selected sequence into benign traffic at a chosen merge point.

This process can be repeated to generate different interleaving and timing configurations for the same attack path, producing multiple realistic variants of the scenario.

We provide the code to reproduce these steps and an example of a resulting test set in our repository [11].

Importantly, this flexibility is enabled by the semi-synthetic approach adopted in this work. Since benign traffic and attack traces

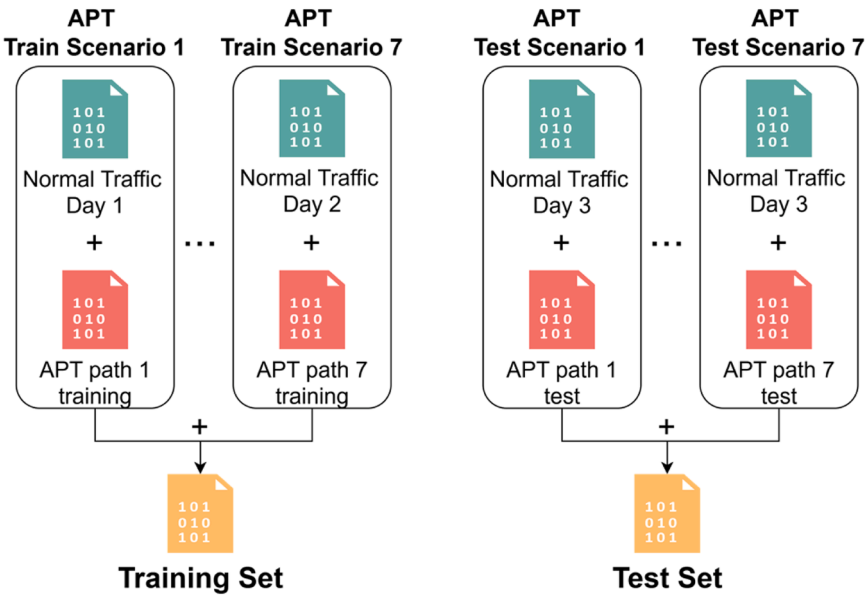
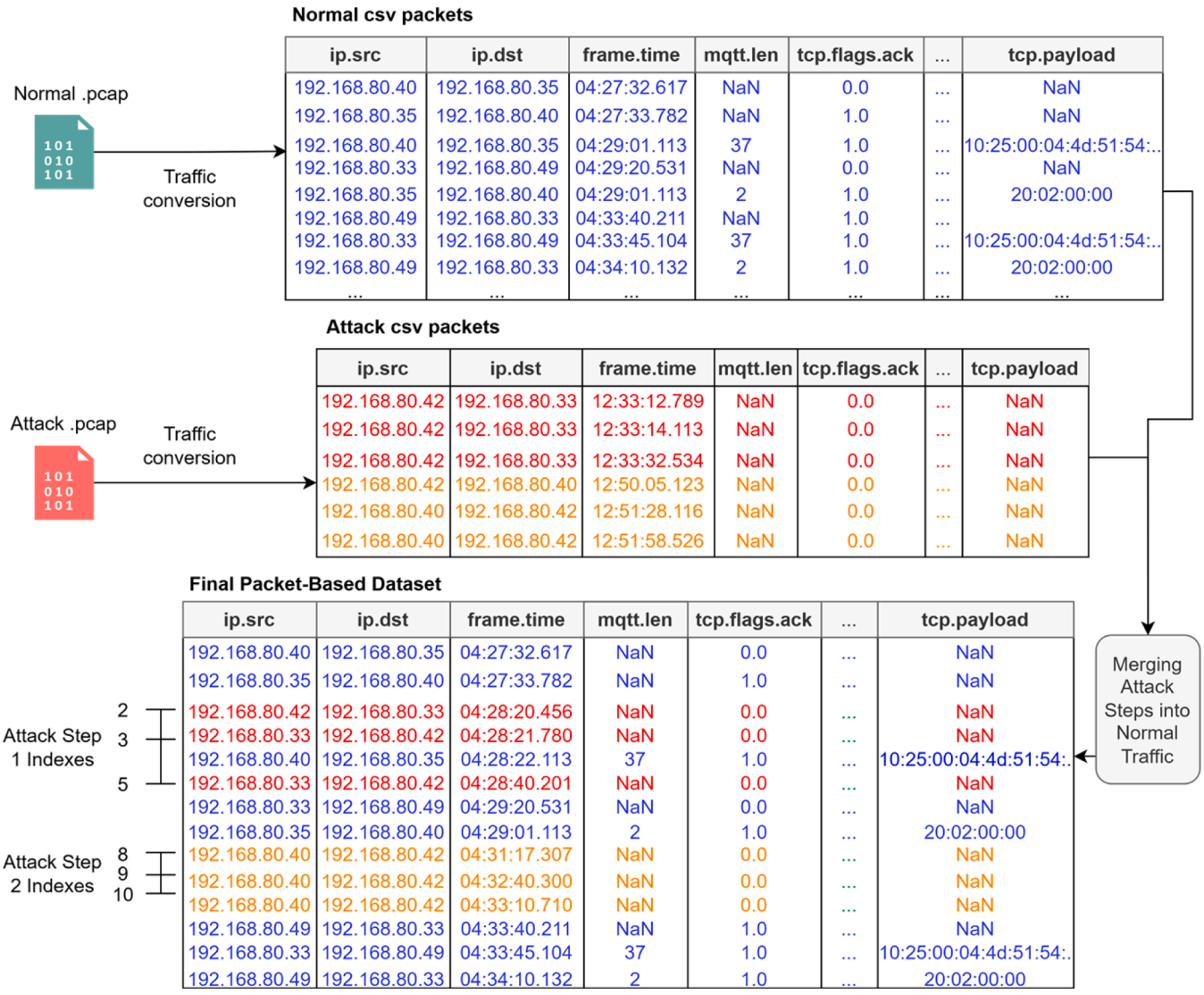


Fig. 4. Process used to construct the train and test sets by merging APT path simulations with normal traffic. Each of the seven APT training paths is combined with a randomly selected day from the first two normal-operation days, and each of the seven test paths is combined with the third day. After merging and adjusting timestamps, we obtain 7 complete APT training scenarios and 7 complete test scenarios. All training scenarios are then concatenated into a single continuous training sequence (and similarly for testing).



ip.src	ip.dst	frame.time	mqtt.len	tcp.flags.ack	...	tcp.payload
192.168.80.40	192.168.80.35	04:27:32.617	NaN	0.0	...	NaN
192.168.80.35	192.168.80.40	04:27:33.782	NaN	1.0	...	NaN
192.168.80.40	192.168.80.35	04:29:01.113	37	1.0	...	10:25:00:04:4d:51:54:..
192.168.80.33	192.168.80.49	04:29:20.531	NaN	0.0	...	NaN
192.168.80.35	192.168.80.40	04:29:01.113	2	1.0	...	20:02:00:00
192.168.80.49	192.168.80.33	04:33:40.211	NaN	1.0	...	
192.168.80.33	192.168.80.49	04:33:45.104	37	1.0	...	10:25:00:04:4d:51:54:..
192.168.80.49	192.168.80.33	04:34:10.132	2	1.0	...	20:02:00:00
...	...	...	...	...	...	...

ip.src	ip.dst	frame.time	mqtt.len	tcp.flags.ack	...	tcp.payload
192.168.80.42	192.168.80.33	12:33:12.789	NaN	0.0	...	NaN
192.168.80.42	192.168.80.33	12:33:14.113	NaN	0.0	...	NaN
192.168.80.42	192.168.80.33	12:33:32.534	NaN	0.0	...	NaN
192.168.80.42	192.168.80.40	12:50.05.123	NaN	0.0	...	NaN
192.168.80.40	192.168.80.42	12:51:28.116	NaN	0.0	...	NaN
192.168.80.40	192.168.80.42	12:51:58.526	NaN	0.0	...	NaN

ip.src	ip.dst	frame.time	mqtt.len	tcp.flags.ack	...	tcp.payload
192.168.80.40	192.168.80.35	04:27:32.617	NaN	0.0	...	NaN
192.168.80.35	192.168.80.40	04:27:33.782	NaN	1.0	...	NaN
192.168.80.42	192.168.80.33	04:28:20.456	NaN	0.0	...	NaN
192.168.80.33	192.168.80.42	04:28:21.780	NaN	0.0	...	NaN
192.168.80.40	192.168.80.35	04:28:22.113	37	1.0	...	10:25:00:04:4d:51:54:..
192.168.80.33	192.168.80.42	04:28:40.201	NaN	0.0	...	NaN
192.168.80.33	192.168.80.49	04:29:20.531	NaN	0.0	...	NaN
192.168.80.35	192.168.80.40	04:29:01.113	2	1.0	...	20:02:00:00
192.168.80.40	192.168.80.42	04:31:17.307	NaN	0.0	...	NaN
192.168.80.40	192.168.80.42	04:32:40.300	NaN	0.0	...	NaN
192.168.80.40	192.168.80.42	04:33:10.710	NaN	0.0	...	NaN
192.168.80.49	192.168.80.33	04:33:40.211	NaN	1.0	...	NaN
192.168.80.33	192.168.80.49	04:33:45.104	37	1.0	...	10:25:00:04:4d:51:54:..
192.168.80.49	192.168.80.33	04:34:10.132	2	1.0	...	20:02:00:00

Fig. 5. Illustrates the process of merging normal traffic simulations with traffic from the simulated APT paths. Pcap files are first converted into CSV format. The timestamp of each attack step is then used to label packets, identifying their sequence, phase, and associated technique. The attack traffic is inserted into the normal traffic at selected points, with timestamps adjusted to preserve the relative timing between packets.

are generated separately, and packets associated with each attack step are precisely tracked, it is possible to isolate, manipulate, and reinsert complete attack-step sequences in a controlled and reproducible manner. This fine-grained traceability allows systematic construction of diverse and realistic APT scenarios while preserving full ground-truth knowledge of each step.

5. Data records

This section introduces the cAPTure dataset. Section 5.1 describes the dataset composition, label distribution, and details of the attack steps included. Section 5.2 defines the dataset features.

5.1. Data summary

The cAPTure dataset has been deposited and is freely available at [11]. The final shape of cAPTure consists of 25,729,596 packets (15,290,494 from the train set and 10,439,102 from the test set, with, respectively, 1,365 and 861 attack step iterations. 54.69% of the data (14,070,400 packets) is normal traffic, while 45.31% (11,659,196 packets) is related to attacks. In Table 4, we detail the label distribution of the dataset separately for the training set, test set, and the overall dataset.

In Table 5, we present information on the iterations of each attack step, separated into training and test sets. We include the number of iterations per attack step, the duration in seconds, and the length measured in number of packets, with their respective minimum, maximum, and standard deviation.

The dataset comprises roughly 7 days and 14 hours of recorded system traffic, with 3 days and 19 hours allocated to both training and test sets.

In Fig. 6, we show the distribution of the number of packets for the dataset. Fig. 6.a and Fig. 6.b display the distribution of normal packets for, respectively, the training and test set. Fig. 6.c shows the distribution of attack packets for the training set, and Fig. 6.d shows the same information for the test set. Overall, the different subfigures of Fig. 6 illustrate how using different days of normal operation, along with varying durations and pauses in the APT scenarios for the training and test sets, results in differences in both normal and attack traffic between the two sets.

Additionally, Fig. 7 compares the packet count distributions for both attack and normal labels, respectively, for the training and test sets, highlighting how attack traffic deviates from normal behavior. The

Table 4
Packet counts (packets # column) and relative percentages per attack step (packets % column. Information is displayed separately for the training set, test set, and overall dataset. The last two rows present the percentage of normal and attack packets.

attack step	training set		test set		overall	
	packets					
	#	%	#	%	#	%
brute_force_m	46,421	0.30	123,261	1.18	169,682	0.66
brute_force_t	430,085	2.81	537,557	5.15	967,642	3.76
dollar_char	739,942	4.84	197,370	1.89	937,312	3.64
empty_conn	6,65	0.04	4,110	0.04	4,775	0.04
empty_conn_ddos	28,269	0.18	9,589	0.09	378,58	0.15
mqtt_cat	6,101	0.04	6,029	0.06	12,130	0.05
nmap_10_T4	2,384,677	15.60	510,233	4.89	2,894,910	11.25
nmap_10_T5	809,253	5.29	2,001,677	19.17	2,810,930	10.92
nmap_banner	119,761	0.78	96,430	0.92	216,191	0.84
nmap_mqtt	52,746	0.34	71,380	0.68	124,126	0.48
nmap_sub	106,785	0.70	41,272	0.40	148,057	0.58
pub_exf	16,592	0.11	12,928	0.12	29,520	0.11
qos_mid	12,477	0.08	7,664	0.07	20,141	0.08
qos_mid_ddos	257,148	1.68	218,891	2.10	476,039	1.85
scp_exf	5,764	0.04	3,277	0.03	9,041	0.04
scp_inst	811	0.01	1,013	0.01	1,824	0.01
sftp_exf	1,792	0.01	633	0.01	2,425	0.01
sftp_inst	707	0.008	934	0.01	1,641	0.01
slash_char	389,562	2.55	424,818	4.07	814,380	3.17
user_prop_ddos	814,529	5.33	1,160,058	11.11	1,974,587	7.67
normal	9,060,422	59.26	5,009,978	47.99	14,070,400	54.69
attacks	6,230,072	40.74	5,429,124	52.01	11,659,196	45.31

figure shows that some APTs exhibit periods in which they send a significantly higher number of packets compared to normal traffic, while others generate packet counts that overlap with normal activity. This indicates that the dataset includes both high-impact and low-profile attacks.

5.2. Dataset features

We use XGBoost [42] as a feature ranker. We train the model using the training set, and we run the *feature_importance* function after performing predictions on the test set, to obtain the most informative features. The dataset includes three labels: i) *phase name,* the phase to which the packet sequence belongs, ii) *sequence_id* label associates a packet to an attack step iteration, and iii) *timestamp* indicates the time the packet is recorded by Tshark. Table 6 reports the best 20 features returned by XGBoost and the dataset label. The full list of the features included in the dataset is available at [11].

6. Experimental campaign

In this section, we detail how we exercise and evaluate the cAPTure dataset by training and testing a variety of machine learning algorithms. Our goal is to demonstrate how the dataset structure and its detailed representation of attack stages can be used to assess the timely detection of APTs. We adopt a favorable case in which the model is trained using the entire training set. However, the dataset's semi-synthetic nature, with precisely controlled labeling, also makes it highly flexible and suitable for alternative configurations. For example, it can be restructured to include reduced training data, to remove specific attack types or stages, or to model different operational conditions such as stealthy attack scenarios. This adaptability enables researchers to tailor the dataset to a wide range of experimental designs and threat models.

The section is organized as follows. Section 6.1 summarizes the overall steps of the experimental campaign. Section 6.2 describes the selected machine learning algorithms. Section 6.4 presents the feature engineering process applied to construct temporal features. Section 6.5 outlines the model training and testing procedures.

6.1. Experimental steps

Our experimental campaign is articulated in the following steps:

1. **Algorithms**: We adopt 5 machine-learning algorithms. As in [43], we partition into algorithms for *unordered data points*, relying on single packet information to identify attacks, and algorithms for *ordered data points*, i.e., suited for time series data. The selected algorithms are detailed in Section 6.2.
2. **Time Features**: We generate temporal statistical features to capture traffic evolution at different time intervals to enrich the cAPTure dataset. The goal is to enhance the performance of algorithms for unordered data points by incorporating information about traffic evolution within each packet, information that would otherwise be unavailable to such models. The new features and methodology applied are detailed in Section 6.4.
3. **Testing**: First, we train the selected algorithms, applying a grid search approach. Then, we perform the evaluation applying both classification metrics and the latency-based evaluation detailed in Section 2.3 and in previous work [44]. We then enrich the dataset with time features, and we use them to train the best model for unordered data points to assess whether these features enhance the detector's timely detection capability. We detail this step in Section 6.5, specifying the evaluation metrics applied.

6.2. Algorithms

We evaluated the cAPTure dataset by training and testing five different machine learning algorithms. These algorithms are divided into two groups: those capable of exploiting the temporal evolution of the network traffic, i.e., algorithms for ordered data points that leverage the time-series structure of cAPTure, and those designed for unordered data points, which perform detection based on the packet fed at time t. The selected algorithms are listed below. Detailed information on the model's configuration is provided in our online repository [11]. We select these detectors to demonstrate that cAPTure contains learnable detection features and to showcase evaluation with latency-based metrics. This makes results easily reproducible and verifiable, which aims to promote the usage of our dataset by researchers. We review in Section

Table 5
For each attack step, we report the number of iterations (iter column), the average duration in seconds (avg. duration), and the average length in number of packets (avg. len). For both duration and length, we also include their minimum (min.), maximum (max.), and standard deviation (std.).

	attack step	iter	avg. duration (s)	min. duration (s)	max. duration (s)	std. duration (s)	avg. length	min. length	max. length	std. length
Training Set	brute_force_m	8	877.2	860.48	893.94	17.7	15,407.62	15,252	15,575	101.86
	brute_force_t	5	1,031.94	1,014.6	1,064.18	22.67	107,511.4	107,314	107,66	128.95
	dollar_char	63	38.95	3.69	302.35	48.52	3,132.85	357	24,511	3,912.12
	empty_conn	31	61.62	32.15	81.89	10.36	132.58	95	165	18.22
	empty_conn_ddos	10	339.91	273.4	391.17	39.83	958.9	838	1,051	69.8
	mqtt_cat	62	5.34	5.21	6.17	0.27	97.24	89	114	5.93
	nmap_10_T4	12	56.29	53.21	56.97	1	42,519.41	42,130	42,629	179.1
	nmap_10_T5	47	55.98	55.6	56.84	0.27	42,588.87	42,567	42,649	19.1
	nmap_banner	107	64.45	64.29	65.24	0.25	901.21	886	925	7.36
	nmap_mqtt	75	242.41	242.32	242.84	0.06	951.73	921	1,011	22.97
	nmap_sub	56	54.52	54.35	55.22	0.26	737	713	801	20.54
	pub_exf	17	639.71	600.96	661.02	29.48	760.47	705	802	32.23
	qos_mid	54	8.04	0.19	52.83	12.44	141.92	28	772	177.16
	qos_mid_ddos	50	303.62	230.01	381.47	26.7	4,377.82	3,377	5,466	369.57
	scp_exf	63	0.18	0.17	0.22	0.01	52.01	51	55	0.68
	scp_inst	19	0.19	0.18	0.19	0	53.31	51	55	1.41
	sftp_exf	17	0.16	0.15	0.2	0.01	38.23	36	39	0.75
	sftp_inst	18	0.17	0.15	0.17	0.01	51.88	49	56	1.56
	slash_char	143	37.51	2.2	228.83	52.87	2,970.75	197	18,056	4,150.93
	user_prop_ddos	4	572.92	532.6	592.88	27.38	290,014.5	274,879	297,541	10,262.44
Test Set	brute_force_m	3	882.54	860.26	893.72	19.3	1,547.7	15,391	15,561	85.1
	brute_force_t	4	1051.6	1,048	1,060.86	6.2	107,521.25	107,421	107,639	89.6
	dollar_char	166	55.44	27.3	908.24	67.3	4,457.5	2,170	72,758	5,387.91
	empty_conn	37	94.92	76.62	114.14	9.6	179.73	146	244	21.42
	empty_conn_ddos	7	2,436.50	2,394.3	2,464.03	27.7	4,038.43	3,983	4,100	45.04
	mqtt_cat	64	5.32	5.2	6.11	0.24	95.33	89	109	4.94
	nmap_10_T4	109	58.20	56.1	227.21	16.34	21,877.8	21,283	85,256	6,126.74
	nmap_10_T5	35	60.85	55.65	224.81	28.53	23,121.51	21,284	85,218	10,804.93
	nmap_banner	133	64.43	64.31	65.24	0.23	900.46	886	960	8.73
	nmap_mqtt	56	242.42	242.33	242.87	0.1	941.9	920	1,024	23.6
	nmap_sub	147	54.13	3.30	55.22	4.22	726.43	553	767	18.9
	pub_exf	48	258.05	120.30	420.72	151.13	345.7	168	597	188.20
	qos_mid	39	20.6	0.83	42	10	319.92	47	610	137.4
	qos_mid_ddos	75	237.54	193.8	297	22.24	3,428.64	2,842	4,247	311.31
	scp_exf	117	0.2	0.17	0.22	0.016	49.26	48	57	1.2
	scp_inst	15	0.2	0.2	0.22	0.01	54.1	51	55	1.22
	sftp_exf	49	0.15	0.13	0.17	0.01	36.6	36	40	0.84
	sftp_inst	14	0.16	0.15	0.17	0	50.5	49	53	1.22
	slash_char	243	20.01	0.9	52.5	10.13	1,603.13	116	4,201	793.21
	user_prop_ddos	4	2,395.53	2,380.4	2,417.17	16.237	203,632.25	200,058	205,783	2,600.3

8.2 other detection approaches that may be applicable to cAPTure to improve detection performance beyond the illustrative baselines reported in this work.

Unordered data points are XGBoost, Extremely Randomized Trees, Isolation Forest, and Cluster-Based Local Outlier Factor.

XGBoost (XGB) is based on a decision tree ensemble leveraging gradient boosting to enhance model performance [44]. It builds an additive ensemble of weak learners, where each tree sequentially corrects the errors of its predecessors.

Extremely Randomized Trees (EXTRA) [45] is an ensemble model conceptually similar to Random Forests. However, instead of selecting optimal split thresholds through information gain or Gini impurity, ExtraTrees randomly chooses thresholds for each feature and selects the best among these random candidates. This approach introduces additional randomness, which can reduce variance and improve generalization. We use the implementation provided by [46].

Isolation Forest (ISOL) [47] is an unsupervised anomaly detection algorithm that isolates observations by recursively partitioning data through randomly selected features and split values. The underlying assumption is that anomalies are rare and differ significantly from normal observations, making them easier to isolate and resulting in shorter average path lengths within the trees. We use the implementation provided by [46].

Cluster-Based Local Outlier Factor (CBLOF) [48] is another unsupervised anomaly detection algorithm based on clustering. It first partitions data into clusters and classifies them as either "large" or "small" based on their size. Each data point is then assigned an outlier score that depends on both its cluster size and its distance to the nearest large cluster. Points belonging to small or distant clusters are assigned higher anomaly scores. We adopt the implementation provided by the PyOD library [49].

Algorithms for ordered data points are Bi-LSTM, LSTMCD, and XLSTM.

Bi-LSTM takes a fixed-length sequence (*input_shape*) and feeds it to a bidirectional LSTM with 10 units. The LSTM outputs a single vector (*return_sequences=False*), which goes through a dense layer for nonlinear feature mixing, followed by dropout for regularization. A final dense layer produces class probabilities. The algorithm is trained with an Adam optimizer and the sparse categorical cross-entropy loss.

The XLSTM model integrates XGBoost with an LSTM network, following approaches proposed in recent studies [50–52]. In this architecture, XGBoost is employed as a feature selector to reduce the number of input features provided to the LSTM. This preprocessing step helps eliminate redundant or less informative attributes, improving both model efficiency and generalization performance.

Additionally, we exercise an enhanced model referred to as LSTMCD [53]. This model is designed to capture both spatial and temporal dependencies within network traffic data. Its architecture consists of a series of convolutional layers, followed by LSTM layers and fully connected (dense) layers. The convolutional component is responsible for recognizing patterns in the input sequences that are relevant to

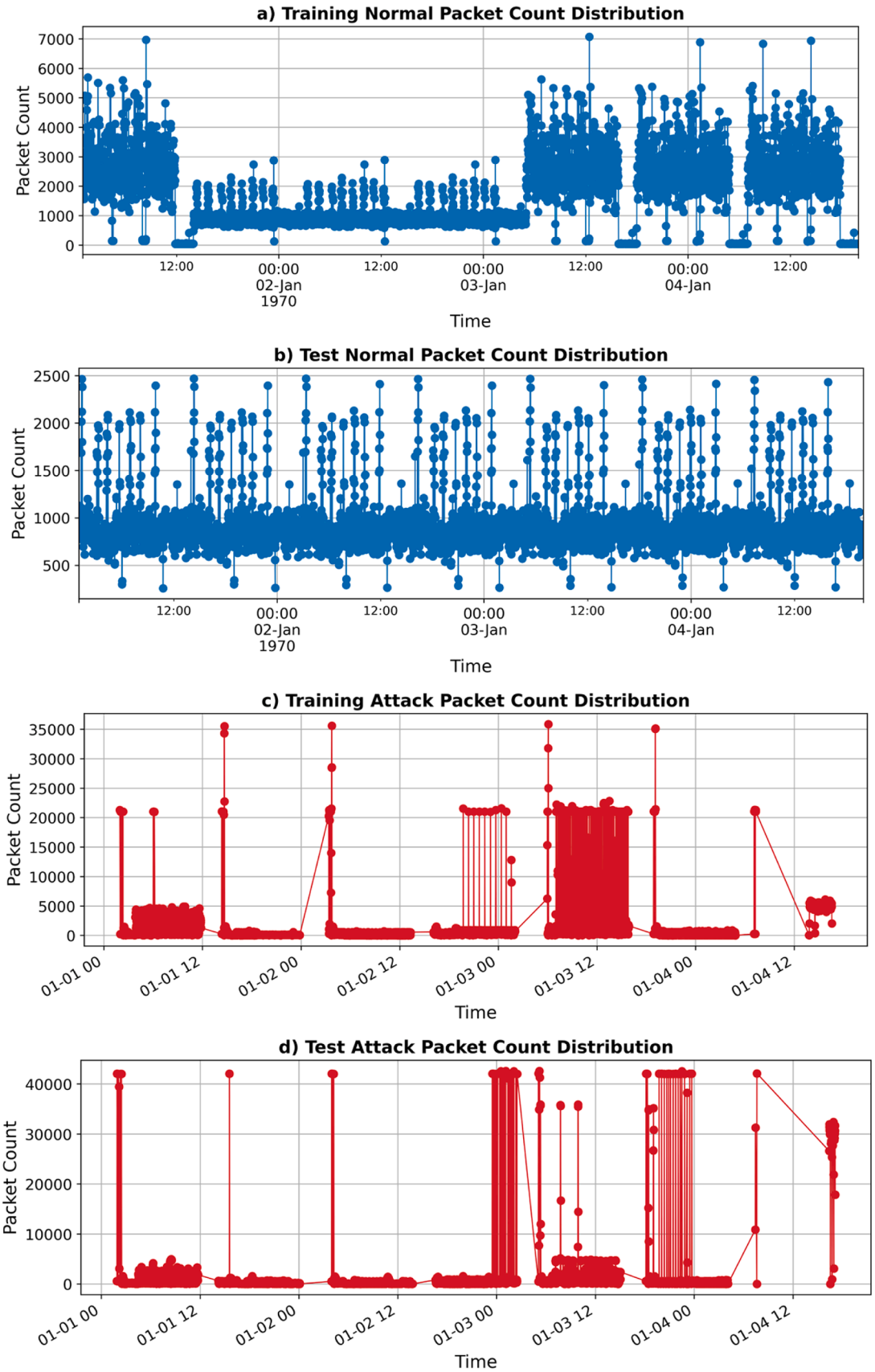


Fig. 6. Distribution of packet counts for the training and test sets. Figures a) and b) present the distributions of normal packets, whereas figures c) and d) show the corresponding distributions of attack packets. The x-axis reports the packet count while y-axis the time in which they are recorder.

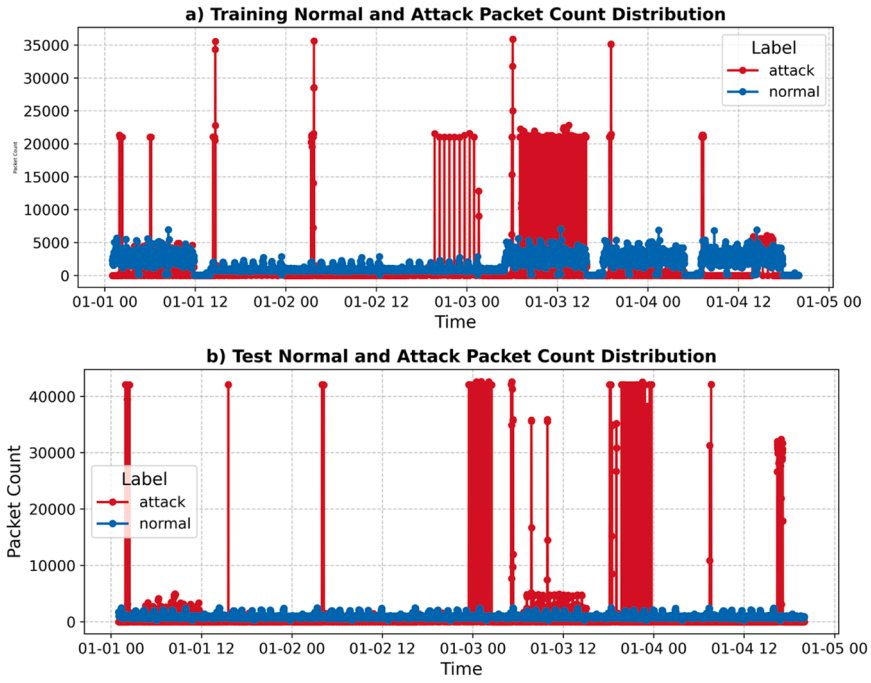


Fig. 7. Comparison of normal and attack packet distributions. Figure (a) illustrates the comparison between attack and normal packets in the training set, while Figure (b) shows the corresponding comparison for the test set.

Table 6
Report the label of the dataset and the 20 top features based on the feature_ranking algorithm of XGBoost.

	feature name	description	score
labels	phase_name	Indicates which APT lifecycle phase the packet belongs to	-
	sequence_id	Unique identifiers that associate a packet with an attack step	-
	timestamp	Time of packet capture	-
	label	Attack step name	
Features	layers_mqtt_mqtt.hdrflags	Header flags	0.162
	layers_tcp_tcp.seq	Sequence Number	0.083
	layers_tcp_tcp.completenes	Conversation completeness	0.078
	layers_tcp_tcp.len	TCP Segment Length	0.063
	layers_frame_frame.protocols	Protocol	0.056
	layers_tcp_tcp.ack	Acknowledgment Number	0.05
	layers_tcp_tcp.window_size	Calculated window size	0.049
	layers_tcp_tcp.nxtseq	Next Sequence Number	0.046
	layers_tcp_tcp.hdr_len	Header Length	0.046
	layers_mqtt_mqtt.msgi	Message Identifier	0.045
	layers_tcp_tcp.analysis_tcp.analysis.push_bytes_sent	Number of bytes sent.	0.038
	layers_ip_ip.version	Version field of the IP heade	0.029
	layers_eth_eth.dst_tree_eth.dst.ig	Individual/Group bit of the dst mac	0.028
	layers_eth_eth.type	Type of the Ethernet header	0.022
	layers_mqtt_mqtt.hdrflags_tree_mqtt.msgtyp	MQTT Message Type	0.017
	layers_tcp_tcp.flags_tree_tcp.flags.str	TCP Flags	0.013
	layers_ip_ip.len	Total length of the IPv4 header	0.013
	layers_eth_eth.dst_tree_eth.addr	Destination MAC address	0.011
	layers_tcp_tcp.payload	TCP payload field	0.01
	layers_tcp_tcp.analysis_tcp.analysis.initial_rtt	Initial round-trip time estimate.	0.005

classification. Specifically, we employ two convolutional layers combined with batch normalization and a ReLU activation. The first convolutional layer contains 64 filters with a kernel size of 5, while the second uses 64 filters with a kernel size of 3. The temporal modeling component consists of three LSTM layers, each configured with 64 units, which capture sequential dependencies across the extracted features. The LSTM outputs are then passed to two dense layers to refine the learned representations. The first dense layer has 100 units, and the second reduces the dimensionality to 50 units. Finally, a softmax output layer with two units produces the binary classification result.

6.3. Models configuration

We applied a grid search strategy to identify the hyperparameter configuration for each model. All models were trained and evaluated on the same training and test sets and using the same evaluation metrics, random seed, and computational environment were used throughout.

We performed model hyperparameter tuning as follows.

We configure XGBoost with *n_estimators* [100, 200, 300], *max_depth* [5,10,18], *learning_rate* [0.01, 0.05, 0.1, 0.5], and column subsampling parameters (*colsample_bynode, colsample_bylevel, colsample_bytree*) [1.0, 0.8, 0.6].

We use for Extra Trees *n_estimators* [100, 200, 300], *max_depth* [5,10, 18], *max_features* [1.0, 0.8, 0.6], *min_samples_split* [5,10,18], and *min_samples_leaf* [5,10,18].

For Isolation Forest, we tuned *n_estimators* [100, 200, 300] and contamination [0.005, 0.20, 0.40, 0.50].

We test CBLOF with *contamination* parameters [0.005, 0.20, 0.40, 0.50].

For the LSTM-based models (LSTM, LSTMCD, XLSTM), we apply a learning rate scheduler that reduces the learning rate by a factor of 0.5 for a minimum of 0.0001 based on validation loss, avoiding extensive grid search. Last, we test dropout rates of 0.2, 0.3, and 0.5, with higher dropout consistently degrading performance.

6.4. Time features

We augment each data point in the dataset, i.e., each packet, with temporal features that describe how traffic evolves around that packet.

This enables models processing unordered data points to take advantage of past information, capturing both short- and long-term temporal dependencies in the traffic for improved classification performance. Similar to what was proposed in previous studies [54,55], we use a per-packet sliding window updated at every packet timestamp and aggregate statistics over multiple window lengths (in seconds): [0.1, 1, 30, 60, 900, 3600]. Short windows capture abrupt changes typical of high-impact DoS bursts, while longer windows capture slow patterns.

We craft three types of features: packet count, packet timing, and payload features.

Packet timing and count features describe the dynamics of communication within each time window and are designed to highlight both short bursts and long-term patterns in the traffic. *Packet count* features capture how the number of packets evolves during APT campaigns.

- *count_from_src*: counts the number of packets sent by the same source MAC address within the selected time interval. This helps identify sources that suddenly increase their transmission rate, as might occur during a DoS attack.
- *count_prev_packtes:* counts all packets observed in the same time interval, regardless of source or destination, providing a measure of the overall traffic load.

Packet timing features (*pkts_timing*) calculate the time between packets within the window. For each interval, we compute the mean, median, minimum, maximum, and standard deviation.

Payload features capture how the content of the packet's payload changes through time. They should reveal unusual or inconsistent payloads that may occur during attacks, such as in denial-of-service scenarios where large volumes of forged packets with the same characters are sent to target machines. We craft the following features:

- *payload_rolling_entropy_tcp*: Computes the Shannon entropy of the TCP payloads transmitted by the same source MAC address within the time window. High entropy indicates highly variable data, while low entropy suggests repetitive or structured content.
- *payload_tcp_unique_chars*: calculates the average number of distinct characters present in the TCP payloads sent by the same source MAC address in the interval. This metric complements entropy by highlighting differences in payload diversity.

We craft a total of 61 additional time features. Each feature is encoded in a dedicated file and available at [11] along with the code to add the features to the dataset.

6.5. Testing

We evaluate the detectors' performance by computing classification metrics and latency-related metrics described in Section 2.3 and in previous work [44]. In Table 7, we summarize the metrics used for evaluating the detectors.

First, we compute the precision–recall and ROC curves that plot the relation between FPR and recall, and between precision and recall, respectively, across different detection thresholds.

Table 7
Evaluation metrics.

Metrics	Formula
accuracy A	A = (TP+TN)/(TP+TN +FP+FN)
false positive rate FPR	FPR = FP / (FP + TN).
recall R	R = TP / (TP + FN)
latency Δl	$\Delta l = t_d - t_i$
average latency ΔL	$\Delta L = \sum_{i=0}^{N} \Delta l_i / N$
sequences detection rate SDR	S = detected/total

We then select and fix multiple FPR thresholds, and we calculate their respective classification and latency-related metrics to enable a fair comparison among detectors: we select FPR thresholds corresponding to a resulting one false positive occurrence within varying time intervals, from 12 hours to 1 minute. This analysis enables the evaluation of each detector under configurations that directly capture its operational behavior.

To compute the FPR thresholds, we use the following formula, considering that the test set contains 55,055 normal packets per hour:

$$FPR = \frac{fp/h}{N}$$

where fp/h is the number of false positives per hour, and N is the number of normal packets per hour.

In Table 8, we report the selected FPR thresholds along with the corresponding FPRs over the specified time intervals. Applying an almost zero FPR threshold, as reported in Table 8, yields precision values that are consistently close to 1. For this reason, we do not report this metric. We do not employ balanced metrics such as the f1-score, as they offer no advantage when operating at near-zero false positive rates.

7. Results

We report and discuss the detector's performance on the cAPTure dataset. Section 7.1 reports overall detection and per-attack step results computed with state-of-the-art classification metrics. Section 7.2 evaluates the detectors using latency-related metrics and discusses the relationship between FPR and attack detection latency. Section 7.3 discusses the improvements obtained with temporal statistical features with both state-of-the-art and latency-related evaluation.

7.1. State-of-the-art metrics

In this section, we discuss the results obtained using classification metrics. These results quantify how challenging it is to train an accurate detector using the cAPTure dataset, that is, a model capable of correctly distinguishing between normal and attack packets.

First, we compute, for each detector, the RoC and precision–recall curves to obtain an initial assessment of the classification performance under different trade-offs. The curves show that the XGB, EXTRA, LSTM, and XLSTM have some ability to identify attack packets, performing significantly better than random chance, while Isolation Forest, CBLOF, and LSTMCD have poor classification performance, resembling random behavior. For this reason, we exclude Isolation Forest, CBLOF, and LSTMCD from our evaluation. The curves are not reported in this paper for brevity, but are easily accessible in our repository [11].

In Table 9, we report on the overall performance of the detectors under the selected FPR thresholds, as listed in Table 8. The EXTRA model shows the best ability to identify individual attack packets within normal traffic. However, the performance remains modest with R=0.46193, at an FPR of 1.5×10^{-6}. The scores show only minor

Table 8
Selected FPR thresholds used to evaluate the detectors (fpr thresholds column). For each threshold, the table reports the target false positive occurrence (fp target occurrence column) and the corresponding expected number of false positives per hour (expected fp/h column).

fp target occurrence	expected fp/h	FPR threshold
1 every 12 hours	0.083	1.5×10^{-6}
1 every 6 hours	0.1667	3.03×10^{-6}
1 every 1 hour	1	1.82×10^{-5}
1 every 30 minutes	2	3.63×10^{-5}
1 every 15 minutes	4	7.27×10^{-5}
1 every 5 minutes	12	2.18×10^{-4}
1 every 1 minute	60	1.09×10^{-3}

Table 9
Overall performance of detectors for each FPR threshold, evaluated using recall (R). Detectors are divided into baseline detectors, which do not exploit temporal features, and XGB detectors trained on different subsets of crafted temporal features. The XGB all time column reports the results of XGB trained using all temporal feature subsets. The best performance among baseline detectors is highlighted in green, while the best performance overall is highlighted in blue.

FPR threshold	baseline detectors				time features			
	XGB	EXTRA	LSTM	XLSMT	XGB payload	XGB count	XGB timing	XGB all time
	R	R	R	R	R	R	R	R
1.5×10^{-6}	0.27799	0.46193	0	0.00461	0.48432	0.00046	0.96044	0.99565
3.03×10^{-6}	0.27799	0.46208	0	0.00752	0.5	0.00153	0.96532	0.9976
1.82×10^{-5}	0.3338	0.46209	0	0.02853	0.56514	0.24344	0.98907	0.99865
3.63×10^{-5}	0.33382	0.46216	0	0.03147	0.61177	0.3363	0.99489	0.99886
7.27×10^{-5}	0.35777	0.46216	0.04096	0.04696	0.66852	0.45016	0.99787	0.99925
2.18×10^{-4}	0.42559	0.4622	0.24131	0.07538	0.70725	0.62812	0.99436	0.99978
1.09×10^{-3}	0.59658	0.48816	0.30278	0.5524	0.78784	0.81645	0.99992	0.99995

variations across thresholds, with the largest gain observed at the highest threshold (1.09×10^{-3}), where the R=0.48816.

The XGB detector, although overall less accurate than EXTRA, displays greater sensitivity to changes in the FPR threshold. Its performance rises from R=0.27799 at 1.5×10^{-6} to R=0.59659 at 1.09×10^{-3}, surpassing extra at this threshold. The LSTM detector shows no ability to detect attacks for thresholds between 1.5×10^{-6} and 3.63×10^{-5}. Its performance improves only at higher thresholds, peaking at R=0.30278 for 1.09×10^{-3}. Finally, the XLSTM model yields almost zero recall across all FPR thresholds except for 1.09×10^{-3}, where it unexpectedly achieves the second-best overall result (R=0.5524).

Overall, at the selected FPR, the detectors show limited capability in identifying attack packets, with over half remaining undetected in most cases, except for XGB and XLSTM at 1.09×10^{-3}, where recall approaches but does not exceed 60%.

Table 10 reports the performance of the detectors per attack step. From this point onward, the evaluation focuses on the threshold 1.82×10^{-5}, which corresponds to approximately one false positive per hour. We adopt this threshold for comparison because it provides a reasonable balance, yielding an actionable number of false positives without significantly impacting system availability by erroneously flagging benign traffic. Currently, no established standards or industry guidelines define an optimal FPR threshold, and recent studies have highlighted this gap in NIDS evaluation [44].

The EXTRA detector achieves the best performance for most attack types, with *nmap_T4_10* and *nmap_T5_10* contributing most to its overall classification accuracy with R=0.988 and R=0.987, respectively. However, XGB provides the broadest coverage across attack steps, successfully detecting a larger variety of attack steps than the extra classifier. The XLSTM model consistently shows poorer performance, with recall values near zero for most attack types. Interestingly, it performs best in detecting brute-force and the *user_prop_ddos* attacks, suggesting that its ability to model temporal patterns in the traffic may help identify similar types of attacks, involving many packets.

Overall, detection is more effective for nmap-based reconnaissance and discovery activities, while attacks involving information exfiltration or DoS attacks exploiting crafted-packet network exploitation remain far more difficult to identify.

7.2. Latency-related metrics

We quantify how fast the detectors can detect the attack steps of the APT strategy, i.e., the attack latency. Table 11 reports the SDR and ΔL for each detector and attack type at the target FPR threshold of 1.82×10^{-5} (1 false positive per hour).

Overall, XGB achieves the best balance between detection coverage

Table 10
Results obtained using state-of-the-art metrics. The top performance for each attack in each dataset is marked in green. FPR threshold is set to 1.82×10^{-5}. Detectors are divided into baseline detectors and XGB detectors, which exploit temporal features. The XGB all time column reports the results using all temporal features. The best performance among baseline detectors is highlighted in green, while the best performance overall is highlighted in blue.

attack step	baseline detectors			time features			
	XGB	EXTRA	XLSTM	XGB payload	XGB count	XGB timing	XGB all time
	R	R	R	R	R	R	R
brute_force_m	0.001	0.002	0.18	0.04	0.043	0.962	0.997
brute_force_t	5.2e-05	0	0.037	1.e-04	0.038	0.991	0.998
dollar_char	0	2e-05	0	0.154	3.5e-04	0.999	0.999
empty_conn	0	0	0	0.007	0	0.972	0.992
empty_conn_ddos	0.009	0.001	1e-04	0.045	5.2e-04	0.939	0.986
mqtt_cat	0.001	0	0	0.004	0	0.868	0.982
nmap_10_T4	0.835	0.988	9.8e-06	0.988	0.727	0.998	0.999
nmap_10_T5	0.685	0.987	0.047	0.987	0.419	0.995	0.999
nmap_banner	0.064	0.186	8.3e-05	0.096	0.039	0.979	0.995
nmap_mqtt	0.054	0.090	0	0.092	0.017	0.966	0.987
nmap_sub	0.08	0.113	2.4e-05	0.119	0.022	0.968	0.989
pub_exf	0.001	0.006	0	0.01	0	0.952	0.968
qos_mid	0	0	0	0.117	0	0.966	0.999
qos_mid_ddos	0.004	0	1e-04	0.097	0.025	0.965	0.994
scp_exf	0	0	0	0	0	0.999	1.0
scp_inst	0	0	0	0	0	0.966	1.0
sftp_exf	0.031	0	0	0.050	0	0.858	0.922
sftp_inst	0.0171	0	0	0.061	0	0.829	1.0
slash_char	2.5e-05	2.8e-05	0.015	0.563	7.5e-04	0.999	0.999
user_prop_ddos	9.5e-05	1.1e-04	0.003	0.233	0.064	0.98	0.999

Table 11
Results obtained using latency-based metrics. The top performance for each attack in each dataset is marked in green. FPR threshold is set to 1.82×10^{-5}. Detectors are divided into baseline detectors, which do not exploit temporal features, and XGB detectors trained on different subsets of crafted temporal features. The XGB all time column reports the results of XGB trained using all temporal feature subsets. The best performance among baseline detectors is highlighted in green, while the best performance overall is highlighted in blue.

attack step	baseline detectors						time features							
	XGB		EXTRA		XLSTM		XGB payload		XGB count		XGB timing		XGB all time	
	SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL
brute_force_m	0.75	345.1	0.625	373.49	1	71.34	1	103.28	0.75	281.1	1	4.4e-05	1	0
brute_force_t	0.6	561.5	0	1031.95	1	69.39	1	194.97	1	133.	1	0	1	0
dollar_char	0	38.95	0.031	33.3	0	38.95	0.65	11.66	0.555	36.45	1	0	1	0
empty_conn	0	61.62	0	61.62	0	61.62	0.903	6.98	0	61.62	1	0	1	0
empty_conn_ddos	0.9	68.05	0.4	248.72	0.1	329.55	1	6.42	0,2	301	1	0	1	0
mqtt_cat	0.048	5.08	0	5.35	0	5.35	0.080	4.9	0	5.35	1	1.1e-05	1	0
nmap_10_T4	1	52.84	1	52.84	0.083	55.88	1	52.84	1	52.84	1	4.02	1	1.6-e06
nmap_10_T5	1	52.77	1	52.77	0.106	55.63	1	52.77	1	52.78	1	3.7	1	0
nmap_banner	1	51.18	1	54.22	0.065	63.84	1	50.68	0.925	54.99	1	4.07e-06	1	4.0-e06
nmap_mqtt	0.987	184.7	1	120.31	0	242.42	1	109.08	0.827	241.4	1	3.3e-05	1	0
nmap_sub	1	52.29	1	54.23	0.018	53.55	1	51.32	0.732	54.34	1	0,0	1	0
pub_exf	0.235	484.2	1	120.31	0	639.72	1	92.01	0	639.7	1	4.9e-05	1	2.3-e05
qos_mid	0	8.04	0	8.04	0	8.04	0.333	0.28	0	8.04	1	0	1	0
qos_mid_ddos	0.74	186.2	0	303.63	0.26	268.92	1	10.06	0.94	72.52	1	1.4e-05	1	0
scp_exf	0	0.18	0	0.18	0	0.18	0	0.18	0	0.18	1	0	1	0
scp_inst	0	0.19	0	0.19	0	0.19	0	0.19	0	0.19	1	0	1	0
sftp_exf	0.235	0.13	0	0.16	0	0.16	0.235	0.13	0	0.16	1	0	1	0
sftp_inst	0.278	0.13	0	0.17	0	0.17	0.5	0.1	0	0.17	1	0	1	0
slash_char	0.077	30.79	0.084	30.09	0.944	4.63	1	0.25	0.175	28.21	1	0	1	0
user_prop_ddos	0.75	90	1	63.9	1	0.09	1	64.07	1	40.37	1	2.5e-05	1	16e-05
all	**0.443**	**68.6**	**0.396**	**67.58**	**0.21**	**76.24**	**0.742**	**29.16**	**0.447**	**69.41**	**1**	**0.06**	**1**	**1.6e-05**

and latency, with an overall SDR=0.443 and ΔL=68.6 seconds. EXTRA follows closely with an SDR=0.39 and ΔL=67 seconds, confirming its stable performance seen in Table 9 and Table 10. XLSTM performs noticeably worse overall, showing an SDR=0.21 and a ΔL=76.2 seconds, but excels in a few specific attack categories where temporal correlation is informative, namely, brute force and *user_prop_ddos* attacks, confirming results from Table 9 and Table 10.

Both XGB and EXTRA achieve nearly perfect detection for Nmap-based attacks. However, both detectors respond relatively slowly compared to the average duration of these attacks reported in Table 5, with detection latencies of approximately 50–55 seconds and 184.72 and 120.31 seconds, respectively, for the *nmap_mqtt* attack. This indicates that they identify the attacks only in their last seconds, potentially allowing the adversary to map most of the network before detection. XLSTM lags behind SDR$\leq$0.1 but still identifies a subset of these scans with slightly higher latencies.

All models struggle to detect exfiltration attack steps such as *pub_exf, scp_exf, sftp_exf, scp_inst*, and *sftp_inst*. The SDR values are mostly close to zero, and the ΔL values indicate late detections occurring near the average duration of each step. The few successful detections by XGB, for instance, *pub_exf* with SDR = 0.235, do not compensate for its overall low sensitivity to this class of attacks.

In Fig. 8, we show the trade-off between the latency and the selected false positive rate thresholds for the best model, XGB. This recalls the concept of RoC and precision recall curves, but with latency and false positive rate on, respectively, the y-axis and x-axis. As demonstrated in previous work [44], this curve shows a clear relation between these two metrics, such that, tolerating more false positives, the detector is faster in identifying ongoing attacker actions. The structure of the cAPTure dataset, which identifies attack-step iterations within normal traffic, enables the use of this curve to fine-tune the detector and gain deeper insight into its performance [44].

The SDR remains identical for the 1.5×10^{-6} and 3×10^{-6} FPR thresholds, with no variation in latency in all three attack subsets of Fig. 8. Therefore, the lower threshold can be selected. This choice leads to consistent improvement, reducing the false positive rate from one every 6 hours to one every 12 hours, which would be obtained with the other thresholds.

A similar reasoning applies to thresholds between 1.82×10^{-5} and 7.3×10^{-5} FPR thresholds, which exhibit the same SDR performance and no changes in latency. As before, selecting the lower threshold is preferable, as it decreases the false positive frequency to one every hour instead of every 30 or 15 minutes at higher thresholds.

In contrast, the 2.18×10^{-4} threshold differs significantly from the 1.09×10^{-3} threshold. Here, the choice depends on the acceptable false positive rate. If considering SDR and latency alone, the higher threshold (1.09×10^{-3}) is preferable, yielding a notably better SDR of 0.776 compared to 0.581, along with consistently shorter latency.

7.3. Results with time features

The results from Section 7.1 and Section 7.2 indicate that, although the models can detect attacks, their performance remains insufficient for deployment as actionable detectors.

This section evaluates the performance of XGB using the engineered statistical time features described in Section 6.4. We focus on XGB because it consistently achieves the best overall performance. Notably, XGB can detect a broader range of attack steps, including *qos_mid_dos, sftp_exf*, and *sftp_inst*, that other models fail to identify.

Payload features. We begin by analyzing the temporal features of the payload. These features enhance the overall recall of attack packets in Table 9 across all FPR thresholds, achieving the best overall performance at a threshold of 1.09×10^{3}, with R=0.78784. However, recall remains limited at the lowest thresholds.

As shown in Table 10, attack step coverage improves, with additional attacks, such as *dollar_char, qos_mid*, and *empty_conn*, now showing some level of detectability, albeit still with near-zero recall. Overall, recall increases in all attacks, except for *scp_exf* and *scp_inst*. The improved recall significantly improves attack step detectability reported in Table 11. For instance, *dollar_char* and *empty_conn* show recall values of only 0.154 and 0.007, respectively, yet their sequences are fully detected, meaning that the model can identify at least one indicative packet within each attack sequence. This highlights the strength of the sequence-based structure of the cAPTure dataset in evaluating APT

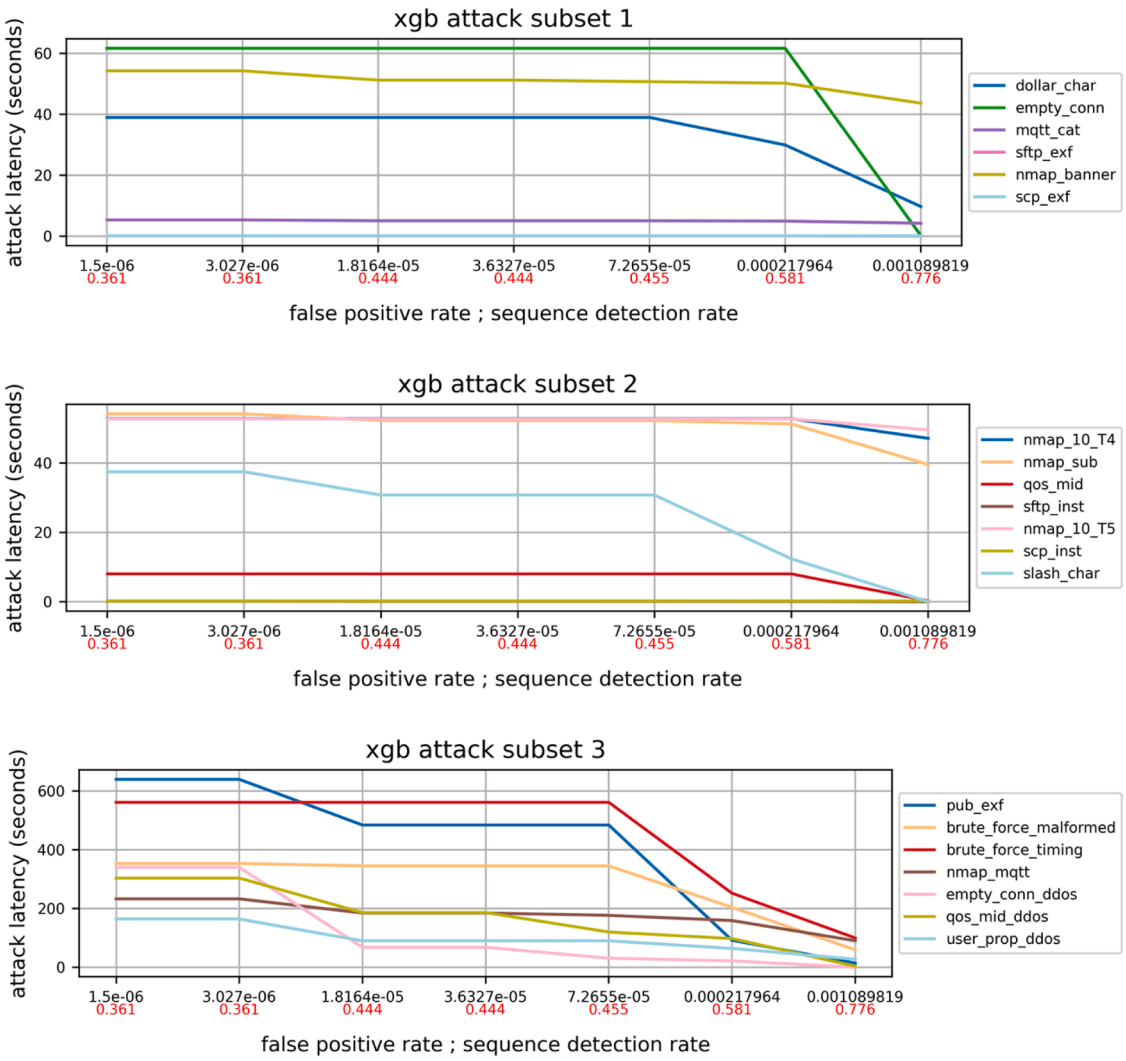


Fig. 8. The plot illustrates the trade-off between attack latency (y-axis) and false positive rate (FPR) with the corresponding SDR (x-axis), evaluated over the total attack step iterations across three attack subsets.

detectability, since recall alone may underestimate the detector's true ability to recognize attack sequences.

Packets count. As shown in Table 9, these features lead to overall weaker XGB performance at lower thresholds (1.5×10^{-6} to 1.82×10^{-5}), although some improvement is observed at higher thresholds. This decline is primarily due to poor recall of *nmap_T4_10* and *nmap_T5_10* attack steps, as highlighted in Table 10.

In terms of SDR, performance is comparable to the baseline model (0.447 against the baseline model 0.443). The modest improvement comes mainly from attacks involving many packets, such as *brute_force_timing*, *dollar_char*, and *user_prop_ddos*, while performance decreases for subtler attacks like *sftp_exf* and *sftp_inst*. Consequently, latency remains similar to that of the baseline model.

Timing features. The inter-packet timing features produce a substantial improvement in both overall and per-attack recall, as shown in Table 9 and Table 10. These features result in a *perfect* SDR across all attack step iterations in Table 11, with nearly instantaneous detection latency (0.06 seconds).

All temporal features combined. When combining all the temporal features, performance improves even further. All attacks in Table 10 achieve recall values above 0.99 across every FPR threshold, with SDR consistently equal to 1. Detection latency also decreases drastically, reaching near-instantaneous levels (1.6×10^{-5}seconds). This demonstrates the high capabilities of temporal statistical features. The high classification scores at the cost of low FPR reported in Table 9 and Table 10 for these features indicate that, when combined with non-temporal features, the detector can not only identify the onset of each attack step, evidenced by the short detection latency and perfect SDR reported in Table 11, but also accurately discriminate between attack and normal packets. However, these results must be interpreted with caution, as the detector may have overfitted to timing regularities in the dataset, a phenomenon discussed in detail in the Limitations section (Section 8.2).

7.4. Sensitivity to timing perturbations

The near-perfect detection performance observed when incorporating timing-based features raises the concern that the model may exploit temporal artifacts rather than robust behavioral information.

For this reason, we perform a sensitivity analysis to assess whether timing artifacts affect the performance of timing-related features. The XGBoost detector is trained on the original data and evaluated on the test set with a perturbed time between packets. Specifically, we apply zero-mean Gaussian jitter to packet timing and update timestamps accordingly, perturbing both benign and attack traffic and recalculating time features based on the new timestamps. Given an average time between packets in the original dataset of 67.68, we introduce four jitter levels, namely, {0.5 ms, 1 ms, 10 ms, 100 ms}, spanning from minimal

perturbations to extreme stress conditions.

We report the recall at multiple FPR thresholds in Table 12 as the timing perturbation increases.

Under small perturbations (0.5–1 ms), recall degrades but remains consistently above the XGBoost model without time features (XGB) across nearly all FPR thresholds. At 10 ms jitter, recall decreases more significantly, particularly at very low FPR. However, detection performance is still consistently improved compared to the baseline XGB detector. At 100 ms jitter, performance degrades consistently. This is expected: the introduced perturbations drastically affect the network communication behavior. Overall, the performance degradation is monotonic with increasing jitter magnitude, suggesting a smooth sensitivity to temporal patterns, which can be attributed to temporal feature dependence rather than overfitting to static artifacts.

SDR results in Table 13 show a similar trend. Under small perturbations (0.5–1 ms), most attack steps remain detectable with high SDR and latency increases. At 10 ms jitter, certain attacks show reduced detection rates, whereas generally, performance remains robust. Under extreme perturbation (100 ms), performance degrades substantially. However, some of the attacks remain partially detectable.

This suggests that the model integrates time and packet features rather than relying exclusively on timing information.

To support this analysis, we study the feature importance of the baseline XGB model and the time features under different jitter.

As reported in the upper plot in Fig. 9, the baseline XGB model relies exclusively on protocol headers and packet-level features. Differently, when timing features are introduced, structural features remain important, but with time features complementary to classification (see lower plot in Fig. 9). The combined model, therefore, integrates structural packet information and traffic dynamics. The most important feature overall is a TCP structural metric (*layer_tcp_tcp_analysis_push_bytes_sent*, gain $\approx$ 0.42), which is not a time feature. The rest of the importance is mostly attributable to timing features, but is distributed across multiple aggregation windows (0.1 s, 1 s, 60 s, 900 s, 3600 s), with no single timing feature dominating. This suggests that the model learns behavioral timing patterns rather than exploiting a single artifact or fixed timestamp offset.

Overall, the observed performance degradation under increasing jitter and the feature importance analysis suggest that the detector leverages meaningful temporal dynamics rather than overfitting to dataset-specific artifacts. At the same time, the sensitivity to substantial timing distortions highlights that robustness analysis is essential when incorporating time-based features. For this purpose, we publicly share the 4 test sets with perturbed timing in our online repository [11].

8. Conclusions and future work

8.1. Conclusions

This work presented an extended methodology for building semi-synthetic APT detection datasets tailored to industrial CPS networks, integrating realistic network traffic with simulated attack campaigns composed of multiple attack steps. By combining normal traffic with controlled APT activity, the resulting cAPTure dataset provides a detailed temporal representation of attacks aligned with the MITRE ATT&CK framework and CVE vulnerabilities.

The dataset's structure enables both traditional classification and latency-based evaluation, allowing researchers to quantify not only the accuracy of intrusion detectors but also timeliness in interrupting attack progressions. Experimental results show that standard packet-based models such as XGB and EXTRA do not achieve sufficient recall when trained on raw features, whereas temporal statistical features derived from traffic evolution substantially enhance detectability and reduce detection latency.

These findings demonstrate that incorporating time-aware features is crucial for capturing APT dynamics and highlight the importance of designing datasets that preserve the chronology of attack steps.

Furthermore, the capability to compute latency-based metrics such as SDR and ΔL serves as a meaningful proxy for assessing a system's operational performance beyond traditional classification measures. Since correctly identifying even a single packet can reveal the attacker's action, evaluation can prioritize attack-step detection rather than exhaustive packet-level accuracy, which is rarely attainable in practice, or due to overfitting to the training dataset. The intrinsic relationship between FPR and attack latency offered by the dataset structure in attack step sequences provides an effective means to adapt detector behavior to specific operational constraints and performance goals.

8.2. Limitations

8.2.1. Reproducibility of the approach

The proposed approach is generalizable to publish/subscribe-based CPS protocols beyond MQTT, provided that the system communication topology and representative real traffic traces are available. In such settings, normal behavior can be reproduced by allocating publish messages to virtual devices and replaying them according to observed timing and roles, a process that can be largely automated.

While the methodology is not tied to MQTT, publish/subscribe architectures are a particularly convenient instantiation that simplifies system replication. If publish/subscribe is not adopted, it is obviously required to generate and monitor normal traffic following the available technology. However, key aspects of the methodology, such as crafting and running attacks in isolated runs, structuring campaigns using an attack graph, and composing multiple attack paths, remain directly applicable.

8.2.2. Dataset representativeness and external validity

The dataset relies on simulated network traffic replayed from real traces within a controlled environment. While this ensures reproducibility and precise labeling, it cannot fully capture the heterogeneity and unpredictability of real industrial networks. The simulated MQTT publish/subscribe network includes 23 nodes. While representative of small

Table 12
Report recall (R) for each FPR threshold of XGBoost models without timing features (XGB), with timing features, and with timing features under Gaussian timing perturbations (σ = 0.5 ms, 1 ms, 10 ms, 100 ms) applied to the test set.

FPR threshold	XGB	XGB all time 0.5 ms	XGB all time 1 ms	XGB all time 10 ms	XGB all time 100ms	XGB all time
	R	R	R	R	R	R
1.5×10^{-6}	0.27799	0.57259	0.52117	0.05922	0.00685	0.99565
3.03×10^{-6}	0.27799	0.65009	0.57072	0.0769	0.01491	0.9976
1.82×10^{-5}	0.3338	0.74315	0.65666	0.24476	0.05331	0.99865
3.63×10^{-5}	0.33382	0.76541	0.7262	0.39087	0.09249	0.99886
7.27×10^{-5}	0.35777	0.8136	0.7458	0.408	0.12718	0.99925
2.18×10^{-4}	0.42559	0.92948	0.78656	0.53689	0.15754	0.99978
1.09×10^{-3}	0.59658	0.972665	0.87619	0.64217	0.20585	0.99995

Table 13
Report the detection performance using latency-related metrics of XGBoost models without timing features (XGB), with timing features, and with timing features under Gaussian timing perturbations (σ = 0.5 ms, 1 ms, 10 ms, 100 ms) applied to the test set.

attack step	XGB		XGB all time 0.5 ms		XGB all time 1 ms		XGB all time 10 ms		XGB all time 100ms		XGB all time	
	SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL	SDR	ΔL
brute_force_m	0.75	345.18	1	0.051	1	0.007	1	39.7	0.375	552.9	1	0
brute_force_t	0.6	561.54	1	0.035	1	0.062	1	0.04	0.8	253.57	1	0
dollar_char	0	38.95	1	0.024	1	0.061	0.873	5.1	0.079	30.91	1	0
empty_conn	0	61.62	0.903	4.794	0.774	11.99	0.129	54.8	0	61.62	1	0
empty_conn_ddos	0.9	68.05	1	0.0001	1	0.0001	0.6	184.62	0	339.9	1	0
mqtt_cat	0.048	5.08	1	0.0169	1	0.017	0.758	1.33	0.241	4.08	1	0
nmap_10_T4	1	52.84	0.916	17.66	0.833	17.93	0.75	18.16	0.75	18.218	1	1.67-e06
nmap_10_T5	1	52.77	0.957	2.38	0.957	2.408	0.957	8.95	0.872	16.182	1	0
nmap_banner	1	51.18	0.99	1.642	0.971	3.85	0.869	18.66	0.168	54.71	1	4.07-e06
nmap_mqtt	0.987	184.72	1	0.034	1	0.034	0.8	120.48	0.226	193.82	1	0
nmap_sub	1	52.29	0.964	2.985	0.928	3.941	0.785	16.66	0.071	50.627	1	0
pub_exf	0.235	484.23	1	3.559	0.823	123.7	0.235	491.2	0	639.71	1	2.3-e05
qos_mid	0	8.04	1	0.004	1	0.009	0.759	1.89	0	8.045	1	0
qos_mid_ddos	0.74	186.20	1	0.0004	1	0.002	1	3.96	0.36	204.62	1	0
scp_exf	0	0.18	1	0.0001	1	0.0004	0.936	0.05	0.031	0.179	1	0
scp_inst	0	0.19	1	0.044	1	0.067	0.526	0.116	0	0.19	1	0
sftp_exf	0.235	0.13	0.941	0.011	0.882	0.021	0.117	0.147	0	0.161	1	0
sftp_inst	0.278	0.13	1	0.007	0.888	0.035	0	0.166	0	0.166	1	0
slash_char	0.077	30.79	1	0	1	0	1	0.062	0.391	26.194	1	0
user_prop_ddos	0.75	90	1	37.12	1	43.48	0.5	281.4	1	45.35	1	16e-05
all	**0.443**	**68.6**	**0.988**	**5.865**	**0.97**	**17.31**	**0.797**	**103.9**	**0.227**	**208.4**	**1**	**1.6e-05**

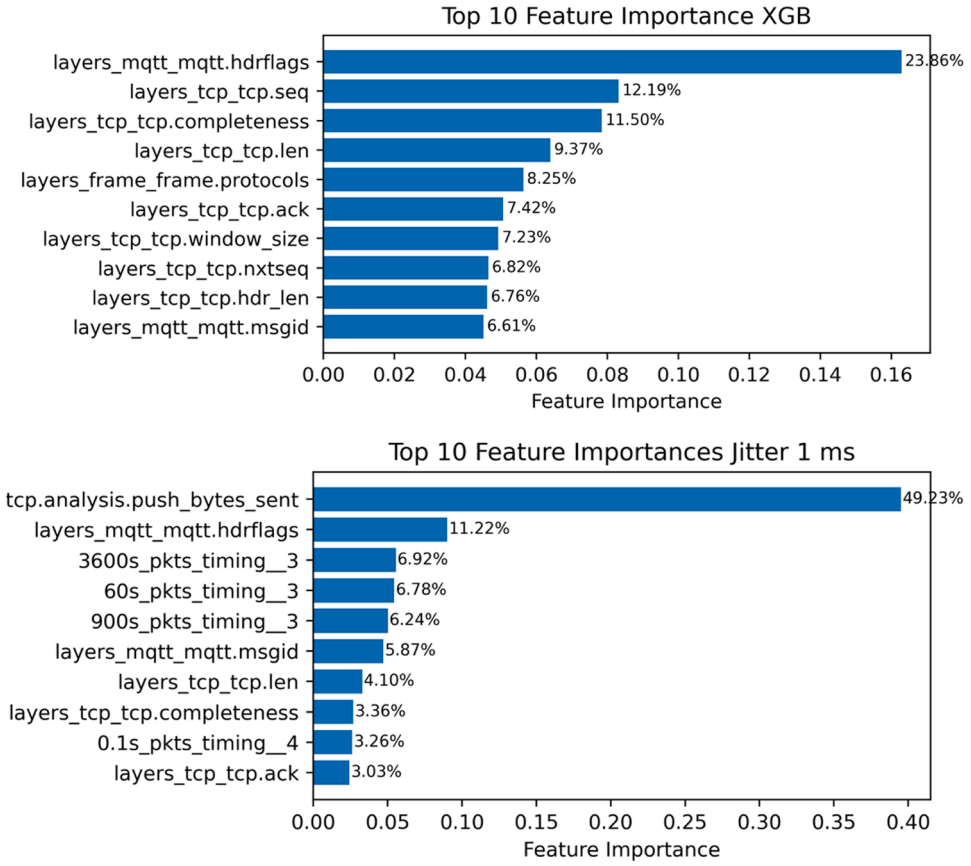


Fig. 9. Illustrates the feature importance of the baseline XGBoost (XGB) detector and the XGB model augmented with timing features under 1 ms Gaussian jitter. The x-axis represents the percentage feature importance (gain), while the y-axis lists the corresponding feature names.

to medium industrial settings, larger or heterogeneous infrastructures could exhibit different communication dynamics and attack propagation paths. Therefore, we do not intend to provide a universal dataset from which detectors can be expected to generalize directly to arbitrary industrial deployments. Instead, we provide a controlled and reproducible benchmark for latency-aware APT detection, together with a methodology and a toolset that can be replicated to construct deployment-specific datasets when representative traces and system

knowledge are available.

8.2.3. Attack sequencing and class imbalance

The APT scenarios in cAPTure are generated from an attack graph and follow a structured lifecycle, rather than being injected as isolated attacks. This provides a realistic ordering of attacker actions while preserving reproducibility and exact packet-level ground truth. Nevertheless, real APT campaigns may involve adaptive choices, longer dwell times, failed attempts, and branching behaviors that are only partially represented. The dataset intentionally includes a high percentage of attacks to support the training and evaluation of each represented attack step. This distribution should not be interpreted as an operational attack base rate. For this reason, our main evaluation fixes detector thresholds according to false-positive budgets computed from benign traffic only and reports SDR and latency rather than relying solely on aggregate metrics that are sensitive to class imbalance. As an additional tool, we provide a script that allows users to construct alternative test scenarios by inserting attack sequences into benign traces, to have a more realistic imbalance while preserving step-level labels.

8.2.4. Timing features and robustness

While the timing-based features yield seemingly *perfect* results, this performance warrants scrutiny. Such uniformly ideal metrics may indicate overfitting to the dataset's temporal patterns. While the timing between different attack steps and their durations uses different time intervals for training and test sets, the timing of the attack techniques when started remains the same in both sets. In real-world scenarios, network timing can vary significantly due to the attacker's ability to adapt their techniques. Therefore, a model achieving perfect recall and SDR on timing features alone might not generalize well beyond the controlled experimental setup.

To validate that the near-perfect performance is not driven by static temporal artifacts, we performed a jitter-based sensitivity analysis. Performance degrades gradually as timing noise increases, remaining above the baseline without time features under small and moderate perturbations, and only substantially declining under extreme distortion. Moreover, feature importance shows that structural packet features remain dominant, with timing features contributing in a distributed and complementary manner.

Overall, these results indicate that the model captures meaningful temporal dynamics rather than memorizing fixed timestamp patterns, while also highlighting the need for robustness evaluation when leveraging timing-based features.

8.2.5. Scope of the experimental evaluation

Last, we emphasize that the objective of this work is not to introduce a new APT detection algorithm, but rather to present a methodology for constructing an APT dataset tailored to industrial CPS networks and to present the cAPTure dataset. To ensure reproducibility and facilitate adoption by the research community, we rely on widely adopted, easy-to-use machine learning models. Notably, many of these approaches rely on widely used benchmark datasets such as CICIDS17 [23], CICIDS18 [23], and KDD/NSL-KDD [20]. Therefore, we believe that the cAPTure dataset can contribute to the broader evaluation and deployment of advanced APT detection solutions, including those cited below, by providing a dedicated dataset specifically designed for industrial CPS environments.

If we focus on detection mechanisms, several approaches have recently been proposed. Especially relevant to our paper's underlying machine-learning technique, the work in [45] aims to determine an optimal subset of features for effective APT description. The paper [46] exploits LSTM to enable both temporal behavior modeling and graph-based relationship modeling. The framework in [47] enhances machine learning detection by integrating six event detection modules. Similarly, MLAPT [48] operates in a multi-stage manner and is able to reconstruct APT scenarios from individual alerts. Last, to overcome the limited time windows that LSTM models can examine, the authors in [49] propose an Enhanced Graph Convolutional LSTM (EGC-LSTM) to improve the spatio-temporal correlation of APTs.

8.3. Future work

As future work, we will exploit the semi-synthetic nature of the cAPTure dataset to generate progressively stealthier APT scenarios and rigorously evaluate temporal features under more realistic adversarial conditions. Concretely, we will: (i) vary the composition of attack phases and the combinations of attack steps to produce alternative attack paths; (ii) perturb packet timings and inter-packet delays to emulate attackers that deliberately spread activity over longer periods or jitter their actions to evade timing-based detectors; these variations will let us develop and assess time-aware features while reducing the risk of overfitting to fixed temporal patterns; iii) study generalization and robustness by training detectors under constrained knowledge scenarios: for example, removing APT scenarios and, consequently, certain attack steps from the training set to simulate zero-day or unknown threats.

Altogether, this will form multiple attacker "skill levels", from novice to advanced levels, that differ in stealthiness and capabilities. Together, these extensions will stress-test temporal features and quantify how detector performance (SDR, ΔL, and the FPR–latency trade-off) degrades or improves as attacks become more capable.

Clinical trial number

Not applicable.

Informed consent

Not applicable.

Ethical approval

Not applicable.

Declaration of generative AI use

During the preparation of this work, the authors used AI exclusively to rephrase and correct text parts that had been entirely authored by the authors. The output was then revised and edited. The author(s) take full responsibility for the content of the published article.

Data availability statement

Tables: Table 1 is reproduced from our previous conference paper [50]. Table 2 includes partial data from [50] supplemented with additional attack steps from our extended APT simulation.

Figures: Fig. 3 includes partial data from [50] supplemented with additional attack steps from our extended APT simulation.

Datasets: The cAPTure data we compose in this paper is made public at [11], folder *dataset*.

Codes: The code to simulate the industrial network is in part from our previous paper [50], but has been consistently modified to run the new enhanced simulation and APT scenarios, and include an additional experimental campaign. The code to compose the dataset and to reproduce the experimental campaign, instead, is entirely new. Both are available at [11], the folder *simulation_network* for accessing files to reproduce the simulation environment, and the *experimental_campaign* folder for training and evaluating detectors.

CRediT authorship contribution statement

Tommaso Puccetti: Writing – review & editing, Writing – original

draft, Visualization, Software, Methodology, Investigation, Data curation, Conceptualization. **Simona De Vivo:** Software, Investigation, Data curation, Conceptualization. **Davide Zhang:** Writing – original draft, Visualization, Software, Investigation. **Pietro Liguori:** Supervision, Conceptualization. **Roberto Natella:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Andrea Ceccarelli:** Writing – original draft, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This paper was partially supported by the MUR PRIN 2022 project FLEGREA - Federated Learning for Generative Emulation of Advanced Persistent Threats, and by the project P2022K7ERB PRIN PNRR 2022 BREADCRUMBS funded by the Italian MUR and by POR CREO Regione Toscana 2023 -D17H24004520009 WAU -Warehouse Automation.

References

[1] ENISA threat landscape 2024 | ENISA, https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024, last accessed 2025/01/14.
[2] C.E. Aanonsen, Stuxnet, revisited (again): producing the strategic relevance of cyber operations, J. Cyber Policy 10 (2025) 68–84, https://doi.org/10.1080/23738871.2025.2492570.
[3] Cyber-attack against Ukrainian critical infrastructure | CISA, https://www.cisa.gov/news-events/ics-alerts/ir-alert-h-16-056-01, last accessed 2025/01/14.
[4] D. Cotroneo, R. Natella, S. Rosiello, A fault correlation approach to detect performance anomalies in virtual network function chains, in: 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), 2017, pp. 90–100, https://doi.org/10.1109/ISSRE.2017.12.
[5] R. Kalakoti, S. Nõmm, H. Bahsi, Federated learning of explainable AI(FedXAI) for deep learning-based intrusion detection in IoT networks, Comput. Netw. 270 (2025) 111479, https://doi.org/10.1016/j.comnet.2025.111479.
[6] Z. Chen, M. Simsek, B. Kantarci, M. Bagheri, P. Djukic, Machine learning-enabled hybrid intrusion detection system with host data transformation and an advanced two-stage classifier, Comput. Netw. 250 (2024) 110576, https://doi.org/10.1016/j.comnet.2024.110576.
[7] Z. Li, C. Huang, W. Qiu, An intrusion detection method combining variational auto-encoder and generative adversarial networks, Comput. Netw. 253 (2024) 110724, https://doi.org/10.1016/j.comnet.2024.110724.
[8] S. Myneni, K. Jha, A. Sabur, G. Agrawal, Y. Deng, A. Chowdhary, D. Huang, Unraveled — a semi-synthetic dataset for advanced persistent threats, Comput. Netw. 227 (2023) 109688, https://doi.org/10.1016/j.comnet.2023.109688.
[9] S.S. Karim, M. Afzal, W. Iqbal, D.A. Abri, Advanced persistent threat (APT) and intrusion detection evaluation dataset for Linux systems 2024, Data Br. 54 (2024) 110290, https://doi.org/10.1016/j.dib.2024.110290.
[10] B.E. Strom, A. Applebaum, D.P. Miller, K.C. Nickels, A.G. Pennington, C.B. Thomas, Mitre att&ck: Design and Philosophy, The MITRE Corporation, 2018. Technical report.
[11] cAPTure, https://drive.google.com/drive/folders/1d6PkZLMF6Nu7bsmxhAjP4wWwuLUZ2E-2, last accessed 2025/11/18.
[12] B. Stojanović, K. Hofer-Schmitz, U. Kleb, APT datasets and attack modeling for automated detection methods: a review, Comput. Secur. 92 (2020) 101734, https://doi.org/10.1016/j.cose.2020.101734.
[13] P. Engebretson, The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy, Elsevier, 2013.
[14] CVE - CVE, https://cve.mitre.org/, last accessed 2024/11/26.
[15] F. Skopik, G. Settanni, R. Fiedler, I. Friedberg, Semi-synthetic data set generation for security software evaluation, in: 2014 Twelfth Annual International Conference on Privacy, Security and Trust, 2014, pp. 156–163, https://doi.org/10.1109/PST.2014.6890935.
[16] I. Friedberg, F. Skopik, G. Settanni, R. Fiedler, Combating advanced persistent threats: from network event correlation to incident detection, Comput. Secur. 48 (2015) e5–e7.
[17] S. Ullah, J. Ahmad, M.A. Khan, M.S. Alshehri, W. Boulila, A. Koubaa, S.U. Jan, M. M. Iqbal Ch, TNN-IDS: transformer neural network-based intrusion detection system for MQTT-enabled IoT networks, Comput. Netw. 237 (2023) 110072, https://doi.org/10.1016/j.comnet.2023.110072.
[18] J.A. de Oliveira, V.P. Gonçalves, R.I. Meneguette, R.T. de Sousa, D.L. Guidoni, J.C. M. Oliveira, G.P. Rocha Filho, F-NIDS — a network intrusion detection system based on federated learning, Comput. Netw. 236 (2023) 110010, https://doi.org/10.1016/j.comnet.2023.110010.
[19] T. Puccetti, A. Ceccarelli, Detection latencies of anomaly detectors - an overlooked perspective?, in: 2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE), 2024, pp. 37–48, https://doi.org/10.1109/ISSRE62328.2024.00015.
[20] M. Tavallaee, E. Bagheri, W. Lu, A.A. Ghorbani, A detailed analysis of the KDD CUP 99 data set, in: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009, pp. 1–6, https://doi.org/10.1109/CISDA.2009.5356528.
[21] G. Creech, J. Hu, A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns, IEEE Trans. Comput. 63 (2014) 807–819, https://doi.org/10.1109/TC.2013.13.
[22] A. Shiravi, H. Shiravi, M. Tavallaee, A.A. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection, Comput. Secur. 31 (2012) 357–374, https://doi.org/10.1016/j.cose.2011.12.012.
[23] I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, ICISSp 1 (2018) 108–116.
[24] InSDN: a novel SDN Intrusion dataset | IEEE journals & magazine | IEEE Xplore, https://ieeexplore.ieee.org/abstract/document/9187858, last accessed 2024/11/13.
[25] H.K. Kim, IoT network intrusion dataset. https://ieee-dataport.org/open-access/iot-network-intrusion-dataset, 2019.
[26] M.J.M. Turcotte, A.D. Kent, C. Hash, Unified host and network data set. Data Science for Cyber-Security, WORLD SCIENTIFIC (EUROPE), 2018, pp. 1–22, https://doi.org/10.1142/9781786345646_001.
[27] Antoine-Lemay: antoine-lemay/Modbus_dataset. https://github.com/antoine-lemay/Modbus_dataset, 2024.
[28] Goh, J., Adepu, S., Junejo, K.N., Mathur, A.: A dataset to support research in the design of secure water treatment systems.
[29] Bohara, A., Thakore, U., Sanders, W.H.: Intrusion detection in enterprise systems by combining and clustering diverse monitor data.
[30] M.M. Anjum, S. Iqbal, 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, New York, NY, USA, Association for Computing Machinery, 2021, pp. 27–32, https://doi.org/10.1145/3450569.3463573.
[31] TON_IoT Telemetry dataset: a new generation dataset of IoT and IIoT for data-driven intrusion detection systems, https://ieeexplore.ieee.org/document/9189760, last accessed 2025/05/06.
[32] M. Al-Hawawreh, E. Sitnikova, N. Aboutorab, X-IIoTID: a connectivity-agnostic and device-agnostic intrusion data set for industrial internet of things, IEEE Internet Things J. 9 (2022) 3962–3977, https://doi.org/10.1109/JIOT.2021.3102056.
[33] E.C.P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, A.A. Ghorbani, CICIoT2023: a real-time dataset and benchmark for large-scale attacks in IoT environment, Sensors 23 (2023) 5941, https://doi.org/10.3390/s23135941.
[34] G. Draper-Gil, A.H. Lashkari, M.S.I. Mamun, A. Ghorbani, Characterization of encrypted and VPN traffic using time-related features, in: Proceedings of the 2nd International Conference on Information Systems Security and Privacy, Rome, Italy, SCITEPRESS - Science and and Technology Publications, 2016, pp. 407–414, https://doi.org/10.5220/0005740704070414.
[35] T. Puccetti, S. Nardi, C. Cinquilli, T. Zoppi, A. Ceccarelli, ROSPaCe: intrusion detection dataset for a ROS2-based cyber-physical system and IoT networks, Sci. Data 11 (2024) 481, https://doi.org/10.1038/s41597-024-03311-2.
[36] M.A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, H. Janicke, Edge-IIoTset: a new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning, IEEE Access 10 (2022) 40281–40306, https://doi.org/10.1109/ACCESS.2022.3165809.
[37] R. Taormina, S. Galelli, N.O. Tippenhauer, E. Salomons, A. Ostfeld, D.G. Eliades, M. Aghashahi, R. Sundararajan, M. Pourahmadi, M.K. Banks, Battle of the attack detection algorithms: disclosing cyber attacks on water distribution networks, J. Water Resour. Plan. Manag. 144 (2018).
[38] Ghiasvand, E., Ray, S., Iqbal, S., Dadkhah, S., Ghorbani, A.A.: CICAPT-IIOT: a provenance-based APT attack dataset for IIoT environment, arxiv.org/abs/2407.11278, (2024). https://doi.org/10.48550/arXiv.2407.11278.
[39] Myneni, S., Chowdhary, A., Sabur, A., Sengupta, S., Agrawal, G., Huang, D., Kang, M.: DAPT 2020-constructing a benchmark dataset for advanced persistent threats.
[40] S. De Vivo, I. Obaidat, D. Dai, P. Liguori, DDoShield-IoT: a testbed for simulating and lightweight detection of IoT Botnet DDoS attacks, in: 2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2024, pp. 1–8, https://doi.org/10.1109/DSN-W60302.2024.00014.
[41] A. Alatram, L.F. Sikos, M. Johnstone, P. Szewczyk, J.J. Kang, DoS/DDoS-MQTT-IoT: a dataset for evaluating intrusions in IoT networks using the MQTT protocol, Comput. Netw. 231 (2023) 109809, https://doi.org/10.1016/j.comnet.2023.109809.
[42] T. Chen, C. Guestrin, XGBoost: a scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, Association for Computing Machinery, 2016, pp. 785–794, https://doi.org/10.1145/2939672.2939785.
[43] T. Puccetti, A. Ceccarelli, On detection latencies of network intrusion detectors – discussion and application, Empir. Softw. Eng. 31 (2025) 23, https://doi.org/10.1007/s10664-025-10766-3.
[44] T. Puccetti, A. Ceccarelli, Detection latencies of anomaly detectors - an overlooked perspective?, in: 2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE), 2024, pp. 37–48, https://doi.org/10.1109/ISSRE62328.2024.00015.
[45] N. Saini, V. Bhat Kasaragod, K. Prakasha, A.K. Das, A hybrid ensemble machine learning model for detecting APT attacks based on network behavior anomaly

detection, Concurr. Comput.: Pract. Exp. 35 (2023) e7865, https://doi.org/10.1002/cpe.7865.
[46] C.D. Xuan, T.T. Nguyen, A novel approach for APT attack detection based on an advanced computing, Sci. Rep. 14 (2024) 22223, https://doi.org/10.1038/s41598-024-72957-0.
[47] G. Ali, S. Shah, M. ElAffendi, Enhancing cybersecurity incident response: AI-driven optimization for strengthened advanced persistent threat detection, Results Eng. 25 (2025) 104078, https://doi.org/10.1016/j.rineng.2025.104078.
[48] I. Ghafir, M. Hammoudeh, V. Prenosil, L. Han, R. Hegarty, K. Rabie, F.J. Aparicio-Navarro, Detection of advanced persistent threat using machine-learning correlation analysis, Future Gener. Comput. Syst. 89 (2018) 349–359, https://doi.org/10.1016/j.future.2018.06.055.
[49] A. Presekal, A. Ștefanov, I. Semertzis, P. Palensky, Spatio-temporal advanced persistent threat detection and correlation for cyber-physical power systems using enhanced GC-LSTM, IEEE Trans. Smart Grid 16 (2025) 1654–1666, https://doi.org/10.1109/TSG.2024.3474039.
[50] T. Puccetti, S. De Vivo, D. Zhang, P. Liguori, R. Natella, A. Ceccarelli, Creation and use of a representative dataset for advanced persistent threats detection, in: B. Gallina, M. Törngren, F. Bitsch (Eds.), Computer Safety, Reliability, and Security, Springer Nature Switzerland, Cham, 2026, pp. 51–66, https://doi.org/10.1007/978-3-032-01241-8_4.
[51] M. Sajid, K.R. Malik, A. Almogren, et al., Enhancing intrusion detection: a hybrid machine and deep learning approach, J. Cloud Comp. 13 (2024) 123, https://doi.org/10.1186/s13677-024-00685-x.
[52] C. Hazman, A. Guezzaz, S. Benkirane, et al., A smart model integrating LSTM and XGBoost for improving IoT-enabled smart cities security, Cluster Comput. 28 (2025) 70, https://doi.org/10.1007/s10586-024-04780-1.
[53] https://www.kaggle.com/code/robertlangdonvinci/time-series-classifier-keras-conv1d-lstm-dense.
[54] C. Ji, M. Du, Y. Hu, S. Liu, L. Pan, X. Zheng, Time series classification based on temporal features, Appl. Soft Comput. 128 (2022) 109494, https://doi.org/10.1016/j.asoc.2022.109494.
[55] M.G. Baydogan, G. Runger, E. Tuv, A bag-of-features framework to classify time series, IEEE Trans. Pattern Anal. Mach. Intell. 35 (2013) 2796–2802, https://doi.org/10.1109/TPAMI.2013.72.