Advances in Knowledge-Based Systems, Data Science, and Cybersecurity Received18-09-2025;Accepted19-11-2025;Published27-11-2025

Uncovering Malicious Persistence: Machine Learning-Based Detection of Windows Scheduled Tasks

Khaled Rahal k.rahal@cylab.be
Cyber Defence Lab, Royal Military Academy
Royal Military Academy
Université Libre de Bruxelles
Rue Hobbema 8, 1000 Bruxelles

Arbia Riahi a.riahi@cylab.be
Cyber Defence Lab, Royal Military Academy
Rue Hobbema 8, 1000 Bruxelles

Georgi Nikolov g.nikolov@cylab.be
Cyber Defence Lab, Royal Military Academy
Rue Hobbema 8, 1000 Bruxelles

Thibault Debatty t.debatty@cylab.be
Cyber Defence Lab, Royal Military Academy
Rue Hobbema 8, 1000 Bruxelles

Jean-Michel Dricot jean-michel.dricot@ulb.be
Ecole Polytechnique – Embedded Systems Design Security
Université Libre de Bruxelles
Avenue F.D. Roosevelt, 50 – 1050 Bruxelles

Corresponding Author: Khaled Rahal

Abstract

Advanced Persistent Threats (APT) represent a serious security concern because they carry out long-term and carefully planned attacks. While a lot of research has gone into finding ways to detect these threats; one crucial area often gets less attention, namely the persistence mechanisms that allow attackers to stay hidden and maintain access to systems over time. In this work, we investigate scheduled tasks, a widely used persistence technique in Windows environments, and analyze their role in APT operations. We conducted an in-depth study of how attackers leverage scheduled tasks to maintain stealthy access and execute malicious actions over time. We introduce Detecting APT Through Malicious Scheduled Tasks (DAPTASK), an approach that leverages Sysmon log data, Word2Vec-based feature representation, and Machine Learning (ML) classifiers to identify malicious scheduled tasks commonly used in APT persistence techniques. Our approach achieves a high detection performance, with an F1-score of 95.19%. Furthermore, we provide a labeled dataset, which can serve as a valuable resource for researchers developing APT detection methods, the dataset and the code used

Citation: Khaled Rahal et al. Uncovering Malicious Persistence: Machine Learning-Based Detection of Windows Scheduled Tasks. **Advances in Knowledge-Based Systems, Data Science, and Cybersecurity**.Research 2025;2(3):16.

Page No. 310-336

are publicly available at https://gitlab.cylab.be/cylab/daptask. Our approach enhances APT detection by addressing persistence techniques, a critical yet often neglected attack vector.

Keywords: APT, Windows Scheduled Task, Persistence, ML, Text Embeddings

1. INTRODUCTION

Nowadays, information systems are growing rapidly and becoming increasingly complicated in terms of both users and applications, which provides a favorable environment for a quick cyberattacks development. APT are an example of attacks, with a significant impact on the cybersecurity market, responsible for a heavy focus and investing into cyber security solutions. Therefore, the protection market is projected to surpass 23 billion U.S. dollars by 2028, up from the 10 billion U.S. dollars projected in 2024 [1]. The SolarWinds APT attack, attributed to APT29, is estimated to have caused losses of up to 90 million U.S. dollars [2]. Given the substantial risk associated with APT, most existing research emphasizes their detection at a holistic level, addressing the full attack lifecycle, which contradicts the nature of APT [3]. Recent studies have started to shift towards the detection of individual attack stages, such as lateral movement or command and control [4–9]. However, the cornerstone of a successful APT attack is persistence [10], which allows threat actors to persist unnoticed and advance through additional phases of their operation.

In this context, we focus on hunting Windows Scheduled Tasks (WST), a common and effective method used in APT attacks [11], know for their effectiveness, stealthy nature, and widespread use in maintaining long-term access to compromised systems. As a built-in feature of the operating system, they often go unnoticed by security tools widely used for automation and legitimate administrative operations, making them an ideal target for attackers seeking persistent footholds like APT41, APT29, Lokibot, DEADEYE [11, 12].

To address the challenge of detecting stealthy persistence mechanisms in APT attacks, we propose DAPTASK, a novel detection approach specifically designed to identify malicious WST. By combining Word2Vec-based embeddings with machine learning classifiers, DAPTASK captures subtle semantic patterns in Sysmon logs, enabling effective detection of suspicious behavior at the moment of task creation and execution which is often a key indicator of long-term system compromise.

Previously [13], we created a dataset of APT persistence techniques on Windows platforms, mapped to the MITRE ATT&CK Framework. In this work, we enhance our dataset for DAPTASK, to be more effective in detecting malicious WST, ensuring that it aligns with the unique patterns and behaviors indicative of APT activity.

This paper addresses the gap in detecting scheduled task-based persistence by the following threefold contributing:

1. Conducting an in-depth study by analyzing how WST are used as a persistence technique in APT.
2. Providing a dedicated pre-processing and labeled dataset of WST events, specifically crafted to support research in anomaly detection and facilitate malicious scheduled task identification.

3. Utilizing text embedding techniques, such as Word2Vec, along with ML classifiers to detect and classify malicious WST based on subtle patterns in their metadata.

The structure of this paper is as follows. Section 2 reviews related work on detecting persistence techniques, with a particular focus on WST. Section 3 explores the creation of WST-based persistence techniques and their role in APT operations. Section 4 presents the overall approach proposed in this work. Section 5 describes the data preparation process (enhancement, labeling, and feature engineering). Section 6 details the experimental setup and presents the results obtained from evaluating the proposed DAPTask approach. Finally, Section 7 concludes the paper and discusses directions for future research.

2. RELATED WORK

When conducting our research, we first explored the APT lifecycle from a general perspective, gradually narrowing our focus to the persistence stage, TABLE 1 summarizes various APT detection approaches, including ML, rule-based detection, provenance graph frameworks.

Several studies have proposed approaches to detect APT activities across various stages of the attack lifecycle. In [14] the authors leverage rule-based and provenance graph to identify known attack behaviors in real time by matching observed system events to predefined tactics, techniques, and procedures (TTP). The authors in [15] apply ML combined with provenance based graphs to reconstruct and visually explain attacker behavior. Provenance graph techniques have also been used to detect unusual execution patterns during initial compromise and data exfiltration phases by tracing anomalous sequences of actions [16]. More broadly, ML approaches span several stages, applying learning models to detect APT campaigns holistically [17]. Additionally, behavior mapping techniques align individual observed techniques with their corresponding APT tactics, providing structured insight into the attacker's overall strategy [18]

[4, 19] focus specifically on detecting lateral movement, tackling the challenge of identifying how attackers move laterally across networks to escalate privileges or access different systems. Similarly, [5, 6, 20] concentrate on detecting command and control (C2) activity, to uncover the communication channels connecting compromised machines with external attackers.

Meanwhile, Some research efforts have specifically addressed the persistence phase by leveraging the MITRE ATT&CK framework. The study in [21] links Windows malware samples to persistence tactics by identifying the mechanisms used to maintain access. Similarly, [7, 8] analyze various persistence methods found in Windows malware and map them to specific ATT&CK techniques. In addition, [9] proposes a taxonomy of persistence techniques aimed at facilitating the detection of previously unseen methods and guiding the appropriate defensive strategies.

Cyber Persistence Detector (CPD) [10] is a system designed to detect cyber persistence through provenance analytics, by introducing pseudo edges for connections between the persistence setup and the execution phases, this approach is rule-based and relies on pseudo-dependency edges and necessitates analyzing both the setup and execution stages to enable detection. As a result, it requires frequent updates and its effectiveness in early detection is limited.

The authors in [22] introduce novel methodologies for proactive threat hunting, leveraging ElasticSearch as a SIEM solution for real-time threat analysis. To improve cybersecurity resilience by utilizing advanced query languages, and focusing on uncovering hidden adversaries, it relies on rule-based methods and does not specifically focus on WTS.

None of these listed approaches addresses the detection of WST. While some works focus on persistence techniques, they primarily rely on rule-based methods rather than ML. This gap highlights the need for targeted research and detection strategies for WST using ML.

Authors	APT Life Cycle	Methods	Goal	Approach
[14]	APT Kill Chain	Rule-based	Real-time detection of APT campaigns	Detects APT by matching observed behaviors to TTP
[15]	C2, LM, Privilege Escalation	ML	Reconstruct and explain APT behaviors via visual attack stories	Provenance-based graphs (attacker paths maping and interpretable alerts generation)
[16]	Initial Compromise, LM, Data Exfiltration	Provenance graphs	Anomalous execution behaviors capture	Graph sketches (unusual sequences of actions tracing)
[17]	Reconnaissance, Initial Compromise, LM, Data Exfiltration	ML	APT detection using learning models	ML over several attack stages
[18]	APT Kill Chain	Behavior Map- ping	Techniques matching to a specific APT tactic	Technique-to-tactic mapping
[19]	LM	Rule-based	Forensic analysis of lateral movement	Predefined rules (analysis of Sysmon logs)
[4]	LM	ML	Identify suspicious internal movements	Supervised models training on Sysmon data
[20]	C2	Behavior-based	Provide real-time detection of C2	Multi-Agent System for APT Detection (MASFAD)
[5]	C2	Rule-based	Recognize covert communication	Patterns and metadata analysis (potential C2 behavior identification)
[6]	C2	ML	Detect botnet traffic	Behavioral patterns recognition
[21]	Persistence	Malware Analysis	Identify persistence capabilities	Link malware behaviors to MITRE techniques
[7, 8]	Persistence	Behavior Mapping	Understand persistence mechanisms in malware samples	Map persistence behaviors to ATT&CK techniques
[9]	Persistence	Analytical framework	A taxonomy of persistence techniques	Detailed classification construction
[10]	Persistence	Rule-based	Link persistent threats to their operational footprints	Persistence setup correlation with later execution activities
[22]	Persistence	Rule-based	Proactive threat hunting in Windows environments	ElasticSearch SIEM (registry modifications and user account creation tracking)

Table 1: APT Detection Approaches

3. SCHEDULED TASKS AS A PERSISTENCE TECHNIQUE

In this section, we focus specifically on WST a commonly exploited technique due to their nature as a built-in feature of the Windows operating system, widely used for legitimate automation. This functionality is often abused by APT actors to achieve stealthy and resilient persistence. Their deep integration into system operations allows malicious tasks to evade traditional security mechanisms, enabling adversaries to maintain long-term access and execute payloads with minimal visibility. By targeting WST, we address a high-impact and frequently overlooked technique, aiming to improve early stage detection of APT activity.

3.1 Scheduled Tasks Operation

The strategic use of this method can be particularly advantageous for attackers because it allows them to craft their malicious tasks to appear benign, using naming conventions and execution times that mimic legitimate operations. For example, an attacker might schedule a task to run during peak usage hours, blending it with normal system activity to avoid detection. This stealthy approach underscores the need for organizations to implement robust monitoring and alert mechanisms that can differentiate between legitimate and potentially malicious task executions. To better understand the critical role of WST within the APT lifecycle, the Kimsuky APT group uses scheduled tasks not only to maintain persistence but also to regularly trigger malware execution every 41 minutes [23]. This mechanism enables the malware to connect to its C2 servers for receiving commands and exfiltrating data, highlighting how WST acts as a reliable and stealthy scheduler that supports continuous adversary control over compromised systems.

MITRE ATT&CK profiles around 181 different threat actor groups, highlighting the various techniques they use to maintain persistence in compromised systems. Among these techniques, WST (T1053.005) stands out, with 52 groups relying on it roughly 29% of the total [11]. This usage makes it a prime candidate for deeper exploration, as understanding how adversaries use this technique can improve detection and response capabilities. The following section outlines the various methods available within Windows systems to create scheduled tasks, providing insight into how these mechanisms are leveraged as part of persistence strategies.

3.1.1 Methods for creating scheduled tasks

As illustrated in FIGURE 1, WST can be created using various methods, each offering different levels of automation and flexibility [24]:

- GUI: the graphical interface allows users to manually configure tasks by specifying triggers (e.g., system startup, user login or a timed schedule), actions (e.g., running a script or application), and conditions (e.g., only when on AC power).
- Command: administrators can use powershell commands to automate task creation (FIGURE 2). Example:

```
$action = New-ScheduledTaskAction  -Execute "notepad.exe"
```

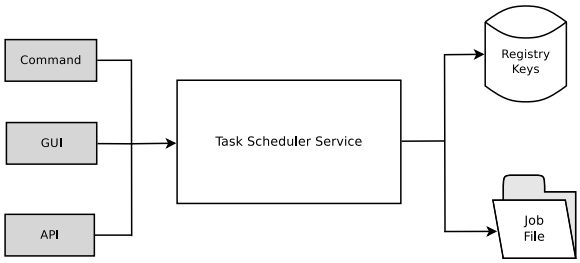


Figure 1: Task Creation

```
$trigger = New-ScheduledTaskTrigger -AtStartup
Register-ScheduledTask -TaskName  "MyTask"  -Action $action -Trigger $trigger  -User "SYSTEM"
```

- API: creation of programmatic tasks in languages such as C++,C#, etc. This method is often exploited by malware to establish persistence while bypassing standard security monitoring.

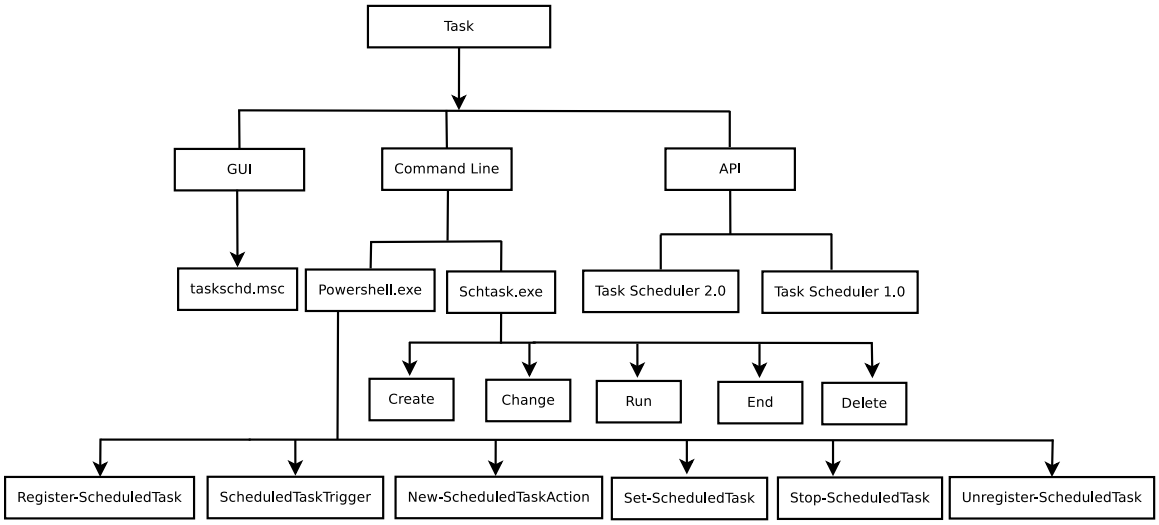


Figure 2: Example of Task Scheduler Creation methods

3.1.2 Storage and structure of scheduled tasks

Windows registry entries: maintain critical metadata for WST, allowing efficient tracking and execution. Specifically, task-related information are stored in the following registry paths: These registry keys contain task metadata (execution history, configuration parameters and Security Descriptors (SD)).

- `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache`
- `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\Tasks`

Task configuration files: in addition to the registry, WST are also represented as XML files stored in:

`C:\Windows\System32\Tasks` ; and `C:\Windows\Tasks`

Each task has a corresponding XML file that defines its configuration settings, triggers, execution conditions, and associated actions. To ensures easy parsing, modification, and forensic analysis.

3.1.3 Evasion techniques

Registry keys: involve directly modification the Windows Registry to create or alter WST. Attackers become able to avoid detection by bypassing the standard event logs related to task creation, which would typically trigger alerts for security teams.

Delete SD: a Windows data structure that defines the access control and security settings of an object, including its owner, permissions. When a SD is deleted, the associated object (file, directory, or registry entry) may have its security settings compromised or removed, resulting in the application of default permissions [25]. This can lead to many issues related to:

- Loss of access control,
- Security vulnerabilities,
- Auditing and compliance issues

Command obfuscation: Using Base64 encoding or PowerShell obfuscation techniques, making it challenging for security systems to identify the payload.

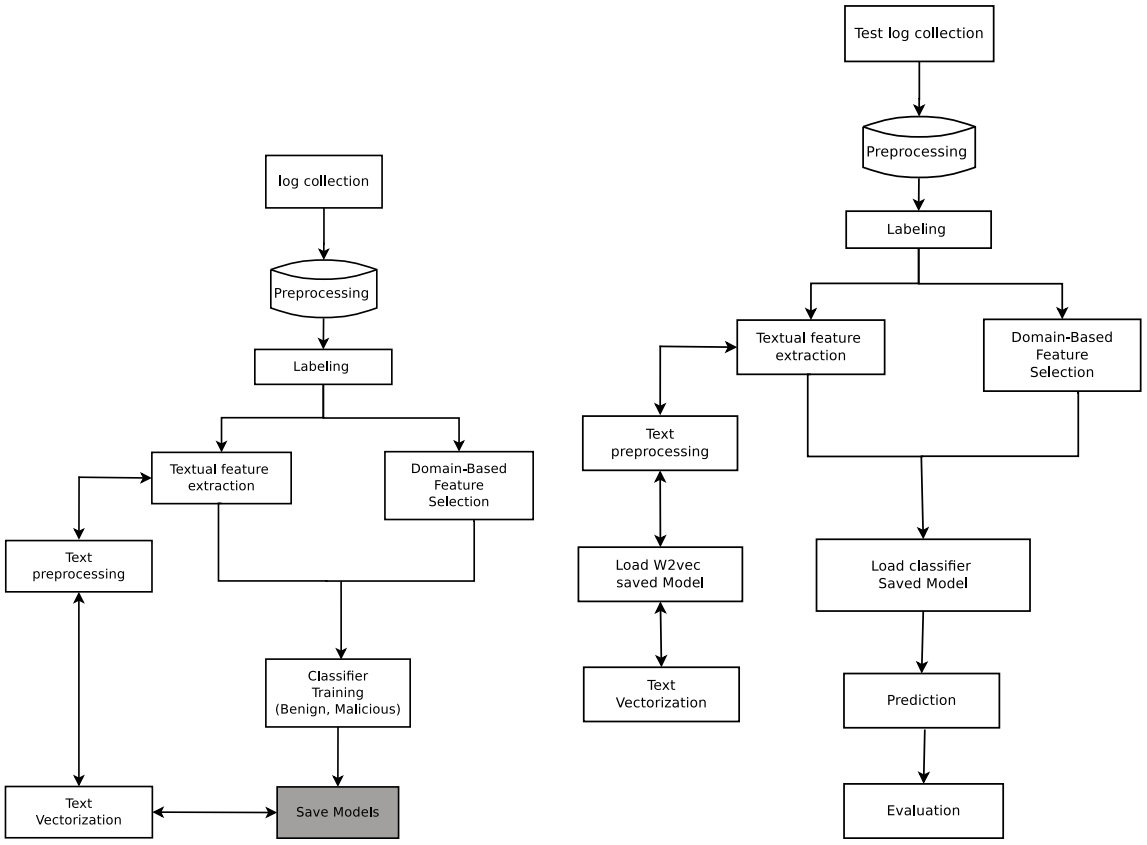
Manipulating task parameters: WST run at very specific times or only under particular conditions that are difficult for traditional monitoring to identify.

Backdoor tasks: exploit existing legitimate WST by injecting additional malicious actions, such as executing scripts or launching malware, once the legitimate task is completed. This technique allows adversaries to maintain persistence while evading detection by blending into normal system operations.

4. PROPOSED APPROACH

Our approach, illustrated in FIGURE 3, for detecting malicious WST.

The proposed pipeline begins with the collection and preprocessing of events logs, where .evtx files are converted into .csv format to enable downstream analysis. These logs are then labeled as benign or malicious using ground truth data sources, such as expert annotations or controlled attack simulations.



(a) Training Architecture (b) Evaluation/Prediction Architecture

Figure 3: Proposed architecture: (a) training phase and (b) evaluation phase using test logs.

Next, the pipeline performs feature extraction. Textual attributes like OriginalFileName, Image, and CommandLine are extracted and preprocessed. A Word2Vec model is trained on the training set to vectorize these textual features, capturing semantic relationships between terms. In parallel, structured features are selected based on domain knowledge to reflect behaviors indicative of WST persistence techniques.

These textual and domain-based features are combined into a unified feature set. A classifier is then trained to distinguish between benign and malicious activity.

For evaluation and prediction, a new dataset composed of logs not seen during training is processed through the same feature extraction and vectorization steps. The trained Word2Vec and classifier model are loaded to perform inference. This ensures that the classifier is evaluated on unseen data, simulating real-world deployment scenarios. The system outputs predictions and evaluation metrics that reflect its generalization performance.

4.1 Technical Choices

We use Word2Vec, a natural language processing (NLP) technique, to convert the text found in events logs into numerical vectors. Since these logs contain a lot of unstructured text, Word2Vec transforms words into vectors such that words with similar meanings have similar vector representations. This allows us to preserve the context and relationships between words. Once the text is vectorized, we feed with crafted feature into a classifier, which can then learn patterns and make accurate predictions based on the combination of contextual meaning and domain based feature selection. Both the classifier and the Word2Vec model are saved for use in future predictions. This pipeline ensures that the system can reliably distinguish between benign and malicious activity in Windows environments.

4.1.1 Log collection

While Windows Event Logs capture fundamental WST activity, they often lack the visibility needed to uncover stealthier techniques employed by advanced attackers. In contrast, Sysmon [26] offers enhanced insights by monitoring and recording detailed events related to WST. TABLE2 shows how specific Sysmon Event IDs aid in detecting malicious WST by offering an overview of their role in monitoring process execution, file activities, registry modifications, and inter-process interactions, which help uncover persistence mechanisms and stealthy behaviors, enabling security teams to detect and analyze malicious behaviors more effectively. Unlike standard logs, Sysmon can:

- Capture process execution in real time (Event ID 1).
- Capture commandline arguments and their parent processes (Event ID 1).
- Detect registry modifications related to scheduled tasks (Event ID 13,12).
- Monitor dll loading (Event ID 7).

4.1.2 Text embeddings

Word2Vec [27] is an unsupervised learning algorithm that transforms words (or short sequences) into numerical vectors while preserving their semantic relationships, captures contextual meaning and similarities between different terms (e.g., commands, process names). We employed Word2Vec, a neural network-based method for learning continuous vector representations of words based on their context. There are two main architectures in Word2Vec [27]:

- CBOW (Continuous Bag of Words): predicts a target word from its surrounding context words.
- Skip-gram: predicts surrounding context words given a target word.

In our scenario, the extracted textual fields often contain rare or domain-specific terms (e.g., executable, task names, script commands). Based on studies [27, 28], Skip-gram performs better

Event ID	Name	Goal
1	*Process Creation*	detect anomalous task executions via process creation and command-line arguments
2	*A process changed a file creation time*	Identify timestomping through changes in file creation timestamps
7	*Image loaded*	DLL loading
10	*ProcessAccess*	Identify suspicious inter-process interactions and privilege escalation attempts
11	*FileCreate*	dropped payloads or scripts executed via scheduled tasks
12	*RegistryEvent (Object create and delete)*	identify registry value created
13	*RegistryEvent (Value Set)*	identify registry value modifications
15	*FileCreateStreamHash*	reveal hidden cleanup behavior
17	*PipeEvent (Pipe Created)*	uncover covert C2 channels

Table 2: Sysmon Event IDs with Names and Goals for Detecting Malicious WST

than CBOW in capturing meaningful representations for infrequent terms, making it well-suited for cybersecurity logs where critical indicators may not appear frequently. Moreover, it can learn richer semantic relationships even with limited data.

The Skip-gram model works by learning to guess the words that appear around a specific target word, with the goal of increasing the chances of correctly identifying those nearby words [29]:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-c\leq j\leq c\\ j\neq 0}}\log P(w_{t+j}\mid w_t) \tag{1}$$

where:

- T is the total number of words,
- c is the size of the context window,
- w_t is the target word,
- w_{t+j} are the surrounding context words.

4.1.3 ML classifier

Selecting an appropriate machine learning classifier is a crucial step that depends on the nature of the data being analyzed. In our case, the dataset is derived from Sysmon logs, which capture detailed and often intricate system activity. These logs exhibit complex interdependencies among features, making it essential to use models that can effectively capture such rich and nuanced patterns.

To address this, we focus on classifiers that excel in modeling complex feature interactions, such as decision trees, random forests, and gradient-boosting machines [30]. These models are well-suited for detecting subtle, high-dimensional behavioral patterns. Using these advanced classifiers, we aim to improve the system's ability to identify sophisticated threats with greater accuracy and greater resilience, especially in realistic environments where data patterns are rarely straightforward.

4.2 Persistence Dataset Overview

Since publicly available datasets focused on persistence techniques are limited, we build upon our previous work [13], where we developed a dataset centered around persistence techniques. This dataset includes realistic user profiles, simulated attack scenarios, and raw Windows log data. To ensure comprehensive coverage, we employed a diverse array of persistence techniques during data generation. Many of these simulations draw inspiration from known threat groups, such as APT29, and incorporate a wide range of techniques commonly used to achieve persistence. These techniques are mapped to the MITRE ATT&CK framework and captured through dedicated logging tools.

While the dataset covers how APT maintain long-term access, our focus shifts to a specific WST technique. To better align the dataset with our research goals, we introduce an enhancement phase aimed at integrating more WST samples, including evasion techniques used by threat actors. In the next section, we detail how we enhanced the dataset, labeled key behaviors, and engineered features to ensure the model could effectively learn to detect WST-based persistence.

5. DATA ENHANCEMENT AND FEATURE ENGINEERING

Detecting malicious WST is particularly challenging due to their dual-use nature. While administrators and legitimate applications rely on them for automation and system maintenance, adversaries exploit the same functionality to evade detection, execute payloads at scheduled intervals, and maintain persistence on compromised systems. This overlap between benign and malicious use cases complicates the differentiation between normal system activity and threat actor behavior. As discussed in Section 4.2, we expand the dataset's coverage of scheduled task-based persistence by using advanced simulation tools, including GhostTask [31], Cobalt Strike [32], SharpPersist [33], and ScheduleRunner [34], each chosen for their ability to replicate realistic and evasive techniques used by threat actors.

5.1 Dataset Labeling

As illustrated in FIGURE4, we labeled the datasets through several steps, by matching command-line inputs to known malicious activity. However, due to the complexity of command-line arguments, often containing special characters, varying structures, and escape sequences, we employed fuzzy similarity matching [35]. This technique allows us to compare command-line entries even when they are not exact matches, accounting for minor variations, typos, or encoded values. Specifically, we applied fuzzy string-matching algorithms to measure the similarity between command lines found in the dataset and those in the ground truth.

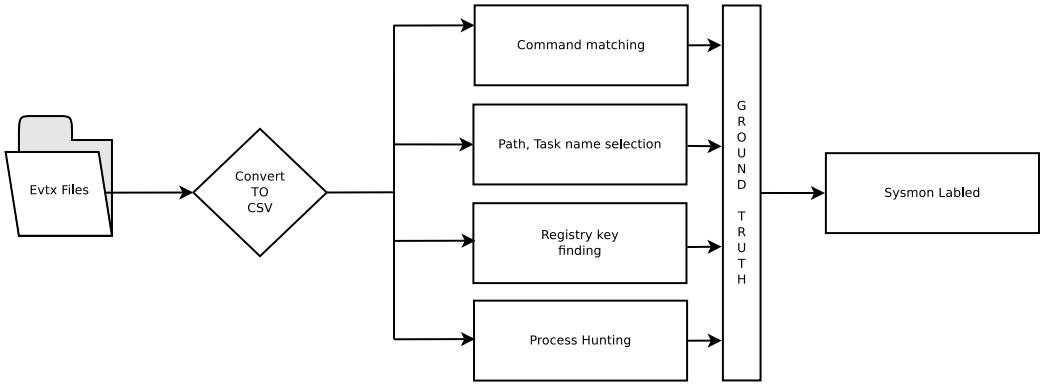


Figure 4: Dataset labeling

Beyond command-line analysis, we investigated paths, task name and registry keys associated with WST. We also tracked the execution of specific processes linked to adversarial techniques as mention in section 5.

No	Feature	Value
1	Name	Microsoft-Windows-Sysmon
3	Opcode	0
4	Keywords	0x8000000000000000
5	Correlation	0
6	Channel	Microsoft-Windows-Sysmon/Operational
7	State	Started or 0
8	Version	15,15
9	SchemaVersion	4.9 or 0
10	Configuration	sysmon.xml
11	ConfigurationFileHash	SHA256(sysmon.xml)

Table 3: Example of Excluded Features and Values

Datasets	**Total Samples**	**Benign**	**Malicious**
TAPD	77,294	73,689	3605
TAPD-V	11,148	10,440	708

Table 4: Overview of the Datasets

Since automated detection alone can introduce false positives or overlook subtle variations, we manually reviewed the labeled entries [36] to make sure they accurately reflected malicious behavior. By combining automated tools with expert human judgment, we were able to boost both the accuracy and trustworthiness of our dataset labels. To further refine our method, we carried out a structured feature selection process to pinpoint the most important attributes for telling apart malicious and benign WST activity.

5.2 Data Preprocessing

Sysmon telemetry contains many textual features such as Image, CommandLine, and ParentCommandLine which provide rich semantic context but also introduce high variance. To address this, we use our enhanced datasets named Task-based APT Persistence Dataset (TAPD) and we implement two parallel feature selection pipelines as show in FIGURE 3a, grounded in domain knowledge and a detailed understanding of how WST operate within the Windows environment and we include the aforementioned textual fields to capture semantic and contextual information. To evaluate model performance and generalizability, we introduce a second dataset, Task-based APT Persistence Dataset Validation (TAPD-V) , constructed using a distinct set of machines, configurations, and temporal conditions. TAPD-V contains unseen data, including novel scheduled tasks used by APT such as RedCurl, APT29, Kimsuky, and APT32, none of which appear in the TAPD dataset. This setup enables us to rigorously assess how well models trained can generalize to new and previously unobserved malicious behaviors. The feature construction process is consistent across both datasets, facilitating meaningful comparison. The TABLE 4, presents an overview of the two datasets used in our study, including the total number of samples in each dataset and the distribution between benign and malicious instances. Irrelevant and redundant features were removed to reduce dimensionality and improve computational efficiency. The list of excluded features is presented in TABLE 3.

5.3 Feature Engineering

Feature engineering refers to the process of transforming raw data into meaningful features that can improve the performance of ML models.

5.3.1 Domain knowledge-based feature selection

We incorporated domain expertise to identify key indicators of persistence, such as Sysmon Event IDs, rare command-line parameters, and process execution frequency linked to known attack techniques. The TABLE5 (Feature Expertise-Based Approach) present the features utilized in the TAPD dataset along with their use cases.

Feature construction: we crafted several composite features like:

- **dll_loading**: this binary feature captures the presence of DLL loading events based on Sysmon EventID 7, which indicates that a process has loaded a DLL. Malicious tasks often use DLL injection or reflective loading techniques.
- **cmd_length**: the total number of characters in the command line string. Extremely short or excessively.
- **cmd_token_count**: the number of arguments (tokens) parsed from the command line. This reflects the complexity or intent of the command (e.g., multi-flag PowerShell commands).

Feature Name	Description
Expert Knowledge-Based Approach	
dll_loading	Indicates dynamic-link library (DLL) loading activity based on Sysmon Event ID 7.
cmd_length	Length (in characters) of the command line string.
cmd_token_count	Number of tokens or arguments parsed from the command line.
cmd_entropy	Shannon entropy of the command line, measuring randomness or obfuscation.
TotalProcessExecutionCount	Total number of times the process has executed system-wide.
HourlyProcessExecutionCount	Number of times the process executes within a one-hour window.
HourlyExecutionCountDelta	Change in execution frequency between consecutive hourly intervals.
AvgTFIDFCommandRarity	Average TF-IDF score reflecting the rarity of command-line terms.
CommandExecutionCount	Number of times the exact command line has been executed.
NormalizedCommandRarity	Frequency of the command normalized across all observed executions.
ProcessTreeDepth	Depth of the process in the parent-child execution tree.
Text-Based Approach	
CommandLine	Raw command used to launch the process, often containing parameters and flags.
ParentCommandLine	Command line used by the parent process, providing contextual linkage.
Image	Full path to the executable that was launched.
ParentImage	Full path of the parent process's executable.
OriginalFileName	Name embedded in the binary at compile-time, often used for identification.

Table 5: Features Utilized in the Expert Knowledge-Based and Text-Based Approaches

- **cmd_entropy**: measures the randomness of the command line using Shannon entropy [37]:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{2}$$

where $p(x_i)$ is the empirical probability of character x_i in the string. Higher entropy values may indicate encoded, obfuscated, or malicious command lines.

- **TotalProcessExecutionCount (TPEC)**: cumulative count of how many times a given executable has run across all data. High values could indicate persistence mechanisms or automation.

- **HourlyProcessExecutionCount (HPEC)**: measures how frequently a process runs per hour.

- **HourlyExecutionCountDelta (HECD)**: the change in hourly execution frequency between consecutive hours:

$$\Delta h = f_t - f_{t-1} \tag{3}$$

where f_t is the execution frequency at hour t. Sudden increases may indicate anomaly or task injection.

- **AvgTFIDFCommandRarity (ATFIDF)**: this feature estimates how semantically rare or unusual a command-line string. Each command line is treated as a sequence of overlapping character 3-grams, and rarity is quantified using TF-IDF [38].

- **CommandExecutionCount (CEC)**: counts how many times a specific command line string has been executed across all observations. Rare commands are often of greater interest.

- **NormalizedCommandRarity (NCR)**: while ATFIDF captures how rare a command-line string appears, TF-IDF inherently assigns higher scores to uncommon sequences. However, this becomes problematic in the context of scheduled tasks, where some malicious commands although syntactically rare may be executed frequently (e.g., every hour), artificially inflating their TF-IDF scores. To mitigate this, we normalize the rarity score by the command's execution frequency:

$$\mathrm{NCR}(c) = \frac{\mathrm{ATFIDF}(c)}{\log(1 + \mathrm{CEC}(c))} \tag{4}$$

- **ProcessTreeDepth**: represents how deep the process is in the process execution tree. A depth of 1 indicates a child of the root process, while deeper levels may suggest indirect or stealthy process creation chains.

To effectively use Sysmon logs for ML-based detection, it's essential to identify key text-based features and apply appropriate methods for converting them into numerical representations

5.3.2 Text-Based feature extraction

Given the importance of textual data in detecting malicious WST behavior, the TAPD dataset incorporates both domain-driven feature selection and textual feature extraction as shown in TABLE 5. Specifically, we leverage text embedding techniques to derive semantic features from key Sysmon log attributes, such as :

- Image: full path of the executed process.
- Command Line: The complete command used to execute a process.
- Parent Command Line: complete parent command used to execute a process.
- OriginalFileName: executable file that retains its original name from compilation, unaffected by renaming, ensuring its true identity.

5.4 Evaluation Metrics

For better evaluatingng the performance of ousolution,on, we use the following evaluation metrics:

Accuracy: Measures the proportion of correctly classified instances among all instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

Precision: The percentage of true positives among all predicted positives.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

Recall: measures how many actual threats were correctly identified as threats (i.e., the ability to correctly identify malicious tasks).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

F1 score: The harmonic mean of precision and recall, useful for balancing both.

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

The False Positive Rate (FPR) refers to how often something that's actually harmless gets wrongly flagged as a threat. It is defined as:

$$\text{FPR} = \frac{FP}{FP + TN} \tag{9}$$

The Matthews Correlation Coefficient (MCC) is a balanced measure of classification quality, especially useful for imbalanced datasets. Consider true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{10}$$

Due to the imbalance in our dataset, we prioritize F1-score and Matthews Correlation Coefficient (MCC) as our primary evaluation metrics [39], ensuring a robust assessment of model performance

6. EXPERIMENTS AND RESULTS

In this section, we present the application of various ML classifiers to detect malicious WST using Sysmon logs.

6.1 Experiments

We evaluate the performance of two distinct approaches:

The baseline approach relies solely on structured features (e.g., cmd_entropy , dll_loading, TPEC, HPEC) and excludes any form of textual data processing or embeddings. It acts as a baseline for evaluating detection performance when there's no extra context from text-based attributes.

Our proposed approach is an enhanced method that incorporates both structured and textual features (e.g., Image, CommandLine), with the textual components transformed into vector representations using Word2Vec embeddings. For both experiments, We used 70% of the datasets for training and the remaining 30% for testing to evaluate the model's performance. Additionally, we applied feature scaling using a StandardScaler to normalize the numerical attributes and ensure consistent input for the classifier.

After testing our approach, we selected the best classifier based on the F1 score and MCC metrics. We saved the model using Joblib, along with the scaler and the trained Word2Vec model. These

saved components are then used during the validation step with the TAPD-V dataset to mimic real-world deployment and evaluate the model's performance.

6.1.1 Experiment 1

In this experiment, we evaluated the performance of several ML classifiers using the TAPD dataset. The classifiers, discussed in section 4.1.3 include:

- Decision Tree (DT): a simple, interpretable model that divides the data into subsets based on the characteristics of the model, making decisions based on these splits [40].
- Random Forest (RF): a straightforward, easy to understand model that splits the data into groups based on certain features, and makes decisions at each step using those splits [41].
- XGBoost: a gradient boost algorithm that excels in handling structured data and can model complex relationships between features [42].
- CatBoost (CB): a gradient boosting algorithm optimized for categorical data, offering high accuracy and minimal parameter tuning [43].
- ExtraTrees (ET): an ensemble method using randomized decision trees to improve variance reduction and model robustness [44].
- K-Nearest Neighbors (KNN): a non-parametric method that classifies new data points based on the most common class among their nearest neighbors [45].
- LightGBM (LGB): a gradient boosting framework optimized for handling large-scale datasets, offering high performance, stability, and a leaf-wise tree growth approach for enhanced efficiency [46].
- Support Vector Machine (SVM): a supervised learning model that constructs hyperplanes in high-dimensional space to separate data points of different classes [47].

  TABLE 6 summarizes the optimized hyperparameters employed for the various machine learning models. The `scale_pos_weight` parameter, where applicable, is dynamically determined from the training data using the ratio of negative to positive instances to address class imbalance.

Model	Hyperparameters
XGBoost	n_estimators=200, max_depth=6, learning_rate=0.1, subsample=0.8, colsample_bytree=0.8, scale_pos_weight=scale_pos_weight, random_state=42
RandomForest	n_estimators=200, max_depth=20, min_samples_split=5, min_samples_leaf=2, class_weight='balanced', random_state=42
LightGBM	n_estimators=200, max_depth=20, learning_rate=0.05, num_leaves=31, subsample=0.8, colsample_bytree=0.8, is_unbalance=True, random_state=42
ExtraTrees	n_estimators=200, max_depth=20, min_samples_split=5, min_samples_leaf=2, class_weight='balanced', random_state=42
DecisionTree	max_depth=15, min_samples_split=10, min_samples_leaf=5, class_weight='balanced', random_state=42
CatBoost	iterations=500, learning_rate=0.05, depth=6, l2_leaf_reg=3, scale_pos_weight=5, verbose=0, random_state=42,scale_pos_weight=scale_pos_weight
KNN	n_neighbors=7, weights='distance'
SVM	kernel='rbf', C=10, gamma='scale', class_weight='balanced', probability=True, random_state=42

Table 6: Optimized Hyperparameters for ML Models

6.1.2 Experiment 2

We apply Word2Vec to transform textual features extracted from Sysmon logs into dense vector representations, effectively capturing contextual relationships. These semantic embeddings are then combined with handcrafted behavioral features to create a comprehensive representation of each event. TABLE 7 summarizes the hyperparameters used to train the Word2Vec model. The embedding dimension (`vector_size`) was set to 100 to balance semantic richness and computational efficiency. A context window size of 5 was chosen to capture nearby contextual relationships. Words occurring fewer than three times (`min_count` = 3) were ignored to reduce noise from rare terms. The model was trained using the skipgram architecture (`sg` = 1), which is more effective for capturing representations of rare words. Training was parallelized across four worker threads, and the model was trained for ten epochs to ensure convergence without overfitting.

Parameter	**Value**
vector_size	100
window	5
min_count	3
sg	1 (skip-gram)
workers	4
epochs	10

Table 7: Word2Vec Hyperparameters

6.1.3 Experiment 3

In this experiment, we evaluate the robustness and portability of our approach by validating it on a new dataset. After training on Dataset WST , we saved both the Word2Vec embedding model (used for representing textual fields such as CommandLine and Image) and the trained classifier and the scaler. These pre-trained models were then loaded and directly applied to the new dataset TAPD-V without any retraining or fine-tuning.

This setup reflects a realistic deployment scenario, where models are trained once and reused in different environments. By keeping the preprocessing and inference pipeline consistent, we assess how well the learned representations and decision boundaries generalize to unseen scheduled task activity.

6.2 RESULTS

6.2.1 Experiment 1

As shown in TABLE 8, Random Forest achieves the best overall performance, with the highest F1-Score (0.9246), indicating a strong balance between precision (0.9122) and recall (0.9373).

It also maintains a low false positive rate (FPR: 0.0043) and false negative rate (FNR: 0.0627), demonstrating both high accuracy and robustness.

XGBoost exhibits the highest recall (0.9687) among all models and a strong F1-Score (0.8987), though its precision (0.8381) is notably lower than that of Random Forest. This implies that XGBoost is more sensitive in identifying positive cases but at the cost of generating more false positives (FPR: 0.0089) as shown in FIGURE 6.

Extra Trees and LightGBM both show solid performance, with F1-Scores of 0.8791 and 0.8705, respectively. Extra Trees provides a slightly better balance of precision (0.8291) and recall (0.9354) compared to LightGBM, which prioritizes recall (0.9801) over precision (0.7830). LightGBM's extremely low FNR (0.0199) highlights its sensitivity, though it tends to produce more false positives (FPR: 0.0129).

CatBoost also favors recall (0.9744) over precision (0.7726), yielding an F1-Score of 0.8618. While slightly lower than Extra Trees and LightGBM in F1, CatBoost still performs well in minimizing false negatives (FNR: 0.0256).

KNN demonstrates high precision (0.9080) but suffers from lower recall (0.8063), which impacts its F1-Score (0.8541). This suggests KNN is more conservative in classifying positives, likely leading to more false negatives (FNR: 0.1937).

Decision Tree emphasizes recall (0.9516) but has the lowest precision among the tree-based methods (0.7203), resulting in a modest F1-Score (0.8200) and the highest FPR (0.0176) among them.

Finally, SVM underperforms across all metrics, with the lowest F1-Score (0.6192) and precision (0.4673), along with the highest FPR (0.0497). Although its recall is relatively high (0.9174), the imbalance between precision and recall, as well as the overall low accuracy, make it the least suitable model in this comparison.

Algorithm	Accuracy	Precision	Recall	F1-Score	MCC	FPR	FNR
RandomForest	0.993057	0.912200	0.937322	0.924590	0.921045	0.004292	0.062678
XGBoost	0.990082	0.838127	0.968661	0.898678	0.896080	0.008900	0.031339
ExtraTrees	0.988313	0.829125	0.935423	0.879072	0.874709	0.009171	0.064577
LightGBM	0.986761	0.783005	0.980057	0.870519	0.869680	0.012920	0.019943
CatBoost	0.985812	0.772590	0.974359	0.861823	0.860845	0.013643	0.025641
KNN	0.987494	0.908021	0.806268	0.854125	0.849250	0.003885	0.193732
DecisionTree	0.981025	0.720345	0.951567	0.819967	0.818909	0.017573	0.048433
SVM	0.948769	0.467344	0.917379	0.619231	0.633966	0.049738	0.082621

Table 8: Performance of ML algorithms on TAPD dataset without Text features

6.2.2 Experiment 2

As shown in TABLE 9, the integration of text-based features—such as CommandLine, ParentCommandLine, Image, ParentImage, and OriginalFileName—encoded using Word2Vec significantly enhances the performance of all evaluated machine learning models.

XGBoost achieves the best overall performance with the highest F1-Score (0.9886), driven by strong precision (0.9849) and recall (0.9924). This demonstrates its excellent ability to correctly identify both positive and negative cases, while maintaining a very low false positive rate (FPR: 0.0007) and false negative rate (FNR: 0.0076) as shown in FIGURE6.

LightGBM closely follows with an F1-Score of 0.9863. It maintains high precision (0.9812) and recall (0.9915), along with one of the lowest FPRs (0.0009), making it a highly reliable option as well.

Random Forest and CatBoost also exhibit strong performance with F1-Scores of 0.9788 and 0.9762, respectively. Notably, CatBoost achieves the highest recall (0.9943), which is particularly advantageous for minimizing false negatives, though this comes with a slightly higher FPR (0.0020).

KNN achieves a balanced F1-Score of 0.9630, with solid precision (0.9621) and recall (0.9639). However, it has the highest FNR (0.0361) among the top-performing models, suggesting a slightly reduced sensitivity to true positives.

SVM and Extra Trees show respectable F1-Scores of 0.9564 and 0.9392, respectively. However, SVM suffers from a higher FPR (0.0038), which may lead to an increased number of false alerts in high-precision applications.

Decision Tree ranks lowest among all models in terms of F1-Score (0.8671) and exhibits the highest FPR (0.0135). Despite its high recall (0.9820), its lower precision (0.7763) suggests a tendency to over-predict positives, making it less suitable for applications where precision is critical.

These improvements clearly demonstrate the value of incorporating semantic-rich text embeddings. Word2Vec successfully transforms raw textual fields into dense vector representations that capture contextual meaning, enhancing model discrimination capability.

Based on this evaluation, we built our final model DAPTASK using the XGBoost classifier in conjunction with Word2Vec embeddings. Both the trained model and the embedding vectors were saved using Joblib for efficient deployment and reuse.

6.2.3 Experiment 3

As shown in TABLE 10, the DAPTASK model based on the XGBoost classifier and enriched with both crafted and textual features embedded via Word2Vec demonstrates strong generalization capabilities on the unseen TAPD-V dataset.

The model achieves a high F1-Score of 0.9519, indicating an excellent balance between precision and recall. Its precision (0.9819) suggests that the vast majority of the alerts it flagged as malicious

Algorithm	Accuracy	Precision	Recall	F1-Score	MCC	FPR	FNR
XGBoost	0.998965	0.984920	0.992403	0.988647	0.988113	0.000723	0.007597
LightGBM	0.998749	0.981203	0.991453	0.986301	0.985661	0.000904	0.008547
RandomForest	0.998059	0.971910	0.985755	0.978784	0.977794	0.001355	0.014245
CatBoost	0.997801	0.958791	0.994302	0.976224	0.975248	0.002033	0.005698
KNN	0.996636	0.962085	0.963913	0.962998	0.961237	0.001807	0.036087
SVM	0.995903	0.925400	0.989554	0.956402	0.954845	0.003795	0.010446
ExtraTrees	0.994178	0.893654	0.989554	0.939162	0.937454	0.005602	0.010446
DecisionTree	0.986330	0.776276	0.981956	0.867086	0.866589	0.013462	0.018044

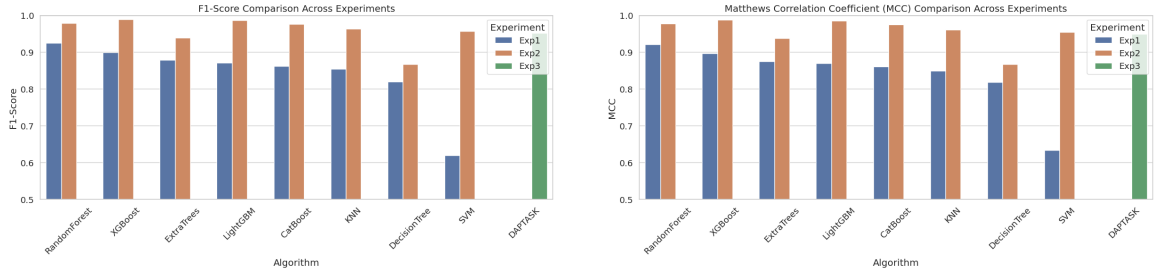
Table 9: Performance of ML algorithms on TAPD dataset with crafted and text features

were indeed correct, while its recall (0.9237) shows its effectiveness in identifying a large proportion of true threats. The MCC of 0.9493 further confirms the robustness of the classifier, reflecting a strong correlation between predicted and actual labels, even in the presence of class imbalance.

Furthermore, the model exhibits very low error rates, with a false positive rate (FPR) of 0.0011 and a false negative rate (FNR) of 0.0762, indicating minimal misclassifications of benign or malicious events, respectively as shown in FIGURE6. These results confirm that DAPTASK not only excels in predictive performance but also maintains reliability in operational scenarios, making it suitable for real-world deployment in detecting persistent threats.

Our Approach	Accuracy	Precision	Recall	F1-Score	MCC	FPR	FNR
DAPTASK	0.994079	0.981981	0.923728	0.951965	0.949317	0.001149	0.076271

Table 10: Performance of DAPTASK on TAPD-V dataset

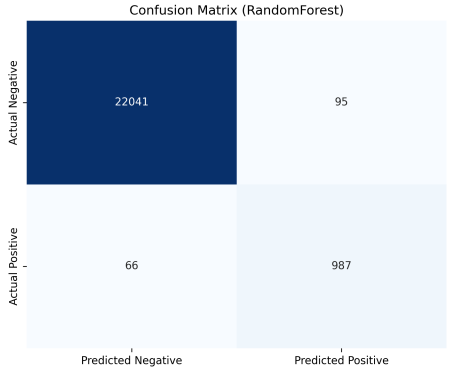


(a) F1-scores

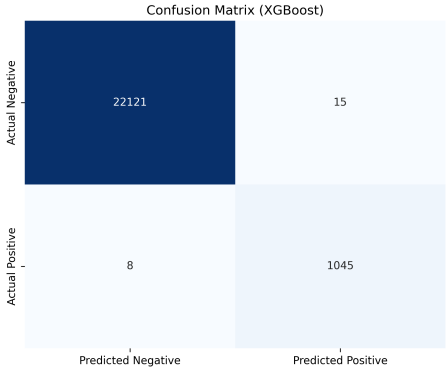
(b) Matthews Correlation Coefficient (MCC)

Figure 5: Comparison of Classification Quality: F1-score and MCC Across Algorithms

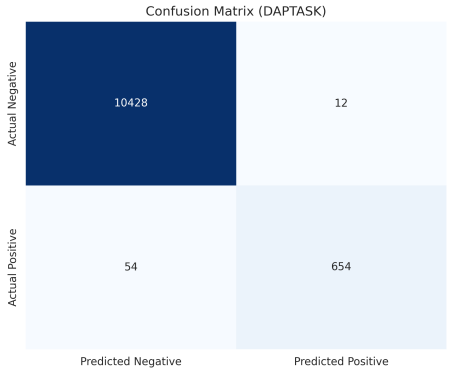
Figure 5 presents the performance comparison across different models, including our proposed model DAPTASK. The results highlight a significant improvement in Experiment 2, where the integration of text embeddings into the pipeline contributed to better generalization. This demonstrates the effectiveness of incorporating semantic information from textual features in enhancing detection capabilities.



(a) RandomForest EXP-1



(b) XGBOOST EXP-2



(c) DAPTASK EXP-3

Figure 6: Confusion Matrices for Experiments 1, 2 and 3

6.3 Results Explainability and Validation

The combination of Word2Vec and XGboost present higher performance results, but it is not always evident to interpret the decisions taken by the model. The application of Explainable Artificial Intelligence (XAI) [48] provides insight into the key features that influence the detection of anomalies in WST, allowing us to identify the most relevant factors that contribute to the model's predictions. One major problem often encountered when using ML algorithms, is the trade-off between Accuracy and Interpretability. The more accurate a model, the higher the complexity and lower the interpretability [48]. Through the use of model explainability, there is a higher understanding of how the model functions and if needed, what changes need to be made to enhance accuracy and stability. Both Word2Vec and XGBoost generate intermidiate results and visualizations, offering the means to evaluate the decisions taken.

One important aspect of model explainability is determining feature importance score [49] and how those features influenced the decision. In the case of DAPTASK, this is accomplished through the combination of Word2Vec and XGBoost. Word2Vec performs hierarchical clustering to determine similarity between various textual terms encountered in log files, command lines, etc., further creating similarity vectors, as shown in FIGURE 7. This serves to link together events, that at first glance, are not connected as they may have happened in different parts of the logs. These

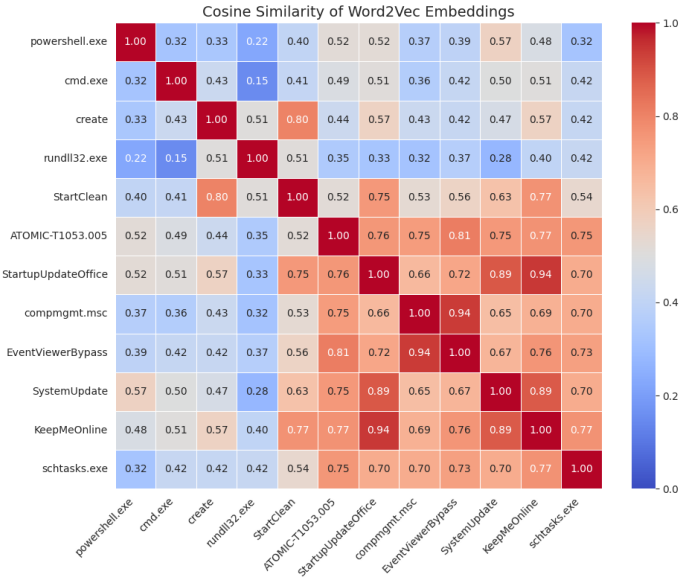


Figure 7: Cosine Similarity generated by Word2Vec, representing similarity vector between terms

vectors are then used to enrich the features used by XGBoost, as shown previously in FIGURE3. The cosine similarity heat map serves to quickly identify features of interest and then determine how they are used through the use of the XGBoost feature importance bar chart shown in FIGURE 8. These visualizations add a degree of explainability, quite useful when debugging and evaluating the fusion features of DAPTASK. This approach correctly follows the principle of applying Visual Analytics to increase explainability and interpretability, without a detriment to accuracy [50].

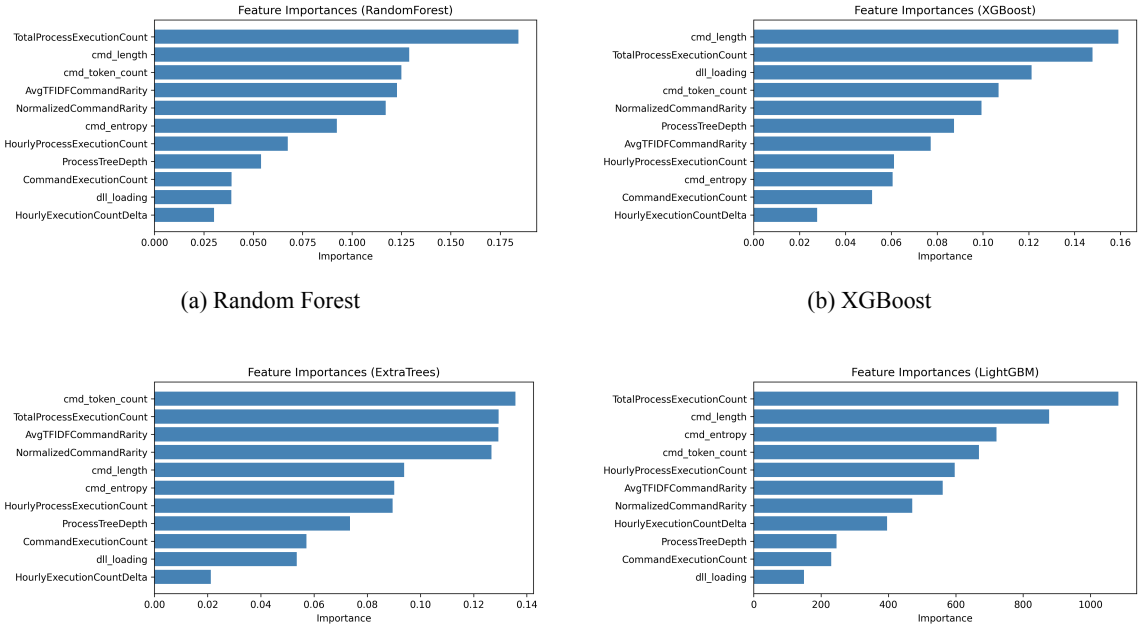
6.4 Comparison With State-Of-The-Art

In comparison to existing state-of-the-art approaches, our work focuses specifically on WST as a persistence mechanism, while frameworks such as CPD [10] are rule-based and designed to detect a broad range of persistence techniques. CPD reports a reduction in the average false positive rate (FPR) by 93%, but it does not compute or explicitly provide the FPR value. In contrast, our proposed method achieves an FPR of 0.001, demonstrating a significantly lower rate of false alarms.

Moreover, [22] ElasticSearch-oriented frameworks do not explicitly address WST persistence detection and generally lack quantitative performance metrics for comparison. This highlights a sparse research landscape concerning the detection of persistence mechanisms based specifically on WST, underlining the novelty and contribution of our approach.

7. CONCLUSION AND FUTURE WORK

In this work, we explored the application of machine learning models by combining Word2Vec-based text embeddings with crafted features to enhance the detection of WST used as a persistence



(a) Random Forest

(b) XGBoost

(c) Extra Trees

(d) LightGBM

Figure 8: Top 4 feature importance visualizations using expert knowledge-based selection on the TAPD dataset.

mechanism. Our approach captures both the semantic context of command-line inputs and domain-specific behavioral patterns, leading to improved detection of malicious scheduled tasks. The results demonstrate gains in identifying adversary tactics, showing that the integration of text embeddings and handcrafted features can reveal subtle indicators of compromise within system logs. Furthermore, the effectiveness of our method was validated on newly collected datasets, confirming its robustness, adaptability, and generalization to unseen environments and attack scenarios.

Rather than replacing current tools, our approach enhances them by filling in the gaps, especially the persistence phase making the overall detection system stronger across the APT lifecycle.

In the future, we aim to further explore the practicality of deploying the proposed system in real-time enterprise environments, with particular attention to evaluating its computational cost and latency. Additionally, we plan to develop a rule-based detection component to enable direct comparison with our solutions, Moreover, we intend to design a deobfuscation module to enhance the detection of command obfuscation techniques employed by adversaries, and integrate large language models (LLMs) with other persistence mechanisms in the detection pipeline, harnessing their ability to understand complex patterns in system logs at a deeper level. This integration is expected to enhance the sophistication of analysis and improve the detection of subtle traces left by adversaries within the system.

Another promising direction for future research is exploring the relationship between persistence techniques and C2 communication. By analyzing how specific persistence mechanisms are linked to C2 activity, we can enhance our ability to identify compromised systems and detect APTs more

effectively. Gaining deeper insight into these connections will help us better understand adversary behavior and improve the detection system's responsiveness to evolving threats.

References

[1] https://www.statista.com/statistics/497945/advanced-persistent-threat-market-worldwide/

[2] https://coverlink.com/case-study/solarwinds-supply-chain-cyberattack/

[3] Malik V, Khanna A, Sharma N, Nalluri S. Advanced Persistent Threats (Apts): Detection Techniques and Mitigation Strategies. International Journal of Global Innovations Solutions. (IJGIS) 2024.

[4] Smiliotopoulos C, Kambourakis G, Barbatsalou K. On the Detection of Lateral Movement Through Supervised Machine Learning and an Open-Source Tool to Create Turnkey Datasets From Sysmon Logs. Int J Inf Secur. 2023;22:1893-1919.

[5] Maffeis S, London I, Alageel A. Detecting APT Malware Command and Control Over HTTP(s) Using Contextual Summaries; 2025. Arxiv preprint: https://arxiv.org/pdf/2502.05367.

[6] Mannikar R, Di Troia F. Enhancing Botnet Detection in Network Security Using Profile Hidden Markov Models. Appl Sci. 2024;14:4019.

[7] Oosthoek K, Doerr C. Sok: AttTechniques and Trends in Windows Malware; 2019:406-425.

[8] Lee G, Shim S, Cho B, Kim T, Kim K. Fileless Cyberattacks: Analysis and Classification. ETRI J. 2020;43:332-343.

[9] Villalón-Huerta A, Marco-Gisbert H, Ripoll-Ripoll I. A Taxonomy for Threat Actors' Persistence Techniques. Comput Secur. 2022;121:102855.

[10] Liu Q, Shoaib M, Rehman MU, Bao K, Hagenmeyer V, Hassan WU. Accurate and Scalable Detection and Investigation of Cyber Persistence Threats. arXiv preprint arXiv: https://arxiv.org/pdf/2407.18832.

[11] https://attack.mitre.org/

[12] https://learn.microsoft.com/en-us/windows/win32/taskschd/task-scheduler-start-page

[13] Rahal K, Riahi A, Debatty T. Dataset of Apt Persistence Techniques on Windows Platforms Mapped to the MITRE AttCK Framework. 2025;17-24.

[14] Milajerdi SM, Gjomemo R, Eshete B, Sekar R, Venkatakrishnan VN. Holmes: Real-Time Apt Detection Through Correlation of Suspicious Information Flows. IEEE Symposium on Security and Privacy (SP), USA, 2019;1137-1152.

[15] Anjum M, Iqbal S, Hamelin B. Anubis: A Provenance Graph-Based Framework for Advanced Persistent Threat Detection. 2021. Arxiv Preprint: https://arxiv.org/pdf/2112.11032

[16] Han X, Pasquier T, Bates A, Mickens J, Seltzer M. UNICORN: Run-Time Provenance-Based Detector for Advanced Persistent Threats. In: Proceedings of the 2020 Network and Distributed system security symposium (NDSS). Internet Society; 2020.

[17] Al-Saraireh J, Masarweh A. A Novel Approach for Detecting Advanced Persistent Threats. Egypt Inform J. 2022;23:45-55.

[18] Zou Q, Sun X, Liu P, Singhal A. An Approach for Detection of Advanced Persistent Threat Attacks. Computer. 2020;53:92-96.

[19] Smiliotopoulos C, Barmpatsalou K, Kambourakis G. Revisiting the Detection of Lateral Movement Through Sysmon. Appl Sci. 2022;12:7746.

[20] GNikolov G, Mees W. Detection of Previously Unknown Advanced Persistent Threats Through Visual Analytics With the MASFAD Framework. International Conference on Military Communications and Information Systems (ICMCIS). 2023:1-10.

[21] Gittins Z, Soltys M. Malware Persistence Mechanisms. Procedia Comput Sci. 2020;176:88-97.

[22] Bhardwaj A, Bharany S, Almogren A, Ur Rehman A, Hamam H. Proactive Threat Hunting to Detect Persistent Behaviour-Based Advanced Adversaries. Egypt Inform J. 2024;27:100510.

[23] https://securityaffairs.com/163265/apt/north-korea-kimsuky-apt-uses-messenger.html

[24] https://learn.microsoft.com/en-us/windows/win32/taskschd/task-scheduler-start-page.

[25] https://www.microsoft.com/en-us/security/blog/2022/04/12/tarrask-malware-uses-scheduled-tasks-for-defense-evasion/?msockid=3ecf7f98fc2064fb19df69fcfd2665a3

[26] https://learn.microsoft.com/fr-fr/sysinternals/downloads/sysmon.

[27] Mikolov, Tomas Corrado, G.s Chen. Kai Dean. Efficient Estimation of Word Representations in Vector Space. 2013:1-12.

[28] McCully GA, Hastings JD, Xu S, Fortier A. Bi-Directional Transformers vs. Word2vec: Discovering Vulnerabilities in Lifted Compiled Code. In: Cyber Awareness and Research Symposium (CARS). IEEE; 2024:1-8.

[29] Mikolov T, Sutskever I, Chen K, Corrado Gs, Dean J. Distributed Representations of Words and Phrases and Their Compositionality. Adv Neural Inf Process Syst. 2013;26.

[30] Grinsztajn L, Oyallon E, Varoquaux G. Why Do Tree-Based Models Still Out Perform Deep Learning on Typical Tabular Data? Adv Neural Inf Process Syst. 2022;35:507-520.

[31] https://github.com/netero1010/GhostTask.

[32] https://www.cobaltstrike.com/.

[33] https://github.com/mandiant/SharPersist.

[34] https://github.com/netero1010/ScheduleRunner.

[35] Bosker HR. Using Fuzzy String Matching for Automated Assessment of Listener Transcripts in Speech Intelligibility Studies. Behav Res Methods. 2021;53:1945-1953.

[36] Arp D, Quiring E, Pendlebury F, Warnecke A, Pierazzi F, et al. Dos and Don'Ts of Machine Learning in Computer Security. In31st USENIX Security Symposium (USENIX Security 22) 2022:3971-3988.

[37] Saraiva P. On Shannon Entropy and Its Applications. Kuwait J Sci. 2023;50:194-199.

[38] Das M, Kamalanathan S, Pja A. A Comparative Study on TFIDF Feature Weighting Method and Its Analysis Using Unstructured. 2023.

[39] Diallo R, Edalo C, Awe O. Machine Learning Evaluation of Imbalanced Health Data: A Comparative Analysis of Balanced Accuracy. MCC, and F1 score; 2024.

[40] Rokach L, Maimon O. Decision Trees; 2005:165-192.

[41] Cutler A, Cutler D, Stevens J. Random Forests. 2012:157-175.

[42] Chen T, Guestrin C. Xgboost: A Scalable Tree Boosting System. InProceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016:785-794.

[43] Dorogush AV, Ershov V, Gulin A. Catboost: Gradient Boosting With Categorical Features Support. arXiv preprint arXiv: https://arxiv.org/pdf/1810.11363.

[44] Chauhan P, Atulkar M. Selection of Tree Based Ensemble Classifier for Detecting Network Attacks in IOT. 2021:770-775.

[45] Guo G, Wang H, Bell D, Bi Y, Greer K. Knn Model-Based Approach in Classification. InOTM Confederated International Conferences" On the Move to Meaningful Internet Systems".2003:986-996.

[46] Ke G, Meng Q, Finley T, Wang T, Chen W, et al. Lightgbm: A Highly Efficient Gradient Boosting Decision Tree. Advances in neural information processing systems. 2017;30.

[47] Evgeniou T, Pontil M. Support Vector Machines: Theory and Applications. 2001;2049:249-257.

[48] Ali S, Abuhmed T, El-Sappagh S, Muhammad K, Alonso-Moral JM, et al. Explainable Artificial Intelligence (XAI): What We Know and What Is Left to Attain Trustworthy Artificial Intelligence. Inf Fusion. 2023;99:101805.

[49] Bhatt U, Xiang A, Sharma S, Weller A, Taly A, et al. Explainable Machine Learning in Deployment. In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency. New York, USA: ACM; 2020:648-657.

[50] Alicioglu G, Sun B. A Survey of Visual Analytics for Explainable Artificial Intelligence Methods. Comput Graph. 2022;102:502-520.