ML-based detection of APT via Windows registry as a persistence technique

Master's Thesis

Submitted by

Ghita BENNOUNA

In fulfillment of the requirements for the degree of

Master of Science in Industrial Engineering, specialization in Computer Science

Academic Year 2025–2026

Acknowledgments

The author would like to express sincere gratitude to the supervisor, Ing. Khaled Rahal for his guidance, expertise, and constructive feedback throughout the course of this research. His insights and continuous support were instrumental in shaping both the technical direction and the scientific rigor of this work. The author also wishes to thank Ing. Quentin Lurkin and Pr. Ing. Thibaut Debatty for their close supervision of the project and for providing valuable feedback that contributed to improving the quality of the study.

The author further extends sincere thanks to Dr. Ing. Arbia Riahi and Dr. Ing. Charles Beumier for accepting to review this report prior to submission and for providing valuable suggestions that helped improve its clarity and overall quality.

The author also acknowledges the contributions of the research community, as well as the developers of open-source tools and publicly available datasets that made this research possible. Finally, appreciation is extended to colleagues and peers for their discussions, encouragement, and support during the completion of this work.

Abstract

The rapid growth in both the volume and sophistication of cyberattacks over the last few decades has significantly increased the risks faced by governments, financial institutions, and large enterprises. Among these threats, Advanced Persistent Threats (APTs) pose a particularly critical challenge due to their emphasis on long-term persistence, covert espionage, and data exfiltration rather than immediate system disruption. While initial intrusion attempts are often detected, the persistence mechanisms that enable attackers to maintain continuous access frequently remain hidden for extended periods, substantially amplifying both financial and reputational damage.

Existing persistence detection approaches primarily rely on rule-based mechanisms and manual analysis performed by Security Operations Center (SOC) teams. However, the high volume of logs and alerts generated in large-scale environments can overwhelm analysts, leading to alert fatigue and reduced detection effectiveness. Consequently, these methods often suffer from high false-positive and false-negative rates, limiting their effectiveness against stealthy and evolving adversaries.

This work [7] focuses on the detection of persistence techniques within the Windows Registry, a frequently exploited component of the Windows operating system. We analyze registry-based persistence mechanisms used to maintain long-term access and investigate the application of machine learning (ML) techniques to improve detection accuracy. To this end, we construct a dataset derived from Windows Registry keys modifications and use it to train and evaluate multiple ML models, including Logistic Regression, Random Forest, LightGBM, XGBoost, and neural network–based approaches such as Multilayer Perceptrons (MLP). The objective of this study is to assess the effectiveness of ML-based detection methods and contribute to more robust defensive capabilities against stealthy, long-term system compromises.

Experimental results demonstrate strong detection performance, achieving an F1-score of 0.88 for the XGBoost model, as well as an F1-score of 0.87 for the Random Forest model.

Requirements specification (Cahier des charges)

This work is structured around the simulation of APT attacks targeting the Windows registry under Microsoft Windows environments. These simulations are conducted within a controlled laboratory setting using virtual machines running Windows 10 and Windows 11. Several persistence techniques are implemented, including T1112, T1547, and T1546, in order to generate realistic malicious activity. System activity is monitored and collected through Sysmon, forming the basis for subsequent data analysis.

The collected event logs are exported from the original EVTX format to Parquet and CSV formats to facilitate processing. A dedicated dataset is then constructed and labeled to distinguish between benign and malicious behaviors. Data preprocessing steps are applied, followed by feature engineering to extract relevant characteristics for ML analysis. The resulting dataset is used to design and train artificial intelligence (AI) models for binary classification tasks, with Random Forest and XGBoost among the selected algorithms. Model performance is evaluated using standard metrics.

Finally, the trained models are validated and compared using external datasets to assess their generalization capabilities. The evaluation is conducted using the same set of performance metrics to ensure consistency. Statistical tests are then performed to determine whether observed differences in model performance are statistically significant.

Contents

List of Figures

List of Tables

1 Introduction

The growing sophistication of APTs presents substantial challenges for cybersecurity teams, particularly within large, high-value environments such as governmental institutions and major enterprises. Unlike opportunistic or commodity attacks, APT campaigns are characterized by their strategic objectives, long-term operational timelines, and focus on high-impact targets. A defining property of APTs is their ability to maintain persistence within a compromised system, remaining active for extended periods while evading detection and continuously supporting the attacker's mission.

Within the context of the Cyber Kill Chain, a framework introduced by Lockheed Martin [5] to describe the sequential phases of targeted intrusions, persistence is typically associated with the Installation stage. Persistence refers to the set of techniques that allow the adversary to re-establish or maintain control after system reboots, user logoffs, privilege changes, or software updates. These mechanisms frequently exploit legitimate operating system components and services, making them difficult to distinguish from benign activity. As a result, well-crafted persistence techniques often bypass conventional security tools, enabling the attacker to operate stealthily over prolonged periods. This prolonged foothold significantly amplifies the operational, strategic, and economic impact of APT intrusions.

Numerous APT groups now conduct highly targeted operations aimed at system disruption, espionage, data exfiltration, or financial gain, and the growing frequency and sophistication of cyberattacks targeting governments, critical infrastructures, financial institutions, and private organizations represents a significant challenge for defenders.

Historical incidents illustrate the severity of persistence-enabled attacks. The Stuxnet worm, uncovered in 2010 by a Belarus antivirus company, infiltrated supervisory control and data acquisition (SCADA) systems and caused substantial damage to the Iranian nuclear program by manipulating programmable logic controllers (PLC). These PLCs were responsible for centrifugation of Uranium at a certain specific speed. However, the malware made them run at a much higher speed. At its peak, the Stuxnet worm infected approximately 200 000 systems worldwide, with an estimated 60 percent of infections occurring within Iran [15]. More recently, the Lazarus Group has been linked to several large-scale cryptocurrency thefts, including the WazirX Exchange breach in July 2024, when the group stole 235 million dollars, and the DMM Bitcoin attack in Japan, which resulted in a 308 million dollars theft [4]. These cases highlight how persistent access enables prolonged exploitation and high-impact operations.

Despite the deployment of security policies, endpoint protection tools, monitoring systems, and continuous training efforts, maintaining full visibility over user activities and potential intrusion attempts remains difficult. Recent years have been marked by a surge in large-scale cyberattacks targeting governments, critical infrastructures, banks, corporations, and political organizations.

Current detection approaches, including rule-based systems, Security Operations Center (SOC) monitoring, Endpoint Detection and Response (EDR) tools, intrusion detection/prevention systems, and SIEM-based analytics, provide valuable defenses but remain susceptible to false positives and false negatives. As attackers become more adept at hiding malicious activity within legitimate processes, conventional detection techniques face increasing limita-

tions.

This challenge is reflected in recent malware statistics. A study conducted by AV-TEST reported that in 2023, more than 90 percent of all malware detections targeted Windows systems [3], with over one billion instances recorded worldwide. Windows-based persistence locations are especially attractive to attackers because they enable automatic execution at user logon or system startup, allowing malicious components to survive system reboots and user logoffs. The Windows Registry, in particular, stores configuration data and system-critical executables that are implicitly trusted by the operating system, while startup folders contain programs designated to run during system initialization. Although access to these locations is generally restricted, attackers who obtain elevated privileges can insert, modify, or replace entries to establish long-term persistence while appearing as legitimate system components.

These observations motivate our focus on ML–based detection of persistence mechanisms through Windows Registry–related artifacts. In this work, we propose a supervised learning approach to identify persistence techniques employed by APTs. Leveraging recent advances in AI, the objective is to develop a detection model capable of identifying persistence indicators with improved accuracy while reducing both false-positive and false-negative rates.

To this end, a large-scale dataset comprising several hundred thousand event logs was constructed using data collected from Sysmon on Microsoft Windows systems. The dataset includes a combination of benign daily user activity and malicious behavior specifically targeting Windows Registry keys. A carefully designed preprocessing pipeline was applied to the dataset, beginning with labeling based on a manually constructed ground truth, followed by data cleaning and feature engineering to extract meaningful characteristics for model training.

Multiple ML algorithms were trained on the resulting dataset, including Logistic Regression as a baseline model, ensemble tree-based methods such as Random Forest, XGBoost, and LightGBM, as well as a neural network model based on a Multilayer Perceptron (MLP). These models were evaluated to assess the accuracy and relevance of their predictions and probability estimates with respect to the labeled training data. Performance was measured using a set of commonly adopted evaluation metrics.

Based on these metrics, the best-performing models were selected and further validated using a publicly available dataset containing a mixture of APT techniques, including persistence through Windows Registry manipulation, for which a dedicated ground truth was constructed. This validation step ensures that the pretrained models are capable of generalizing to unseen data and detecting real-world persistence mechanisms, even when certain techniques, registry keys, or locations were not encountered during either the training or testing phases. Ultimately, this research aims to evaluate the effectiveness of learning-based approaches in enhancing the detection of stealthy, long-term intrusions in modern Windows environments.

The remainder of this paper is organized as follows. Section 2 reviews the related work and outlines the motivation for this study. Section 3 introduces the Windows Registry and explains its role as a persistence mechanism. Section 4 describes the proposed methodology in a detailed and technical manner. Section 5 presents and discusses the experimental results. Section 6 reports and analyzes the validation of the proposed models using external datasets. Finally, Section 7 concludes the paper with a discussion of the implications of

the findings and directions for future work.

Artificial intelligence–based tools were employed as support during the revision of the manuscript to improve language clarity, grammar, and stylistic consistency, and to assist in code review for troubleshooting. The scientific content and conclusions remain the responsibility of the authors.

2 Literature review and motivation

Recent research has made some progress in understanding and detecting different mechanisms employed by APTs, particularly on Windows-based systems.

One of the earliest works addressing persistence at the operating-system level is [2], which demonstrated that monitoring anomalous Windows Registry accesses could effectively identify malicious software and persistent threats. This work established the feasibility of registry-level anomaly detection and highlighted the registry as a critical attack surface long before the formalization of modern APT models.

Building on this early insight, later research introduced structured taxonomies of persistence techniques. The study [43] systematically categorizes persistence mechanisms used by threat actors, providing a conceptual foundation for subsequent detection approaches and reinforcing the central role of registry-based techniques among others.

With the increasing availability of system telemetry, several works have explored dynamic malware detection by leveraging detailed Sysmon event logs that capture runtime behaviors such as process creation, registry edits, file modifications, network connections, and DNS queries. The study in [42] analyzes thousands of malware samples to quantify how frequently persistence mechanisms are adopted and evaluates ML models for their detection. While comprehensive, this work focuses on malware behavior in general rather than isolating specific persistence vectors.

Similarly, An Insight into [17] applies memory forensics and ML-based classification to detect fileless attacks. Although relevant to stealthy persistence, this approach targets memory-level artifacts rather than registry-based persistence mechanisms.

Another study [1] collects Sysmon telemetry to classify benign and malicious programs using supervised and unsupervised learning. However, persistence is treated as one behavioral signal among many (e.g., networking or DNS activity), and detection is driven by program behavior rather than explicit technique-level modeling aligned with MITRE ATT&CK.

Beyond individual detection approaches, the lack of realistic and reproducible datasets has motivated the creation of simulated APT scenarios. The dataset presented in [32] simulates 36 full-chain APT attack scenarios modeled after Chinese threat actors and mapped to MITRE ATT&CK tactics. While this dataset is publicly available and valuable for evaluating end-to-end APT detection systems, its broad scope makes it unsuitable for training specialized ML models focused on a single persistence technique, such as registry-based persistence. Nevertheless, it remains useful for external validation.

Addressing this limitation, a significant advancement is introduced in work [35], which presents a publicly available dataset specifically designed to characterize APT persistence techniques on Windows systems and aligned with MITRE ATT&CK. This dataset fills an important gap by shifting focus from early attack stages (e.g., initial access or lateral movement) toward long-term

persistence. It includes a wide range of persistence categories, such as registry autorun keys, startup folders, scheduled tasks, WMI event subscriptions, and shortcut modification. However, its deliberately broad scope means that it does not focus exclusively on registry-based persistence, despite the prevalence of such techniques in real-world campaigns.

Following this work, [36] presents a more specialized approach. The authors propose a ML system for detecting malicious scheduled tasks using Sysmon telemetry and Word2Vec-based feature representations. Their models achieve high detection performance with low false-positive rates, demonstrating the effectiveness of learning-based detection for a specific persistence vector. However, this study focuses solely on scheduled tasks and does not address registry-based persistence.

In parallel with dataset-driven and ML-based approaches, more recent research explores provenance-based and graph-based detection frameworks. The CPD system in [18] separates persistence into "setup" and "execution" phases and links events through dependency graphs to reduce false positives. Similarly, frameworks such as TREC and TBDetector [19] apply few-shot learning and transformer-based models to provenance graphs to recognize ATT&CK techniques and long-term APT behavior. These approaches demonstrate strong detection capabilities but require extensive system-wide logging and graph construction, leading to high computational overhead.

At the operational level, defenses such as SIEM platforms [8], SOC workflows, Sigma/YARA rules, and EDR solutions remain widely deployed. While effective for known patterns, these rule-based systems struggle to adapt to evolving persistence techniques and often suffer from false positives and limited generalization.

Despite these advances, several limitations remain.

First, AI-based detection of persistence mechanisms (particularly those implemented via Windows Registry keys) remains extremely limited. Although persistence is frequently acknowledged, registry-based techniques are often mentioned only briefly rather than studied in depth. This limitation is partly explained by the lack of datasets explicitly targeting registry persistence.

Second, existing datasets are narrow or overly broad. Available resources typically focus on general APT behavior, scheduled tasks, or fileless malware. To the best of our knowledge, no public dataset is dedicated specifically to registry-based persistence techniques, which significantly restricts the training and evaluation of specialized detection models.

Finally, provenance-based and graph-based approaches, while powerful, incur substantial computational costs due to fine-grained logging and graph construction. This limits their practicality in large-scale or resource-constrained environments.

CyLab, the cybersecurity research laboratory of the Royal Military Academy (RMA), conducts active research in the areas of network monitoring, intrusion detection, cryptographic protocol development, and advanced cyber threats, including APTs. The laboratory is recognized for its contributions to the analysis and detection of sophisticated attack techniques, particularly persistence mechanisms, and has established strong credibility through sustained research efforts supporting both academic inquiry and operational defense. Notably, CyLab is among the few research groups that explicitly investigate APT persistence strategies in depth, addressing a critical yet often underexplored aspect of modern cyber threats. [9]

As cited earlier, the work presented in [36] represents a significant advancement in this field by deepening the understanding of a sophisticated persistence technique employed by APTs. While this contribution constitutes an important step forward, it also underscores the broader challenge posed by the diversity and continuous evolution of strategies used by APTs to maintain long-term access to compromised systems. In the same line of research, the author introduced a dedicated dataset focusing on Windows Registry–based persistence techniques, as described in [35]. Building on this continuity, CyLab aims to systematically document persistence techniques across the APT landscape and to develop multiple specialized detection agents, each targeting a specific persistence mechanism. Within this broader research vision, the present work contributes as one such specialized agent, addressing Windows Registry–based persistence as part of a larger, modular defense strategy.

3 Registry keys as a persistence technique

Given these gaps, particularly the lack of datasets and detection methods targeting registry-based persistence, it becomes essential to examine how registry keys are used as a persistence mechanism and why they represent a high-value target for threat actors. The following section provides an overview of the registry structure and explains why protecting these components is critical for system security.

3.1 Definition

To understand the importance of securing the Windows Registry, it is necessary to outline what it is, how it functions, and why it is frequently exploited by APT groups.

The Windows Registry is a critical hierarchical database used by the Microsoft Windows operating system. It stores configuration information for the OS itself, installed applications, hardware drivers, and user-specific settings. Its structure is organized into five main hives, each containing multiple keys and subkeys as illustrated in [Figure 1]. Each subkey holds values, which consist of a value name, a data type, and associated data, often pointing to executables or processes that run automatically during system events such as startup or user logon.

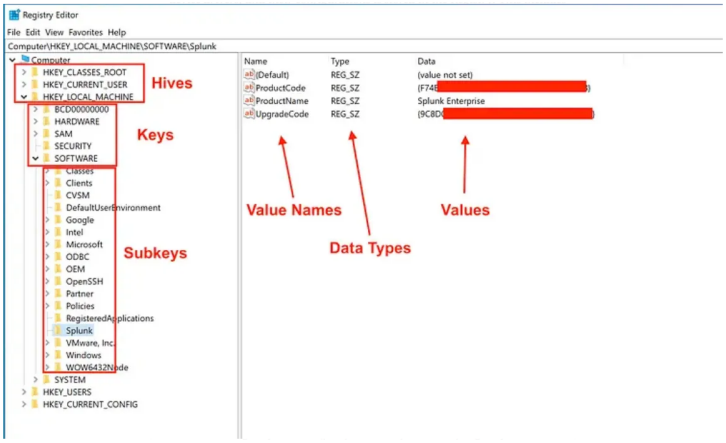


Figure 1: Example of Windows registry structure [20]

This database governs essential system behavior and underpins numerous configuration, persistence, and autostart mechanisms. To support system administration and software operation, Microsoft exposes several interfaces(ranging from graphical tools to low-level programmatic APIs) that allow registry interaction. While indispensable in legitimate environments, these interfaces are inherently dual-use and are frequently abused by threat actors when insufficiently secured or monitored. [41]

3.2 Techniques to access and modify Windows registry

There exist numerous methods and techniques for accessing and modifying the Windows Registry, which may be used for either legitimate or malicious purposes. To effectively detect registry manipulation, it is first necessary to understand these mechanisms as illustrated in [Figure 2].

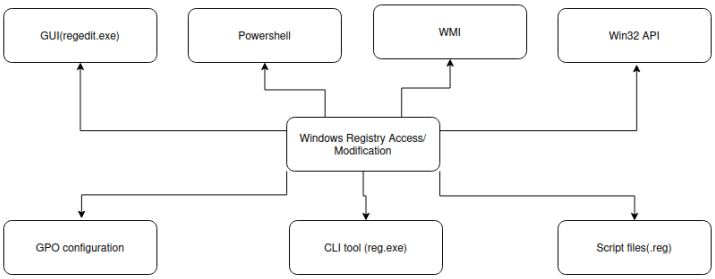


Figure 2: Some Windows registry access techniques [26]

Windows allows direct graphical interaction with the registry through the Registry Editor, commonly known as regedit.exe. This utility can be launched via Windows Search, the Run dialog, File Explorer, or the command line, and enables users to browse registry hives as well as create, modify, or delete keys and values. While regedit is widely employed for troubleshooting and manual configuration tasks, it offers little in the way of built-in safeguards. As a result, it is particularly susceptible to accidental misconfiguration by administrators and deliberate misuse by malicious actors [21].

Beyond manual interaction, Windows also supports automated and large-scale registry operations through scriptable and command-line interfaces. PowerShell, for instance, exposes the registry as a hierarchical drive structure such as HKLM and HKCU, and provides native cmdlets that allow reading, creating, updating, and deleting registry keys and values. This design makes PowerShell a cornerstone of modern configuration management, reporting, and system administration. However, its flexibility, deep system integration, and default availability have also made it a favored tool for fileless malware and stealthy persistence techniques. [13, 28]

Similarly, Windows includes lightweight command-line utilities that provide direct registry access. The built-in reg.exe tool supports querying, exporting, importing, and modifying registry data through simple commands. Because it is present by default on most Windows systems and can be easily scripted, reg.exe is extensively used in legitimate deployment and maintenance workflows, while also being leveraged in adversarial activity for stealthy configuration changes.

Closely related to command-line tools are registry script files with the .reg extension, which function as portable instruction sets containing predefined registry modifications. When imported, either through the graphical editor or via command-line utilities, these files immediately apply the specified changes. Their simplicity and effectiveness make them common in administrative automation, but the same characteristics also render them attractive for malicious payload delivery. [33]

In enterprise environments, registry interaction is often abstracted through centralized management frameworks. Windows Management Instrumentation (WMI), for example, provides powerful local and remote access to registry data through scripting languages such as PowerShell and VBScript, as well as through command-line tools. By leveraging specific WMI classes, administrators can read and modify registry settings across multiple systems from a central location. While this capability supports efficient enterprise management, it has also been repeatedly observed in malware campaigns, particularly for lateral movement and remote execution. [27, 11]

At an even higher level, Group Policy plays a fundamental role in corporate Windows infrastructures. Many system, security, and application settings enforced through Group Policy are ultimately implemented as registry configurations. Consequently, registry changes can be deployed centrally and automatically across entire domains, often without any direct user interaction, amplifying both their administrative power and their potential impact if misused.

Finally, at the lowest level, both legitimate software and malicious code may rely on the Win32 API for direct registry manipulation. Native functions such as RegOpenKeyEx allow applications to open and enumerate existing keys, while RegSetValueEx and RegDeleteKeyEx enable writing to or removing registry data[25]. This low-level and fine-grained access is essential for system software and installers but is equally attractive to custom malware operating with sufficient privileges. [23]

3.3 Most Abused Registry Keys

The MITRE ATT&CK framework [31] provides a comprehensive and structured knowledge base of adversarial techniques observed across real-world cy-

berattacks, covering the different phases of the Cyber Kill Chain. This framework is grounded in empirical evidence collected from documented campaigns conducted by APT groups, and it maps specific techniques to known threat actors, tools, and attack scenarios.

Within the persistence tactic, several techniques explicitly rely on the abuse of Windows Registry keys to maintain long-term access to compromised systems [12]. These techniques exploit legitimate system mechanisms originally designed to support configuration management, application startup, and user preferences. By modifying specific registry locations, adversaries can ensure that malicious code is executed automatically at system boot, user logon, or application launch, often without requiring further user interaction.

Based on the techniques listed in the MITRE ATT&CK framework, this work focuses on the registry-based persistence mechanisms most frequently observed in APT campaigns. For each identified technique, we introduce a summary in [Table 1] that presents its corresponding ATT&CK identifier, a concise explanation of the attacker's objective or typical exploitation scenario, the specific registry paths involved, and the level of privileges required to perform the modification. This structured representation aims to facilitate both defensive analysis and detection efforts by highlighting common abuse patterns and their operational constraints. [29]

4 Methodology of the proposed approach

In this section, we present the methodology of the proposed approach and highlight the main contributions of this work. One of the primary challenges in studying persistence mechanisms lies in the limited availability of publicly accessible datasets, particularly those specifically addressing persistence techniques implemented through the Windows Registry. Existing datasets often focus on general malware activity and rarely provide sufficient coverage of registry-based persistence behaviors, which remain a critical component of APT campaigns.

To address this gap, we constructed from scratch a comprehensive dataset specifically dedicated to Windows Registry–based persistence mechanisms. The dataset is built around an exhaustive set of persistence techniques mapped to the MITRE ATT&CK framework and reflects methods commonly employed by APT groups. It captures a broad spectrum of registry-related activities, encompassing both benign system operations and malicious persistence behaviors.

This contribution fills a notable gap in the current landscape of security datasets by enabling the training and evaluation of ML models specifically tailored to the detection of registry-based persistence techniques. By providing realistic and labeled registry event data, the dataset supports the development of effective and scalable detection systems capable of identifying stealthy persistence behaviors in Windows systems. In addition, it lays the groundwork for future research directions, including advanced AI-driven detection methods and automated threat-hunting approaches. By making this dataset publicly available, we aim to support reproducible research and foster continued progress in persistence detection and defensive security analytics.

Before training the ML models, it is necessary to construct an appropriate dataset and apply a series of preprocessing steps. The dataset is organized in a tabular form, where each row represents a system event and each col-

Registry Location / Key(s)	Purpose / Abuse Scenario	Privilege & Scope	MITRE ATT&CK Technique
HKCU/HKLM\Software\Microsoft\Windows\CurrentVersion\Run	Autostart entries executed at user logon. Common persistence mechanism used at both user and system levels.	User-level (HKCU) / Admin-level (HKLM)	T1547.001 (Registry Run Keys)
HKCU/HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce	One-time execution at next logon. Often used for staged or transient persistence.	User-level / Admin-level	T1547.001
HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnceEx	Extended RunOnce mechanism enabling controlled or conditional execution.	Admin-level	T1547.001
HKCU/HKLM\Software\Microsoft\Windows\CurrentVersion\RunServices	Legacy service-style autostart executed at/after logon; can background components.	User-level / Admin-level	T1547.001
HKCU/HKLM\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce	One-time legacy service-style autostart executed at/after login.	User-level / Admin-level	T1547.001
HKCU/HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run	Policy-based autostart locations that blend with legitimate administrative configurations.	User-level / Admin-level	T1547.001
HKLM\System\CurrentControlSet\Control\SessionManager\BootExecute	Executes programs very early during system boot, before user logon.	Admin-level only	T1547.001
HKCU/HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\ShellFolders	Defines shell folder paths (Desktop, Start Menu). Can be abused to redirect startup-related execution paths.	User-level / Admin-level	T1547.001
HKCU/HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\UserShellFolders	Expandable shell folder paths, commonly abused to redirect startup execution.	User-level / Admin-level	T1547.001
HKCU\Software\Microsoft\WindowsNT\CurrentVersion\Winlogon\Shell	Per-user override of Winlogon shell behavior; can hijack shell execution at logon.	User-level	T1547.004 (Winlogon Helper DLL)
HKCU/HKLM\Software\Microsoft\WindowsNT\CurrentVersion\Winlogon	Per-user or system-wide Winlogon settings controlling logon execution (e.g., Shell, Userinit).	User-level / Admin-level	T1547.004
HKLM\System\CurrentControlSet\Control\ServiceControlManagerExtension	Boot/service control manager extension settings; can be abused to influence service behavior early in boot.	Admin-level only	T1547.008 (Boot or Logon Autostart)
HKLM\System\CurrentControlSet\Control\SessionManager\Execute	Session Manager execution list; abused for early execution during boot/session initialization.	Admin-level only	T1547.008
HKLM\System\CurrentControlSet\Control\SessionManager\SubSystems\Command	Command execution during early session initialization (SSO); can enable stealthy pre-logon execution.	Admin-level only	T1547.008
HKLM\System\CurrentControlSet\Control\SessionManager\SetupExecute	Programs executed during setup/boot processing; may be abused for persistence or pre-logon execution.	Admin-level only	T1547.008
HKCU\Environment\UserInitMprLogonScript	Add logon scripts that execute automatically at logon initialization.	User-level	T1037.001

Table 1: Grouped Windows Registry persistence locations (Autorun/Auto-Start Execution Points – ASEPs) mapped to MITRE ATT&CK techniques and Sysmon detection events [29]

umn(referred to as a feature) captures a specific characteristic of that event. These features describe various aspects of system behavior, including user context, process execution, timestamps, tools involved, file paths, and file types. As a result, the dataset contains both relevant and irrelevant information, requiring careful preprocessing to ensure effective learning.

The extracted features are heterogeneous in nature. Some features are categorical, meaning they take values from a finite and relatively small set of possible categories. Examples include EventID, EventType, IntegrityLevel, LogonType, and User, which remain stable across events and do not exhibit high variability. Other features contain textual or high-cardinality data, characterized by a large number of distinct values. These include fields such as Image, ParentImage, ImagePath, TargetObject, CommandLine, and Hashes, which often encode detailed file paths, registry keys, or command-line arguments. Due to their high variability, these features require specialized encoding techniques to be used effectively by ML models. In addition, the dataset includes numerical features derived during feature extraction, such as counts, lengths, frequencies, and entropy-based measurements, which provide quantitative descriptions of system activity.

In [Figure 3], we present a global overview of the proposed processing and detection pipeline, illustrating the successive steps from data collection and preprocessing to feature engineering, model training, and evaluation.

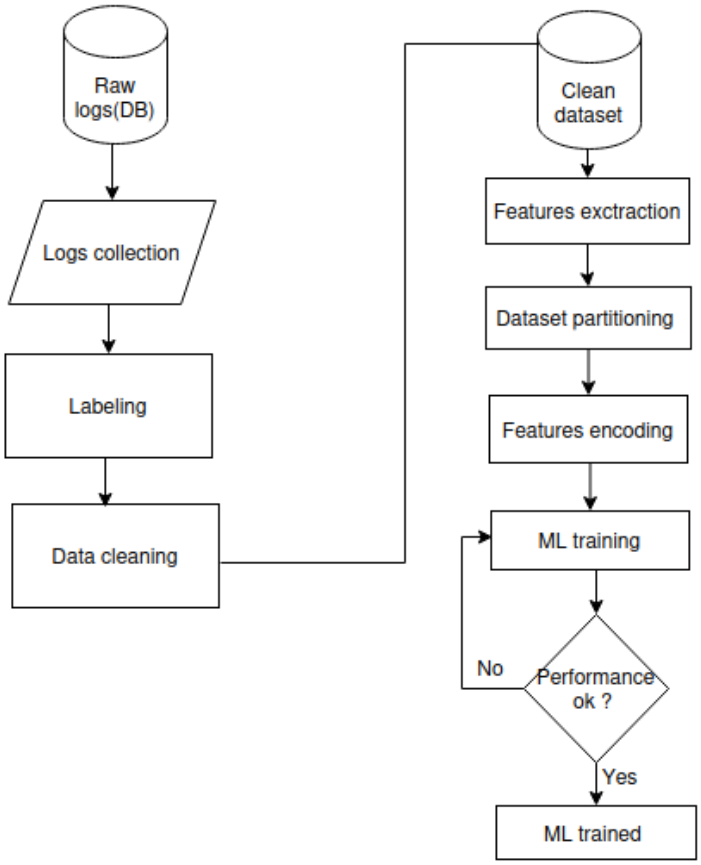


Figure 3: Overview of the project pipeline

4.1 Logs collection

4.1.1 Experimental environment and benign activity generation

The first step in constructing the dataset consisted of simulating a realistic operational environment. To this end, we deployed multiple virtual machines (VMs) running Windows 10 and Windows 11, configured to emulate typical enterprise endpoints. Within this environment, benign user activity was generated using the GHOSTS framework [40], which reproduces common day-to-day user behaviors such as document editing in Microsoft Word, Excel, and PowerPoint, web browsing, note-taking via Notepad, and email communication. The GHOSTS framework was configured on each VM according to the pipeline described in [6], which involves installing the GHOSTS framework itself alongside Microsoft 365 tools and the Firefox browser. Web browsing automation was enabled through the integration of the GeckoDriver component.

4.1.2 Adversary emulation and persistence simulation

To simulate malicious activity, we incorporated multiple adversary emulation frameworks that collectively cover a wide range of persistence techniques aligned with the MITRE ATT&CK framework. This combination enabled the controlled generation of realistic attack traces while maintaining reproducibility and precise ground truth.

First, we employed Atomic Red Team [37], a framework that provides fine-grained, technique-level tests mapped directly to individual MITRE ATT&CK techniques. Atomic Red Team allows the execution of isolated adversarial actions, such as specific registry modifications or startup mechanism abuses, without requiring a full attack chain. This approach is particularly well suited for validating individual persistence behaviors in a controlled manner.

In addition, we utilized the CALDERA adversary emulation platform [30], which enables the execution of multi-step attack campaigns through autonomous agents deployed on target systems. CALDERA operates using a command-and-control (C2) architecture, where lightweight agents installed on the victim machines receive instructions from a central server. Through predefined adversary profiles and abilities, CALDERA orchestrates realistic attack workflows, including privilege escalation, lateral movement, and persistence establishment. In this study, CALDERA was primarily configured to execute persistence-related techniques, allowing us to observe how registry-based persistence integrates within broader attack chains.

To further extend adversarial realism, we incorporated Empire [34], a post-exploitation framework designed to maintain and expand an attacker's foothold after initial compromise. Empire provides a rich set of modules for persistence, credential access, and system manipulation, and supports both PowerShell-based and Python-based agents. Its emphasis on stealthy, fileless, and registry-backed persistence mechanisms makes it particularly relevant for studying long-term compromise scenarios.

4.1.3 Host-based telemetry collection with Sysmon

To construct the dataset used in this study, Sysmon—part of the Microsoft Sysinternals [24] suite—was deployed on all Windows VMs. Sysmon is a host-based monitoring tool that provides fine-grained telemetry on system activity, including process creation, file system operations, network connections, and

registry modifications, depending on its configuration. For the purpose of this work, Sysmon was explicitly configured to focus on Windows Registry–related events. By tailoring the configuration to registry activity, the collected logs emphasize high-value events directly relevant to the detection objectives of this study.

4.1.4 Log validation and retention strategy

Following the execution of both benign system behavior and controlled persistence scenarios, the generated events were manually inspected to verify that the expected registry-related activity was correctly captured. All events were recorded under the Sysmon Operational log and accessed through the Windows Event Viewer [Figure 4], ensuring that raw, unfiltered telemetry was retained. To prevent event loss during prolonged or high-frequency activity, the maximum log size was increased to the highest allowable value and automatic log archiving was enabled. This configuration ensures continuous logging and preserves historical data across multiple execution phases, which is essential for accurately modeling persistence behaviors that may occur intermittently or across reboots.

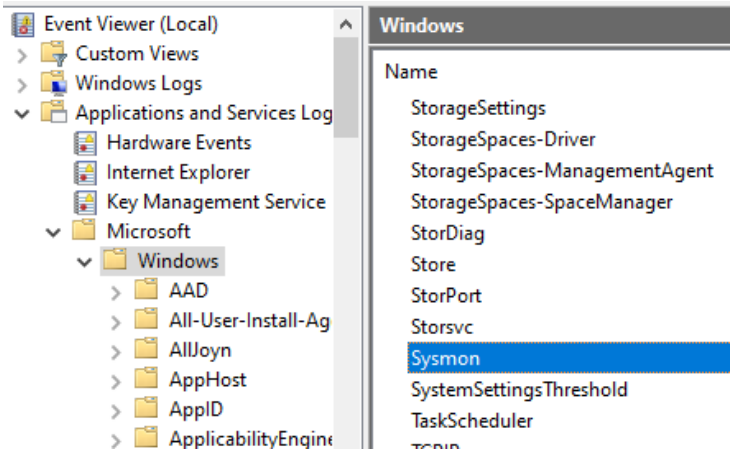


Figure 4: Sysmon path displayed in the Windows Event Viewer on a Windows 10 VM

4.1.5 Decentralized log extraction and data handling

Rather than relying on centralized log collection mechanisms such as Security Information and Event Management (SIEM) platforms or Windows-native forwarding and authentication components (e.g., Winlogon-based mechanisms), logs were collected directly from each VM. This design choice was motivated by the need to preserve complete event fidelity and avoid preprocessing, normalization, or filtering commonly applied by SIEM infrastructures, which may obscure low-level registry operations critical for persistence analysis. Additionally, centralized solutions typically require complex infrastructure, continuous network connectivity, and elevated administrative privileges, making them less suitable for controlled experimental environments.

In contrast, archived Sysmon logs were exported in their native `.evtx` format and extracted from each VM by temporarily hosting a lightweight HTTP

server on the guest systems. The archived logs were then downloaded to a host environment running Ubuntu for further processing and analysis. This approach minimizes system interference, remains OS-independent on the analysis side, and provides a reproducible and modular data collection pipeline.

4.1.6 Methodological implications and scalability

By adopting direct host-level log collection through Sysmon, accessible via the Windows Event Viewer as illustrated in [Figure 4], the proposed methodology remains independent of external logging infrastructures and is capable of operating as a standalone solution in isolated environments. At the same time, the approach is designed to be sufficiently modular to allow future integration as a complementary component within existing SIEM deployments, enabling fine-grained persistence detection without disrupting established security architectures. This flexibility allows the dataset construction methodology to scale from controlled research settings to real-world deployment scenarios, while maintaining full control over data quality and completeness.

Sysmon provides several event identifiers that are particularly relevant for detecting persistence techniques involving the Windows Registry as illustrated in [Table 2].

Event ID	**Event name**	**Event description**
1	Process Creation	Logs the creation of a new process, including command-line arguments and parent process information. Useful for detecting suspicious executables launched via persistence mechanisms
12	Registry Object Created or Deleted	Captures the creation or deletion of registry keys, including common persistence locations such as Run, Services, or Shell keys
13	Registry Value Set	Logs modifications to existing registry values, which may indicate the insertion of startup entries, disabling of security controls, or alteration of system configuration
14	Registry Object Renamed	May indicate attempts to conceal persistence by renaming registry keys or values
15	Registry Object Deleted	Can reflect cleanup actions by attackers or removal of obsolete persistence mechanisms
21	WMI Event Binding	Detects persistence via WMI event subscriptions, a stealthy and often fileless technique

Table 2: Mapping of Sysmon events to Windows Registry–based persistence techniques [22]

4.2 Dataset labeling

Before training ML models, the raw log data must undergo preprocessing to ensure consistency, reduce noise, and improve model performance. We can export the logs from Sysmon in the `.evtx` format, which is not directly usable

for ML workflows. Therefore, each log file was converted into a CSV format, which facilitates feature extraction and subsequent processing.

Each CSV file was then labeled according to a predefined ground truth, assigning a label of 0 for benign activity and 1 for malicious persistence activity.

Labeling plays a critical role in supervised ML, as it provides the model with explicit examples of normal and malicious behavior from which discriminative patterns can be learned. Accurate labels are essential for guiding the learning process during training and for enabling reliable performance evaluation during testing. Incorrect or inconsistent labeling can introduce bias, degrade model generalization, and lead to misleading evaluation metrics, particularly in security contexts where false positives and false negatives carry significant operational consequences.

After labeling, the CSV files from all machines were merged into a single dataset for unified preprocessing and model training.

4.3 Dataset cleaning

4.3.1 Empty columns removal

As part of the preprocessing pipeline, all features in the merged dataset were systematically examined to identify columns containing exclusively empty or null values. Such features do not convey any discriminative information and would otherwise introduce unnecessary noise during model training, potentially degrading learning efficiency and performance. Removing these columns ensures that subsequent learning algorithms operate solely on informative features that contribute to distinguishing benign activity from persistence-related behavior.

In total, 19 columns were removed during this step. These included: CallTrace, CommandLine, Company, CurrentDirectory, Description, FileVersion, GrantedAccess, ID, IntegrityLevel, OriginalFileName, ParentCommandLine, ParentImage, ParentProcessGuid, ParentProcessId, ParentUser, Product, SourceProcessGUID, SourceThreadId, and TargetProcessGUID. The elimination of these uniformly empty features resulted in a more compact and semantically meaningful feature set.

4.3.2 Path normalization

We implemented a systematic text normalization procedure for the paths features in the dataset, with the aim of preparing the data for ML. Specifically, it reads structured log data from a CSV file and applies a series of transformations to normalize variable elements in textual fields.

The normalization of path-related features is a key design choice for persistence detection, as file system and registry paths often contain environment-specific elements such as user names, temporary directories, or randomly generated identifiers. These components introduce high variability without conveying meaningful information about persistence behavior. Without normalization, models may overfit to absolute path values instead of learning generalizable structural patterns, such as directory hierarchy or the use of sensitive locations (e.g., autorun paths). By canonicalizing paths and removing volatile components, the normalization process preserves relevant semantic and structural information while reducing feature sparsity, enabling models to learn patterns that generalize across systems and deployment environments.

The normalization process addresses several types of system-specific and user-specific information, including:

- Identifiers and cryptographic hashes: Globally unique identifiers (GUIDs), MD5, SHA-1, and SHA-256 hashes are replaced with generic placeholders.
- System paths and user data: File system paths, user directories, and drive letters are canonicalized, removing personal or environment-specific details.
- Network and hardware addresses: IP addresses and MAC addresses are replaced with normalized tokens.
- Software versions and numeric sequences: Version numbers and other numeric sequences are generalized to reduce variability.
- Registry hives: Windows registry keys are mapped to a canonical form to ensure consistency across different representations.

All transformations aim to reduce irrelevant variability in the text, allowing for more robust statistical analysis, pattern recognition, or feature extraction. The normalized data is appended as new columns to the original dataset and exported to a separate CSV file, facilitating downstream processing without altering the original content.

4.3.3 Duplicated rows removal

As part of the preprocessing pipeline, a deduplication step was initially applied with the objective of removing duplicate rows, as multiple entries were observed to share identical feature values except for their timestamps. However, further analysis revealed that this approach was not necessarily appropriate in our context. Although such records appear redundant at first glance, the temporal dimension carries meaningful information about event frequency, recurrence, and persistence behavior. In particular, repeated events occurring at different timestamps may reflect sustained activity patterns or periodic execution mechanisms, which are essential for accurately modeling and detecting persistence techniques. Consequently, removing these entries risked discarding valuable temporal signals and biasing the dataset. For this reason, the deduplication step was reconsidered and ultimately excluded from the final preprocessing pipeline.

4.4 Feature engineering

Once a labeled and cleaned dataset has been obtained, the feature engineering phase can be conducted. Feature engineering aims to transform raw data into a set of informative, discriminative, and ML-ready features that enhance the model's ability to capture relevant patterns. This phase encompasses several key steps, including feature extraction, dataset partitioning, and the application of appropriate data encoding techniques.

4.4.1 Features extraction

To enrich the dataset with structural and statistical characteristics of system events, a set of additional features was computed. These features include Shannon entropy, which quantifies the level of randomness or disorder in string-based attributes such as file names, registry values, command lines, and pipe names, which is useful for identifying obfuscation or encoded payloads. Structural features such as path depth, string length, token counts, and argument counts are also extracted to characterize the complexity and structure of system artifacts. In addition, we introduce domain-informed binary indicators, including flags for executable file types, system directory usage, autorun-related registry keys, suspicious command-line arguments (e.g., PowerShell execution, encoded commands), base64-like content, and named pipe usage.

By combining these statistical, structural, and contextual features, the resulting dataset provides higher-level abstractions of system behavior that enhance the learning model's ability to detect persistence mechanisms.

4.4.2 Dataset partitioning

To ensure a rigorous evaluation of the proposed ML models and to assess their ability to generalize to unseen data, it is essential to clearly separate the data used for training from that used for testing. Using the same collection of logs for both phases would introduce data leakage, leading to overly optimistic performance estimates by allowing models to memorize host-specific patterns rather than learning generalizable characteristics of persistence behavior.

To mitigate this risk, the dataset was partitioned into two disjoint subsets based on the originating VM, rather than through random event-level sampling. Logs collected from three VMs(two running Windows 11 and one running Windows 10) were used exclusively for model training, while logs from two different VMs(running Windows 10 and Windows 11 respectively), were reserved for testing. This machine-level partitioning ensures that evaluation is performed on systems entirely unseen during training and prevents bias toward a specific operating system version, thereby enabling a more realistic assessment of cross-environment generalization. In total, the dataset contains 490,323 events, of which 358,078 were assigned to the training set and 132,245 to the test set.

The partitioning process was implemented programmatically by filtering the feature-enriched dataset according to a machine identifier associated with each event. The partitioning script described in [**gitlabrepo**] verifies the presence of both a machine identifier and a class label, assigns predefined machine groups to the training and test sets, and performs explicit sanity checks to ensure that no machine appears in both subsets. The script also reports the size and label distribution of each split to validate dataset characteristics prior to model training. The resulting label distribution reveals a notable class imbalance in both partitions, with benign activity (label 0) dominating the dataset. In the training set, approximately 89.1% of events are labeled as benign and 10.9% correspond to malicious persistence activity. In contrast, the test set contains a higher proportion of malicious events, with 27.0% labeled as malicious and 73.0% benign. Finally, the resulting datasets are exported as separate files for downstream training and evaluation, ensuring a controlled, reproducible experimental setup that closely reflects real-world deployment conditions.

4.4.3 Encoding pipeline

To ensure a consistent numerical representation of heterogeneous Sysmon data while preserving semantic relevance, multiple encoding strategies are applied in parallel within a unified preprocessing pipeline. Encoding methods are selected according to the structural and statistical properties of each feature category, enabling diverse event attributes to be integrated into a common feature space suitable for ML.

Several Sysmon fields, including Image, TargetFilename, TargetObject, and Details, exhibit extremely high cardinality, often containing tens or hundreds of thousands of distinct values. Encoding such attributes using one-hot encoding would be computationally impractical and memory-intensive. To address this limitation, a hashing-based encoding strategy is employed. This approach maps textual values into a fixed-dimensional numerical space that is independent of the number of unique entries, enabling scalable and efficient processing of large and evolving datasets. In this pipeline, the normalized textual attributes are first concatenated into a single text representation per event and subsequently transformed using a HashingVectorizer. This method avoids explicit vocabulary construction while preserving relevant distributional characteristics.

In parallel, low- to medium-cardinality categorical features are encoded using one-hot encoding, producing sparse binary representations while safely handling categories that may appear only in the test data. Numerical features, including length-based metrics, entropy-based measures, and other quantitative indicators derived during feature extraction, are passed through without modification in order to preserve their original semantic meaning. All transformations are coordinated through a column-wise preprocessing framework that combines the encoded textual, categorical, and numerical features into a unified sparse feature matrix. To prevent information leakage and ensure a realistic evaluation, the encoding pipeline is fitted exclusively on the training data and subsequently applied to the test data. The resulting encoded feature matrices, corresponding labels, and the trained preprocessing pipeline are serialized and stored to support reproducible and modular downstream model training and evaluation.

After encoding, the training set yields a feature matrix of size (358,078 $\times$ 262,181), while the test set yields a matrix of size (132,245 $\times$ 262,181). The identical dimensionality across both subsets confirms that preprocessing is applied consistently, which is essential for valid model training and evaluation. The final feature space is dominated by the hashing-based representation of the textual fields, with the remaining dimensions contributed by the one-hot encoded categorical variables and the passthrough numerical features.

4.4.4 Term Frequency–Inverse Document Frequency (TF–IDF)

As part of our preprocessing experiments, we investigated the use of TF–IDF [45] to transform textual fields extracted from Sysmon logs into numerical representations suitable for ML models. TF–IDF is a statistical technique that assigns weights to terms based on their frequency within individual records (term frequency) and their rarity across the entire corpus (inverse document frequency). This weighting scheme emphasizes distinctive terms while down-weighting ubiquitous ones, making it particularly attractive for identifying persistence-related artifacts, such as registry paths or autorun keys, that appear infrequently compared to benign system activity.

In this experiment, TF–IDF was applied to text-bearing fields relevant to persistence detection, namely Image, Details, TargetObject, and TargetFilename. These fields often contain complex file paths, registry keys, and command-line arguments, which can be treated as structured textual data. After loading the preprocessed dataset, missing values were replaced with empty strings, and the selected textual fields were concatenated into a single representation per event to capture the full execution and registry context. The TF–IDF vectorizer was configured to retain the most informative terms by limiting the vocabulary size, incorporating both unigrams and bigrams, ignoring extremely rare terms, and removing a custom list of high-frequency Windows-related stop words. The resulting representation yielded a sparse feature matrix with 5,000 lexical features, and an inspection of the highest-weighted terms confirmed that several persistence-related patterns were effectively captured.

To address the high dimensionality of the TF–IDF representation, we further experimented with dimensionality reduction using Truncated Singular Value Decomposition (TruncatedSVD) [39], a technique well suited for sparse matrices. By projecting the data onto a lower-dimensional latent space, this step aimed to preserve the dominant semantic structures while reducing computational complexity and overfitting risk. Although this approach demonstrated strong performance during initial evaluations on the constructed dataset, it was ultimately not adopted in the final methodology. The primary limitation observed was the lack of interpretability and generalization: TF–IDF-based models rely heavily on the presence of specific lexical tokens learned during training. When evaluated on external datasets containing different path structures or previously unseen terms, the models consistently failed to detect persistence events, often yielding zero detections. This behavior indicates that the approach captures word-level correlations rather than underlying behavioral patterns. As a result, despite promising in-dataset performance, the TF–IDF-based representation was deemed unsuitable for robust persistence detection across diverse environments and was therefore excluded from the final pipeline.

4.5 ML training and testing

After completing the data cleaning, preprocessing, and feature engineering steps, the next phase consists in training the selected models, generating predictions, and evaluating their performance using appropriate evaluation metrics. The implementation of the models and the corresponding training and evaluation pipelines are available in the project's GitLab repository [**gitlabrepo**].

The scripts implement the training, prediction and evaluation of various ML models using previously encoded Sysmon event data.

The input feature matrices for training and testing, along with their corresponding labels, are loaded from disk after being generated by an earlier preprocessing and encoding pipeline.

Following training, the model is evaluated on a held-out test set that was not used during fitting. Predictions are generated and assessed using standard classification metrics, including accuracy, precision, recall, and F1-score, which collectively capture both overall performance and class-specific detection capability. In addition, a confusion matrix is computed to derive false positive and false negative rates, metrics of particular importance in intrusion and persistence detection scenarios where both missed detections and false alarms carry operational costs. A detailed classification report is also produced to summa-

rize per-class performance. Finally, the trained Logistic Regression model is serialized and saved to disk, enabling reproducible experiments and facilitating later deployment or comparative analysis with other ML models.

4.5.1 Logistic regression

Logistic Regression is a widely used supervised learning algorithm for binary classification problems [10]. It models the probability that an input sample belongs to a given class by applying a Sigmoid (logistic) function to a linear combination of the input features. Despite its simplicity, Logistic Regression is particularly well suited for high-dimensional and sparse feature spaces, which are common in security telemetry and log-based datasets. In addition, its linear nature makes it highly interpretable, allowing insight into which features contribute most strongly to the classification decision. For these reasons, Logistic Regression is employed in this work as a baseline model for detecting persistence-related behavior in Sysmon event data.

Formally, Logistic Regression estimates the probability that a sample x belongs to the positive class (persistence) as

$$P(y = 1 \mid x) = \sigma(w^\top x + b) = \frac{1}{1 + e^{-(w^\top x + b)}}$$

Here, x is the feature vector representing an input sample, and w is the learned weight vector that determines the importance of each feature. The term $w^\top x$ computes a weighted linear combination of the features, while b is the bias term that shifts the decision boundary. The $\sigma(\cdot)$ maps this linear score to a value in $[0, 1]$, allowing the output to be interpreted as the probability that the sample belongs to the positive class ($y = 1$).

Model training consists of optimizing these parameters by minimizing the logistic loss, also referred to as the negative log-likelihood:

$$\min_{w,b} \sum_{i=1}^{n} \log\left(1 + e^{-y_i(w^\top x_i + b)}\right)$$

To account for class imbalance commonly observed in security datasets, the model is configured with balanced class weights, ensuring that minority-class samples contribute proportionally to the optimization process. The model is trained using an iterative solver with an increased iteration limit to guarantee convergence on the large feature set.

4.5.2 Random Forest

Random Forest is an ensemble learning algorithm that combines the predictions of multiple decision trees in order to achieve robust and well-generalized classification performance. By aggregating the outputs of diverse learners, typically through majority voting, the model effectively mitigates the variance and instability commonly associated with individual decision trees. This property makes Random Forest particularly well suited for high-dimensional and noisy data sources, such as system and security logs [14].

At a conceptual level, the algorithm operates by constructing a collection of decision trees, each trained on a different bootstrapped subset of the training data. In addition, at every split within a tree, only a random subset of features is considered when determining the optimal splitting criterion. This dual

randomness(applied both to data sampling and feature selection)decorrelates the trees and prevents them from converging toward identical structures. As a result, the ensemble captures a broader range of decision boundaries and improves generalization to unseen data.

From a technical perspective, each decision tree recursively partitions the feature space by selecting thresholds that maximize a purity measure, such as Gini impurity or information gain. Once the ensemble is trained, predictions for new samples are obtained by aggregating the individual tree outputs via majority voting. This collective decision-making process reduces overfitting and increases resilience to noise and outliers.

Random Forest models are capable of handling imbalanced and complex feature distributions while maintaining strong discriminative performance.

In our implementation, the Random Forest classifier is configured with a fixed random seed to ensure reproducibility and trained using parallel computation across multiple CPU cores to improve efficiency. The model consists of 300 decision trees, each trained exclusively on the training dataset to avoid information leakage. After training, the ensemble is used to predict labels for previously unseen samples.

4.5.3 Extreme Gradient Boosting (XGBoost)

XGBoost is an ensemble learning algorithm based on gradient-boosted decision trees, designed for efficiency, scalability, and high predictive performance on structured data. It constructs a strong classifier by iteratively combining multiple weak learners, where each learner is a decision tree that captures nonlinear relationships and feature interactions [46]. XGBoost is particularly effective in handling heterogeneous feature sets, sparse input representations, and complex decision boundaries, which are common characteristics of system-level telemetry and security event data.

The model is configured for binary classification using a logistic objective function, which enables it to estimate the probability that an event corresponds to persistence-related behavior. During training, XGBoost builds decision trees sequentially, with each new tree trained to minimize the residual errors of the previous ensemble by following the gradient of the loss function. Key hyperparameters, including the learning rate, maximum tree depth, number of estimators, and subsampling ratios for both rows and features, control the trade-off between model complexity and generalization. These mechanisms reduce overfitting while allowing the model to learn expressive, non-linear patterns from the data. Training is parallelized across multiple CPU cores, significantly improving computational efficiency on large datasets.

XGBoost is well suited to the objectives of this work because persistence detection often relies on subtle interactions between multiple event attributes rather than on single features in isolation. By combining many shallow trees, the model can capture complex behavioral patterns across registry activity, process execution, and command-line characteristics, while remaining robust to noise and class imbalance.

Since the task is formulated as a binary classification problem, XGBoost optimizes the binary cross-entropy loss (logistic loss), defined as:

$$L = -\sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Where y_i denotes the ground-truth label of sample i, and p_i represents the probability predicted by the model that the sample belongs to the positive class.

During training, XGBoost builds decision trees sequentially, with each new tree learning the negative gradient of the loss function, which corresponds to the direction of the current prediction error. As a result, the first trees capture coarse decision patterns, while subsequent trees progressively focus on correcting the residual errors of the ensemble. After a sufficient number of boosting iterations, the combined model achieves high predictive accuracy by aggregating these incremental improvements.

The model is configured using a set of hyperparameters that control the trade-off between learning capacity and generalization. The learning rate=0.1 scales the contribution of each individual tree, reducing the risk of overfitting. The maximum tree depth max_depth=6 limits the complexity of each tree, while the number of estimators n_estimators=300 determines the total number of boosting rounds. Subsampling parameters subsample=0.8 and colsample_bytree=0.8 introduce randomness by using only a fraction of the training samples and features per tree, which helps reduce variance and decorrelate trees. Finally, the objective="binary:logistic" setting enables probabilistic binary classification, and eval_metric="logloss" is used as the optimization metric during training

4.5.4 Light Gradient Boosting Machine (LightGBM)

LightGBM is a gradient boosting framework based on decision tree learners, designed to achieve high predictive performance while maintaining computational efficiency and low memory consumption. Unlike other boosting methods that grow trees level-wise, LightGBM adopts a leaf-wise growth strategy, allowing it to focus on splits that yield the largest reduction in loss [16]. This makes the model particularly effective for large-scale, high-dimensional, and sparse datasets.

In this implementation, LightGBM is configured for binary classification, with the objective of distinguishing benign activity from persistence-related behavior. To address the pronounced class imbalance, the ratio between benign and malicious samples is explicitly computed and used to scale the contribution of positive-class instances during training. The model is trained using a relatively small learning rate combined with a large number of boosting iterations, enabling gradual refinement of the decision function and reducing the risk of overfitting. Model complexity is further controlled through parameters such as the number of leaves, while parallel execution across multiple CPU cores ensures efficient training.

4.5.5 Multilayer Perceptron(MLP)

The MLP is a feed-forward neural network composed of fully connected layers with nonlinear activation functions. It learns hierarchical representations of the input space through backpropagation. While simpler than Transformer-based architectures, MLPs can capture nonlinear relationships across features extracted from Sysmon logs [44]. Their flexibility and moderate complexity make them suitable for supervised classification of persistence-related events.

The code follows the standard ML workflow used in for the previous models: data loading, preprocessing, train/test splitting, model training, evaluation,

and model serialization.

Here, 25% of the samples are allocated to testing

The MLP requires normalized inputs to ensure stable gradient descent. The script standardizes features using:

$$x' = \frac{x - \mu}{\sigma}$$

where: μ = feature mean (training set), σ = standard deviation.

Scaling is fitted only on the training set and applied to both sets to avoid data leakage

As an input layer, we use the engineered features. And we use 2 hidden layers, where the layer 1 has 256 neurons and the layer 2 has 128 neurons.

The activation function ReLU enables efficient training and helps avoid vanishing gradients.

For the training of the algorithm, MLP uses the Adam optimizer, which performs adaptive gradient estimation by combining Momentum and RMSProp, adjusting learning rates dynamically and converging faster on large datasets

The model monitors validation performance and stops training when no improvement is observed for 10 consecutive epochs, which prevents overfitting and reduces unnecessary computation.

To handle class imbalance, in the script we compute sample weights by assigning higher weights to minority class samples (so malicious events).

Effectively, the loss function becomes:

$$L = \sum_i w_i \cdot \ell(y_i, \hat{y}_i)$$

which penalizes misclassification of rare samples more heavily

The MLP is trained using weighted samples. Followed by a second training loop that refines the model further, ensuring additional epochs are spent improving convergence.

During training, each neuron performs:

$$z = Wx + b$$

$$a = \text{ReLU}(z) = \max(0, z)$$

The network adjusts weights using back propagation:

$$W \leftarrow W - \eta \cdot \frac{\partial \mathcal{L}}{\partial W}$$

This minimizes the classification loss, improving the decision boundary separating benign from persistence-related events.

Both the trained classifier and the scaler are saved using joblib.

5 Results interpretation

5.1 Evaluation metrics

To assess the performance of the ML models during both the training and testing phases, we rely on several standard classification metrics. These metrics [38] provide a comprehensive understanding of how well each model distinguishes between benign and malicious instances.

- Precision quantifies the proportion of samples classified as positive that are truly positive. It reflects the model's ability to avoid false alarms (false positives), which is particularly important in cybersecurity settings where excessive alerts can overwhelm analysts.

  Precision = $\frac{TP}{TP+FP}$

- Recall, also known as sensitivity or true positive rate, measures the proportion of actual positive samples that the model successfully identifies. High recall indicates the model's ability to detect attacks and minimize false negatives.

  Recall = $\frac{TP}{TP+FN}$

- Because precision and recall often exhibit a trade-off, we employ the F1-score, the harmonic mean of precision and recall, to provide a balanced evaluation. It is especially useful in imbalanced datasets, such as those common in intrusion detection.

  F1-score = $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

- Support represents the number of true samples in each class (dataset distribution).

- In addition to these scalar metrics, we use the confusion matrix, which provides a detailed breakdown of:

  True Positives (TP) False Positives (FP) True Negatives (TN) False Negatives (FN)

  From the confusion matrix, we derive the false positive rate (FPR) and false negative rate (FNR), which are critical indicators in security-oriented models:

  FPR = $\frac{FP}{FP+TN}$ FNR = $\frac{FN}{FN+TP}$

  In cybersecurity detection systems, low FNR is essential to avoid missed attacks, while low FPR reduces alert fatigue.

- While the accurancy represents the overall correctness of predictions.

  Accuracy = $\frac{TP+TN}{\text{Total}}$

These combined metrics allow us to evaluate and compare different model families, such as tree-based methods, neural networks, and ensemble approaches, by highlighting their strengths and weaknesses. Tree-based models, for example, may offer better interpretability and robustness to noise, while neural networks may achieve higher recall but risk overfitting if poorly regularized. The metrics presented in [Table 3] enable a detailed and systematic interpretation of these behaviors.

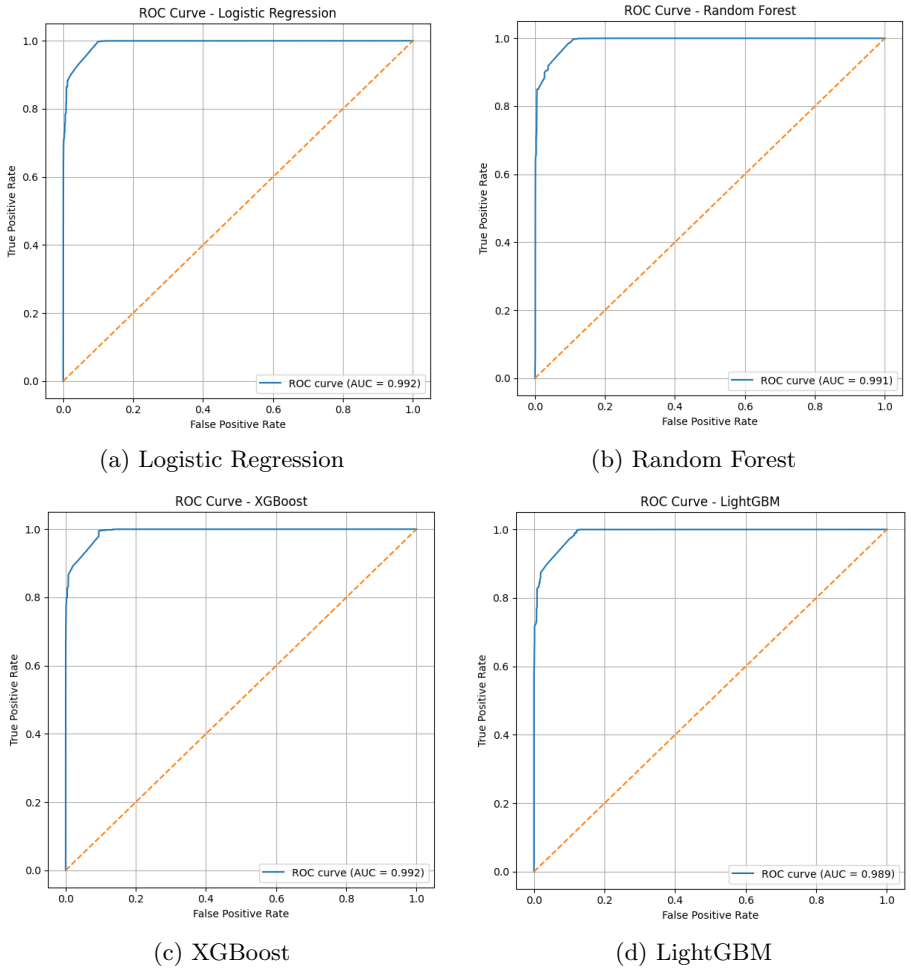
ML model	**Precision**	**Recall**	**F1 score**	**FPR**	**FNR**	**Accuracy**
Logistic Regression	0.57	1.00	0.73	0.2796	0.0000	0.7960
Random Forest	0.98	0.78	0.87	0.0051	0.2188	0.9371
XGBoost	0.78	1.00	0.88	0.1025	0.0035	0.9243
LightGBM	0.76	0.98	0.86	0.1134	0.0156	0.9131
MLP	0.78	0.92	0.85	0.0958	0.0789	0.9087

Table 3: Evaluation metrics of ML models for registry-based persistence detection

The evaluation results demonstrate clear trade-offs between detection accuracy and error behavior across the tested ML models. Logistic Regression achieves perfect recall, indicating that all malicious instances are detected; however, this comes at the cost of a very high false positive rate, resulting in reduced precision and overall accuracy. While such behavior may be acceptable in highly conservative security settings, it is likely to generate excessive false alerts in operational environments.

In contrast, ensemble-based models exhibit a more balanced performance. Random Forest achieves the highest accuracy and the lowest false positive rate, making it well suited for environments where minimizing false alarms is critical, although its higher false negative rate indicates some missed attacks. XGBoost and LightGBM provide strong overall performance, combining high recall with moderate false positive rates, which suggests an effective compromise between detection coverage and alert precision. The MLP model also demonstrates high recall and competitive accuracy, but with a higher false positive rate than tree-based ensembles. Overall, these results indicate that ensemble models offer the most favorable balance between detection effectiveness and error control for registry-based persistence detection.

5.2 ROC curves comparison



(a) Logistic Regression (b) Random Forest

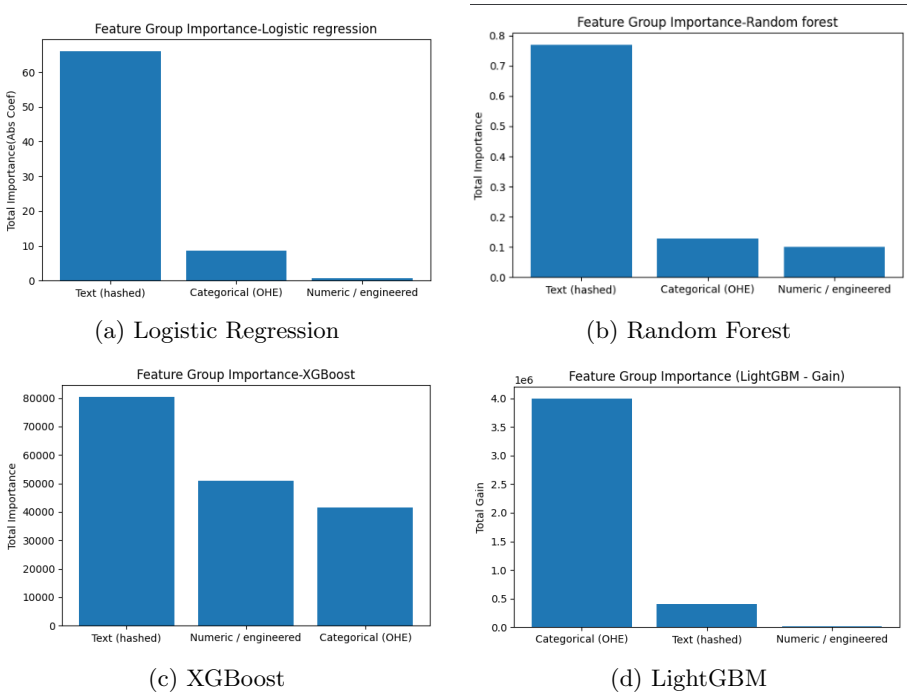
(c) XGBoost (d) LightGBM

Figure 5: ROC curves comparing the performance of the evaluated ML models

Overall, as illustrated in [Figure 5], all evaluated models demonstrate strong classification performance, with AUC values exceeding 0.98, indicating that the engineered feature set provides highly discriminative information for detecting registry-based persistence mechanisms. Among the models, XGBoost and Random Forest achieve the highest AUC scores, suggesting superior capability in capturing complex, non-linear patterns inherent to persistence activity. LightGBM follows closely, offering comparable performance with potential advantages in training efficiency. Logistic Regression, while slightly less performant, still achieves robust results and serves as a valuable baseline due to its simplicity and lower computational overhead.

From a practical perspective, the ensemble-based models are better suited for high-accuracy detection scenarios, whereas Logistic Regression may be preferred in environments where model interpretability and simplicity are prioritized.

5.3 Feature importance comparison



(a) Logistic Regression (b) Random Forest

(c) XGBoost (d) LightGBM

Figure 6: Feature importance comparison across the evaluated ML models for registry-based persistence detection

Feature importance is a commonly used concept in ML to understand how input variables influence a model's predictions. It quantifies the relative contribution of features to the decision-making process and is particularly valuable for interpreting trained models, validating that predictions rely on meaningful inputs, and guiding feature engineering. In tree-based models such as Random Forest, LightGBM, and XGBoost, feature importance is typically derived from criteria such as information gain, impurity reduction, or the frequency with which features are used in decision splits.

In this work, direct feature-level importance analysis was not feasible due to the encoding strategies applied during preprocessing. Textual features were transformed using hashing-based vectorization, which maps inputs to a fixed-dimensional space without preserving a deterministic relationship between encoded dimensions and original attributes. In addition, categorical features were expanded through one-hot encoding into high-dimensional binary representations. As a result, individual encoded features lack clear semantic meaning, making fine-grained feature importance interpretation unreliable.

To address this limitation, we instead performed a feature group importance analysis by aggregating importance scores across three predefined feature categories as presented in [Figure 6]: hashed textual features, one-hot–encoded categorical features, and numerical features passed through unchanged, including engineered attributes such as depth, length, and entropy. Although this approach provides a coarser level of analysis, it remains informative, as it reveals how different models prioritize broad types of information rather than

individual encoded dimensions.

The results show that models differ in how they exploit these feature groups for registry-based persistence detection.LightGBM relies predominantly on categorical features, highlighting the importance of structured event attributes, while textual features play a secondary role. In contrast, XGBoost and Random Forest place greater emphasis on hashed textual features, with additional contributions from numerical and categorical features, reflecting their ability to integrate semantic and behavioral information. Logistic Regression is largely driven by textual features, consistent with its linear nature and sensitivity to high-dimensional representations.

6 Validation and Discussion

Evaluating the model on heterogeneous datasets, containing diverse environments, malware families, and persistence techniques, allows us to assess its degree of generalization. This step is essential to determining whether the model can robustly identify Registry-based persistence mechanisms even when confronted with previously unseen attack scenarios or mixed APT behaviors. A strong performance on external data would therefore provide evidence that the proposed approach is not only effective in controlled experimental setups but also applicable to real-world, multi-variant threat landscapes.

6.1 Evaluation metrics

ML model	Precision	Recall	F1 score	FPR	FNR
Random Forest	0.9632	0.3183	0.4784	0.0002	0.6817
XGBoost	0.0752	0.3026	0.1205	0.0484	0.6974

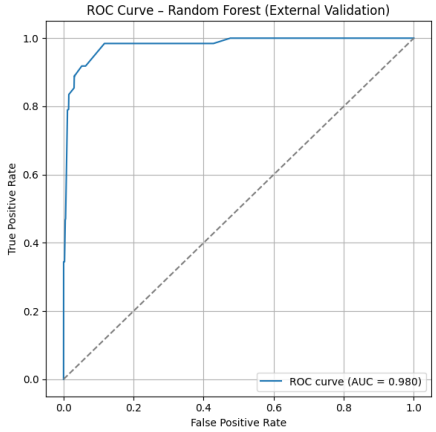
Table 4: Evaluation metrics of ML models for registry-based persistence detection for validation

The evaluation metrics presented in [Table 4] highlight markedly different behaviors between the Random Forest and XGBoost models in the context of registry-based persistence detection. The Random Forest model exhibits a very high precision (0.9632), indicating that when the model predicts persistence, it is almost always correct. However, this strong precision comes at the cost of a low recall (0.3183), meaning that a large proportion of actual persistence instances are not detected. This imbalance is further reflected in the high false negative rate (FNR = 0.6817), suggesting that the model adopts a highly conservative decision strategy, prioritizing the minimization of false positives over comprehensive detection. Consequently, the moderate F1 score (0.4784) reflects this trade-off between precision and recall.

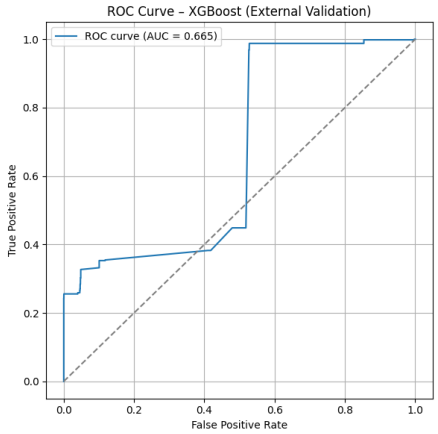
In contrast, the XGBoost model demonstrates substantially lower precision (0.0752), indicating a high proportion of false positive predictions. While its recall (0.3026) is comparable to that of the Random Forest model, the elevated false positive rate (FPR = 0.0484) suggests that XGBoost struggles to effectively discriminate between benign and persistent registry behaviors. The combination of low precision and moderate recall results in a low F1 score (0.1205), highlighting weaker overall classification performance. These results

indicate that, in its current configuration, XGBoost tends to over-predict persistence, reducing its reliability in scenarios where false alerts are costly.

6.2 ROC curves comparison



(a) Random Forest



(b) XGBoost

Figure 7: ROC curves comparing the performance of the evaluated ML models for validation

The ROC curves presented in [Figure 7] provide further insight into the discriminative capabilities of the two models across varying classification thresholds. The Random Forest ROC curve demonstrates a stronger deviation from the diagonal baseline, indicating a better balance between true positive and false positive rates over a range of thresholds. This behavior suggests that Random Forest captures relevant patterns in the feature space that enable it to distinguish persistent from non-persistent registry activity more effectively, even though its default operating point favors precision.

Conversely, the ROC curve of the XGBoost model lies closer to the diagonal, reflecting limited discriminative power. This indicates that changes in the decision threshold do not significantly improve the trade-off between sensitivity

and specificity. As a result, even though XGBoost achieves recall values similar to Random Forest at the chosen threshold, its overall ability to separate classes remains weaker, which is consistent with its poor precision and higher false positive rate observed in the evaluation metrics.

6.3 Discussion

The observed discrepancy between model performance on the training dataset and the external validation dataset motivates several hypotheses that may explain why two highly performant ML models exhibit reduced effectiveness when evaluated on external data. One plausible explanation relates to differences in the pipeline of data collection, attack execution, and system behavior across datasets.

The external dataset obtained from the Mendeley Data repository [32] is derived from logs collected by Wazuh and includes events generated by multiple processes and attack scenarios associated with different APT techniques. According to the dataset documentation, the attacks were conducted using the CALDERA adversary emulation framework. CALDERA operates by deploying agents on victim machines, which communicate with a command-and-control server through periodic beaconing [30]. However, the dataset does not provide explicit guarantees regarding the lifecycle of these agents after delivery. The agents may have been terminated or disconnected due to network interruptions, defensive countermeasures, or configuration constraints, thereby preventing sustained beaconing activity.

In particular, there is no assurance that system reboots or user logoff events occurred following agent deployment, which are often required for certain persistence mechanisms to be activated. Moreover, it is unclear whether persistence was successfully established, specifically through Windows Registry–based techniques, or whether such mechanisms were detected and neutralized by endpoint protection or monitoring tools present on the system.

These uncertainties introduce variability in the presence and visibility of persistence-related artifacts within the external dataset. This highlights the challenges associated with finding publically available datasets that are suitable for training or even validating specific ml models.

Another explanation is that while a ground truth was constructed for the public dataset before the validation (since it wasnt provided with the dataset) , its reliability may inherently differ from that of the internally curated dataset due to fundamental differences in data generation and labeling processes. In the controlled experimental setting used for internal data collection, attack execution, timing, and persistence mechanisms were fully supervised, allowing labels to be assigned with high confidence based on direct observation of malicious actions. In contrast, labeling external datasets typically relies on secondary sources such as attack descriptions, tool documentation, alert correlations, or inferred behaviors derived from log analysis. This indirect approach introduces uncertainty, as not all malicious actions may be observable, fully executed, or consistently logged, particularly when attacks are partially mitigated or interrupted by defensive mechanisms.

As a result, the external dataset may contain samples that are mislabeled, incompletely labeled, or inherently ambiguous with respect to persistence behavior. For example, registry modifications associated with persistence may be absent, short-lived, or indistinguishable from benign system activity, despite

an attack being nominally present.

6.4 Impact of threshold on model performance

Threshold	T=0.1	T=0.2	T=0.3	T=0.4	T=0.5	T=0.6	T=0.7	T=0.8	T=0.9
F1-score RF	0.052	0.546	0.504	0.478	0.478	0.478	0.176	0.050	0.024
Recall RF	1.00	0.791	0.343	0.32	0.318	0.318	0.097	0.026	0.012
F1 XGBoost	0.026	0.029	0.022	0.068	0.121	0.116	0.114	0.118	0.389
Recall XGBoost	0.998	0.998	0.449	0.353	0.303	0.283	0.276	0.256	0.243

Table 5: Performance metrics as a function of classification threshold T

As shown in [Table 5], we investigated the impact of adjusting the classification decision threshold, which is set to 0.5 by default in most ML classifiers. Lowering this threshold increases the sensitivity of the model, leading to higher recall values, whereas increasing the threshold results in a more conservative decision rule and consequently reduces recall.

This behavior can be explained by the trade-off inherent to probabilistic classifiers. When the threshold is decreased, a larger number of samples are classified as positive, allowing the model to capture a greater proportion of true persistence events, at the cost of potentially increasing false positives. Conversely, higher threshold values require stronger confidence before assigning the positive class, which reduces false positives but increases the likelihood of false negatives. These observations are consistent with the expected precision–recall trade-off and highlight the importance of threshold tuning when recall is a critical objective, particularly in persistence detection scenarios where missed detections can have significant security implications.

7 Conclusion

The increasing frequency and sophistication of APT attacks underscore the urgent need to investigate system vulnerabilities, understand adversarial techniques, and develop effective countermeasures to prevent, detect, and mitigate malware, intrusions, and, in particular, stealthy persistence mechanisms. Existing endpoint and security monitoring solutions often suffer from alert fatigue or fail to detect advanced malware due to their reliance on known attack patterns and rule-based signatures. At the same time, recent advances in AI have lowered the barrier to developing and orchestrating large-scale automated attacks through the use of scripts and autonomous agents, further exacerbating the threat landscape. In this context, the approach presented in this work aims to address persistence detection through ML–based techniques.

A major challenge encountered during this research was the limited availability of publicly accessible datasets suitable for training and evaluating persistence detection models. To address this gap, a large-scale dataset was constructed, focusing on a variety of persistence techniques targeting the Windows Registry. This dataset was derived from Sysmon telemetry collected in controlled Windows 10 and Windows 11 environments and includes both benign user activity and malicious persistence behavior. After applying a carefully designed preprocessing pipeline, feature engineering, and strict train–test separation at the virtual machine level, multiple machine learning models were trained using heterogeneous feature representations combining hashed textual

fields, one-hot–encoded categorical attributes, and engineered numerical indicators. During internal evaluation, ensemble-based methods demonstrated the strongest performance, with Random Forest and XGBoost achieving F1-scores of 0.87 and 0.88, respectively, and emerging as the most effective classifiers for distinguishing benign activity from registry-based persistence behavior.

External validation on an independent dataset further emphasized the importance of model generalization under distributional shifts and labeling uncertainty. For the Random Forest model, recall reached its maximum value of 1.00 at a low decision threshold (T=0.1) but decreased sharply as the threshold increased, indicating high sensitivity to threshold selection and a significant loss in detection capability under more conservative decision rules. In contrast, XGBoost exhibited a more gradual decline in recall as the threshold increased, maintaining higher sensitivity across a wider range of thresholds. However, this stability came at the cost of a weaker balance between detection accuracy and prediction reliability. Overall, these findings highlight the critical role of threshold tuning and cross-dataset validation in the deployment of machine learning–based persistence detection systems and demonstrate both the potential and limitations of learning-based approaches in addressing stealthy, long-term intrusions in modern Windows environments.

Due to time constraints, it was not possible to conduct additional experiments or perform further validation on a wider range of external datasets, nor to construct a new dataset specifically tailored to the models' requirements. These limitations naturally define several directions for future work. In particular, extending the evaluation to additional datasets and designing custom datasets targeting specific persistence behaviors would allow for a more comprehensive assessment of model generalization. Furthermore, systematic hyperparameter tuning could be explored to further optimize model performance and better understand the trade-offs between detection sensitivity, precision, and robustness across different operating conditions.

References

[1] Riki Mi'roj Achmad et al. *Sysmon Event Logs for Machine Learning-Based Malware Detection*. 2025. URL: https://www.sciencedirect.com/science/article/pii/S277291842500027X.

[2] Frank Apap et al. *Detecting malicious software by monitoring anomalous Windows registry accesses*. 2002. URL: https://www.cs.columbia.edu/~sh553/papers/drafts/rad-dist02.pdf.

[3] AV-ATLAS. *Total amount of malware and PUA under Windows*. 2025. URL: https://portal.av-atlas.org/malware/statistics.

[4] Osato Avan-Nomayo. *North Korean hackers $308m DMM Bitcoin heist ranked 2024 biggest. AI will make attacks even worse*. 2024. URL: https://www.dlnews.com/articles/web3/ai-make-crypto-hacks-worse-as-investors-lost-23bn-in-2024/.

[5] Lenaerts-Bergmans Bart. *What is the Cyber Kill Chain? Process Model*. 2022. URL: https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/cyber-kill-chain/.

[6] Ghita Bennouna. *GHOSTS v8.0 Implementation: Orchestrating Realistic Traffic for PoC Attack Simulation and Log Monitoring*. 2025. URL: https://cylab.be/blog/395/ghosts-v80-implementation-orchestrating-realistic-traffic-for-poc-attack-simulation-and-log-monitoring.

[7] Ghita Bennouna. *ML-detection-of-peristence-via-persistence*. 2025. URL: https://gitlab.cylab.be/G.Bennouna/ML-detection-of-persistence-via-registry/.

[8] CardinalOps. *Enterprise SIEMs Miss 79% of MITRE ATT&CK Techniques Used by Adversaries, According to CardinalOps' 5th Annual Report*. 2025. URL: https://www.prnewswire.com/news-releases/enterprise-siems-miss-79-of-mitre-attck-techniques-used-by-adversaries-according-to-cardinalops-5th-annual-report-302473779.html.

[9] CyLab, Royal Military Academy. *About CyLab*. 2025. URL: https://cylab.be/about.

[10] EntropyObserver. *Logistic Regression*. 2025. URL: https://techenglish.top/article/1c7d698f-3512-80f8-8fa0-c949bde042fc?utm_source=chatgpt.com.

[11] ESET. *WMI in the Hands of Malware*. 2015. URL: https://www.welivesecurity.com/2015/08/20/wmi-malware/.

[12] Maurice Fielenbach. *Detecting the Most Popular MITRE Persistence Method – Registry Run Keys / Startup Folder*. 2025. URL: https://www.nextron-systems.com/2025/07/29/detecting-the-most-popular-mitre-persistence-method-registry-run-keys-startup-folder/.

[13] FireEye. *Fileless Malware*. 2017. URL: https://www.fireeye.com/current-threats/what-is-fileless-malware.html.

[14] GeeksforGeeks. *Random Forest Algorithm in Machine Learning*. 2025. URL: https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/.

[15] HandWiki. *Stuxnet*. 2022. URL: https://encyclopedia.pub/entry/37304.

[16] Mohtasim Hossain. *Mastering LightGBM: An In-Depth Guide to Efficient Gradient Boosting*. 2022. URL: https://medium.com/@mohtasim.hossain2000/mastering-lightgbm-an-in-depth-guide-to-efficient-gradient-boosting-8bfeff15ee17.

[17] Osama Khalid et al. *An Insight into the Machine-Learning-Based Fileless Malware Detection*. 2023. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC9861630/.

[18] Qi Liu et al. *Accurate and Scalable Detection and Investigation of Cyber Persistence Threats*. 2024. URL: https://arxiv.org/abs/2407.18832.

[19] Mingqi Lv et al. *TREC: APT Tactic / Technique Recognition via Few-Shot Provenance Sub-graph Learning*. 2024. URL: https://arxiv.org/abs/2402.15147.

[20] Ömer MEMES. *What is Windows Registry and Why?* 2024. URL: https://medium.com/@omermemes83/what-is-windows-registry-and-why-1a0f3bb82507.

[21] Microsoft. *Registry Editor (regedit.exe)*. 2024. URL: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/regedit.

[22] Microsoft. *Sysmon Events*. 2025. URL: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon#events.

[23] Microsoft. *Win32 API Documentation: Registry Functions*. 2024. URL: https://learn.microsoft.com/en-us/windows/win32/sysinfo/registry.

[24] Microsoft Corporation. *Sysmon (System Monitor) – Sysinternals*. 2024. URL: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon.

[25] Microsoft Corporation. *winreg.h header - Win32 API Registry Functions*. 2025. URL: https://learn.microsoft.com/en-us/windows/win32/api/winreg/.

[26] Microsoft Learn. *Windows Registry Information for Advanced Users*. 2025. URL: https://learn.microsoft.com/en-us/troubleshoot/windows-server/performance/windows-registry-advanced-users.

[27] MITRE. *ATT&CK Technique T1047: Windows Management Instrumentation*. 2024. URL: https://attack.mitre.org/techniques/T1047/.

[28] MITRE. *ATT&CK Technique T1059.001: PowerShell*. 2024. URL: https://attack.mitre.org/techniques/T1059/001/.

[29] MITRE ATT&CK. *Persistence (TA0003)*. 2024. URL: https://attack.mitre.org/tactics/TA0003/.

[30] MITRE Corporation. *CALDERA – A Scalable, Automated Adversary Emulation Platform*. 2025. URL: https://caldera.mitre.org/.

[31] MITRE Corporation. *MITRE ATT&CK®*. 2025. URL: https://attack.mitre.org/.

[32] Maryam Mozaffari, Abbas Yazdinejad, and Ali Dehghantanha. *Windows-APT 2025: A Dataset for APT-Inspired Attack Scenarios on Windows Systems*. 2025. URL: https://data.mendeley.com/datasets/b8fmtzvpy8/2.

[33] Bryan Muehlberger. *Console Registry tool for Windows*. 2003. URL: https://www.computerworld.com/article/1377867/console-registry-tool-for-windows.html.

[34] PowerShell Empire Project. *PowerShell Empire – Building an Empire with PowerShell*. 2025. URL: https://www.powershellempire.com/.

[35] Khaled Rahal, Arbia Riahi, and Thibault Debatty. *Dataset of APT Persistence Techniques on Windows Platforms Mapped to the MITRE ATT&CK Framework*. 2025. URL: https://ieeexplore.ieee.org/document/10943025.

[36] Khaled Rahal et al. *Uncovering Malicious Persistence: Machine Learning-Based Detection of Windows Scheduled Tasks*. 2025. URL: https://cybersecurityjournal.info/uploads/archivepdf/3962Uncovering_Malicious_Persistence__Machine_Learning_Based_Detection_of_Windows_Scheduled_Tasks__Latest%20(2)%20(1).pdf.

[37] Red Canary / Atomic Red Team Community. *Atomic Red Team – Adversary Emulation Tests*. 2025. URL: https://www.atomicredteam.io/atomic-red-team.

[38] scikit-learn developers. *Metrics and scoring: Classification metrics*. 2026. URL: https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics.

[39] scikit-learn developers. *sklearn.decomposition.TruncatedSVD — scikit-learn 1.4.2 documentation*. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html.

[40] Software Engineering Institute, Carnegie Mellon University. *GHOSTS: Realistic User Simulation Framework for Cyber Experimentation, Simulation, Training, and Exercise*. 2025. URL: https://github.com/cmu-sei/GHOSTS.

[41] Splunk. *From Registry With Love: Malware Registry Abuses*. 2023. URL: https://www.splunk.com/en_us/blog/security/from-registry-with-love-malware-registry-abuses.html.

[42] J. A. L. Starink, Andrea Continella, and Marieke Huisman. *Dynamic Detection and Classification of Persistence Techniques in Windows Malware*. 2023. URL: https://www.utwente.nl/en/eemcs/scs/education/assignments/finished-assignments/master/20230526-dynamic-detection-and-classification-of-persistence-techniques-in-malware/.

[43] Antonio Villalon-Huerta, Hector Marco-Gisbert, and Ismael Ripoll-Ripoll. *A Taxonomy for Threat Actors' Persistence Techniques*. 2022. URL: https://www.sciencedirect.com/science/article/pii/S0167404822002498.

[44] Wikipedia contributors. *Multilayer perceptron*. 2026. URL: https://en.wikipedia.org/wiki/Multilayer_perceptron.

[45] Wikipedia contributors. *Tf–idf — Wikipedia, The Free Encyclopedia*. 2025. URL: https://en.wikipedia.org/wiki/Tf-idf.

[46] Wikipedia contributors. *XGBoost (eXtreme Gradient Boosting)*. 2026. URL: https://en.wikipedia.org/wiki/XGBoost.

Declaration of honour concerning compliance with referencing rules and the use of generative AI as part of the Master's thesis

I, the undersigned, Ghita BENNOUNA, student in Master of Science in Industrial Engineering, specialization in Computer Science, declare on my honour that all the sources used in the context of my Master's thesis comply with the rules of referencing as set out in the study regulations of the Haute École ICHEC-ECAM-ISFSC and explained in the Master's thesis writing guide.

Generative artificial intelligence tools have been used ethically and responsibly for the following purposes :

Type of use of generative AI		**Check box**
No use of AI		
Searching for information	Use as a research tool to explore a theme and locate relevant sources and content	X
Assistance with text revision	Use of an AI tool to correct the spelling, grammar and/or syntax of the text	X
	Reformulating a personal text using AI	X
Text generation assistance	AI-generated content, reworked in a personal way at a later time	X
	Complete generation of a section of text using AI, without any personal input	
Text translation assistance	Using AI to translate a text not included in the work	
	Using AI to translate text integrated into the work	X
Assistance with the production of visuals	Using AI to create visuals, graphics or images	
Other uses (to be completed)	Assistance with code generation and revision	X

The AI tools used in this work are as follows (to be completed) :

- Chatgpt
- Perplexity

I undertake to respect these declarations and to provide any additional information required concerning the use of AI in this work :

- I have appended the standard questions asked of the AI.
- I put together a file containing all the questions and answers I got from the AI and I insert the link to share them in my TFE.
- I can explain what type of assistance I have used and for what purpose.

Done at Brussels , on 05/01/2026

Signature : Ghita Bennouna

CAHIER DES CHARGES RELATIF au TRAVAIL DE FIN D'ETUDES de

Ghita Bennouna inscrite en MA2 IN

Année académique : 2025/2026

Titre provisoire : AI Detection of APTs' Persistence via Windows Registry

Objectifs à atteindre :

- Simulation d'attaques APT (clés de registre sous Microsoft Windows)
- Création d'une dataset
- Pretraitement de la dataset
- Conception et entrainement d'un modele IA
- Validation et évaluation du modele

Principales étapes :

1. Simulation (lab)

- Créer des machines virtuelles (Windows 10 & 11)
- Implémenter les techniques T1112, T1547,T1546...
- Collecter les event logs : Sysmon + Windows Event logs

2. Dataset

- Exporter les logs (evtx → Parquet/CSV)
- Labling de la dataset
- Pretraitement
- Effectuer le feature enginneering

3. Modèle IA

- Tâche : classification binaire
- Modeles : Random Forest / XGBoost ..
- Évaluer : Recall/F1 , matrice de confusion, FPR, FNR, Feature importance..

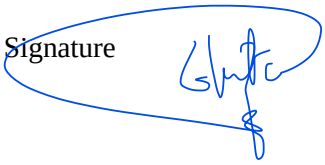
4. Validation & comparaison

- Sélectionner des datasets externes
- Évaluer les modèles selon les mêmes métriques
- Réaliser des tests statistiques pour confirmer les différences significatives

Fait à Bruxelles, le 13/10/2025

L'Etudiant

Nom-prénom : Bennouna Ghita

Signature 

Le Promoteur

Nom-prénom : DEBATTY Thibault

Entreprise : ERM

Signature