MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

Computer Science and Information Technology

PhD THESIS
– ABSTRACT –

Models and Techniques for Identifying and Neutralizing Malicious Applications

PhD Student:
Eng. Radu-Marian Portase

PhD Supervisor:
Prof. Eng. Gheorghe SEBESTYEN-PAL, PhD

Examination committee:

Chair: Prof. Eng. **Mihaela Dînşoreanu**, PhD – Technical University of Cluj-Napoca;
PhD Supervisor: Prof. Eng. **Gheorghe Sebestyen-Pal**, PhD – Technical University of Cluj-Napoca;

Members:
- Prof. Eng. **Bogdan Groza**, PhD – Politehnica University Timişoara;
- Assoc. Prof. Eng. **Haller Piroska**, PhD – University of Medicine, Pharmacy, Science, and Technology "George Emil Palade" of Târgu Mureş;
- Prof. Eng. **Vasile Teodor Dădârlat**, PhD – Technical University of Cluj-Napoca.

– Cluj-Napoca –
2024

Contents

1. Introduction 4
1.1 Context 4
1.2 Research objectives 4
2. Theoretical Background and State of the Art 5
2.1 Malware Detection 5
2.2 Definitions for Application Behaviour 5
2.3 Techniques for Application Monitoring on Windows 5
2.4 Deciding if Application Behavior is Malicious 6
2.5 Machine Learning Concepts for Malware Detection 6
2.6 Classic Machine Learning Models 7
2.7 Deep Learning and Neural Networks 7
2.8 State of the Art in Applying Machine Learning to Detect Malware 8
2.9 Datasets for Malware Detection 8
2.10 Malware Detonation in Sandboxes 9
2.11 Adversary Emulation 9
2.12 Multi-process Attacks against Malware Detectors 9
2.13 Performance Tests 10
3. Modelling Process Behavior for Malware Detection 10
3.1 LEDA - Layered Event-based Malware Detection Architecture 10
3.2 Processes as Series of Events 12
3.3 Extracting Behaviour Data Using the Curator Sandbox 12
3.3.1 Labelling Behaviour and Removing Noise 13
3.3.2 Building and Evaluating Datasets Using Curator 14
3.3.3 Measuring the Impact of Curator's Labelling Strategy 14
3.4 SpecRep Adversary Emulation System 15
3.4.1 Emulation System Architecture 15
3.4.2 Formalised Attack Specification 16
3.4.3 Compiling Metalanguage to Attack Specification 17
3.4.4 Execution of Internal Attack Specifications 17
4. Multi-Stage Detection of Malicious Processes During Execution 18
4.1 Multi-Stage LEDA based on Reasonable Doubt 18
4.1.1 Dynamic Feature Extraction 19
4.1.2 Static Feature Extraction 19
4.1.3 Machine Learning Models 19

4.1.4 Evaluation Based on "Reasonable Doubt" 19
4.1.5 Experiment: Multi-Stage LEDA Detection Capabilities 20
Performance Test: Multi-Stage LEDA with Reasonable Doubt vs Other Trigger Configurations 20
4.2 Automated Learning of Detection Triggers 21
4.2.1 Training Methodology 21
4.3 Metrics for the Current Version of LEDA 21
4.3.1 Experiment: Detection on the LEDA Benchmark Dataset 22
4.3.2 Performance Test: Evolution of Performance Impact 22
5. Multi-Process Attacks Against Malware Detectors 22
5.1 Code Injection for Defence Evasion 23
5.2 Group-Aware LEDA 23
5.3 Monitoring Techniques for COM, WMI, and ALPC-Based Multi-Process Malware 24
5.4 Metrics for the Current Version of LEDA 26
5.4.1 Experiment: Detection on the LEDA Benchmark Dataset 26
5.4.2 Performance Test: Evolution of Performance Impact 26
6.1 Persistent Entity Maps for Malware Detection 27
6.1.1 Introducing Entity Maps 27
6.1.2 Extension to LEDA to Accommodate Entity Maps 28
6.1.3 Building the Entity Map 28
6.1.4 Modified LEDA Algorithms for Entity Maps 29
6.1.5 Dealing with Persistence 29
6.1.6 Entity Description Language 29
6.1.7 Signature Rules and Matching 29
6.1.8 Examples: Entity Map Management 30
6.2 Neutralizing Malicious Applications After Execution 30
6.3 Metrics for the Current Version of LEDA 31
6.3.1 Detection Experiment on the LEDA Benchmark Dataset 31
6.3.2 Performance Test: Evolution of Performance Impact 32
7. Conclusions 32
7.2 Contributions of the Thesis 33
7.3 Correlation of Contributions with Objectives 34
8. List of Publications 35
8.1 US Patents 35
8.2 Publications in ISI Indexed Journals 35
8.3 Publications at International Conferences 36
References 36

1. Introduction

1.1 Context

Humans are deeply connected to computers and the Internet, handling activities like shopping, banking, and communication online, which attracts malicious actors. Malware has evolved from simple experimental viruses to a commercial industry, with five waves of development. Initially, malware was basic and error-prone, but it became more sophisticated with email worms, modern worms, and Advanced Persistent Threats (APTs), targeting data theft or sabotage.

As malware grows in complexity, security solutions have evolved, including advanced tools like Endpoint Detection and Response (EDR) and Extended Detection and Response (XDR). Machine learning is now routinely used to enhance real-time detection, helping security systems identify threats quickly based on incomplete activity logs. Timely detection is critical, especially against threats like ransomware, where delays can result in irreversible damage.

This thesis focuses on improving real-time malware detection, with incomplete logs, using machine learning, and restoring infected systems, with a focus on Windows platforms.

1.2 Research objectives

The objectives of this thesis focus on developing robust models and methodologies to detect and neutralize cyber threats. The primary goals include creating a structured view of process activity, providing solutions for process behaviour and cyber attack modelling, and training machine learning models to identify malicious processes. We design and build a platform for malware detection capable of effectively handling and neutralizing multi-process and multi-component malware. These efforts aim to enhance cybersecurity by improving the detection and mitigation of advanced persistent threats and evolving cyber attacks.

The main objectives can be broken down as follows:

Objective 1: Develop models for process activity and cyber attacks.
- *Objective 1.1:* Propose a new structured view of process activity.
- *Objective 1.2:* Create and implement a solution for malware detonation in a sandbox to capture process activity information.
- *Objective 1.3:* Propose a methodology to identify malfunctioning samples and remove noise.
- *Objective 1.4:* Propose a new structured view of cyber attacks.
- *Objective 1.5:* Create a solution for reproducing cyber attacks based on openly available white papers.

Objective 2: Train machine learning models to detect malicious processes.
- *Objective 2.1:* Build a platform to monitor process activity and detect malicious processes.
- *Objective 2.2:* Develop a feature extraction process for static and dynamic features.
- *Objective 2.3:* Train machine learning and deep learning models for malware detection.
- *Objective 2.4:* Create a system that determines the optimal times to evaluate a machine learning model.
- *Objective 2.5:* Study the temporal decay in the performance of our machine learning models.

Objective 3: Improve detection models to face multi-process attacks.
- *Objective 3.1:* Study how malware uses code injection to conceal malicious actions.

- *Objective 3.2:* Propose a solution to defend the model against code injection.
- *Objective 3.3:* Study how malware can use remote procedure calls to server processes to evade detection models.
- *Objective 3.4:* Propose defenses for the evasion tactics identified.

Objective 4: Develop detection and neutralization strategies for multi-component malware.

- *Objective 4.1:* Propose a model for advanced persistent threats that utilize multiple components deployed across multiple boot sessions.
- *Objective 4.2:* Create a detection system for multi-component malware.
- *Objective 4.3:* Develop a detection system capable of maintaining context across multiple boot sessions.
- *Objective 4.4:* Propose efficient methods to neutralize malware post-detection and identify the origin of the compromise on the machine.

2. Theoretical Background and State of the Art

2.1 Malware Detection

Malware detection involves assessing application behaviour to determine if it's malicious. Detection methods are classified as anomaly-based, specification-based, or signature-based, using data from static, dynamic, or hybrid approaches.

Static detection relies on file signatures but is easily evaded. Dynamic detection observes runtime behaviour, making it more proactive but still vulnerable to evasion techniques. Hybrid methods combine both for better accuracy.

This text focuses on dynamic detection as a key approach for identifying advanced malware and restoring systems, addressing the challenges of detecting malware during execution and system recovery.

2.2 Definitions for Application Behaviour

Researchers define application behaviour in two ways: as a sequence of actions impacting the operating system[2] or as data flows between system entities like processes and files [3]. Windows, being complex, allows many security-relevant actions, such as file system changes, code injection, or network activities. A subset of these actions, important for security, includes memory writing, process creation, and registry modifications.

The complexity of process behaviour makes it challenging to describe malware actions clearly. The MITRE ATT&CK framework was created to categorise adversary behaviours, initially focusing on Windows but expanding to other platforms. It organises cyber threats into Tactics, Techniques, Sub-Techniques, and documented procedures, providing a structured approach for threat analysis and response.

2.3 Techniques for Application Monitoring on Windows

Microsoft Windows, one of the most widely used operating systems, is a major target for malware. To counteract this, Microsoft invests heavily in security, developing advanced application behaviour monitoring methods.

Library call interception, the most basic method, is flexible but introduces stability and performance issues, and can be bypassed by malware. To improve reliability, Microsoft introduced *Windows Notification Routines*, which allow kernel-level monitoring of key activities, such as process creation and registry changes. These callbacks are more stable but limited in scope.

Windows Filter Drivers enable low-level monitoring by intercepting I/O operations, but due to performance challenges, higher-level solutions like *Filesystem Minifilters* are preferred. Minifilters monitor file system activities but can be bypassed by direct disk access.

For network monitoring, the *Windows Filtering Platform (WFP)* manages connection rules and inspects packets at multiple layers of the network stack. However, it can suffer from performance bottlenecks, especially with encrypted traffic.

Finally, *Event Tracing for Windows (ETW)* provides a robust framework for tracing system-wide activities. ETW allows detailed monitoring of application behaviour, particularly useful for detecting code injection and other out-of-process execution techniques used by malware. Each of these methods contributes to enhancing the security of Windows, though none are foolproof, requiring a combination of techniques for effective monitoring and detection.

2.4 Deciding if Application Behavior is Malicious

Research in dynamic malware detection follows three main approaches: graph-based analysis, threat scoring, and machine learning.

The first approach involves converting application behaviour into graphs and comparing them to known malicious samples. Some researchers unpack executables and extract system call graphs, while others prefer running processes in sandboxes for analysis. Variations like using data flows instead of system calls or adding clean actions aim to reduce false positives.

The second approach uses threat scoring, where behavioural characteristics are evaluated in real time and passed to a scoring engine. Systems like the one by Hajmasan et al. [4] aggregate scores across multiple actions and processes, improving accuracy and reducing false positives.

The third approach involves applying machine learning to detect malware.

This thesis utilises all three approaches where most applicable.

2.5 Machine Learning Concepts for Malware Detection

Machine learning techniques are mathematical models used to address complex problems like prediction, classification, and data conceptualisation. These models can be classified as unsupervised, supervised, or semi-supervised.

Unsupervised learning is typically used for exploratory data analysis, where natural groupings are unknown, or to validate training variables. It can cluster malware samples into families based on features. Supervised learning, on the other hand, involves training the

model with labelled data, applicable to malware detection and classification. Semi-supervised learning combines both methods, using a small amount of labelled data and a large amount of unlabelled data, improving model performance, especially when labelled data is limited.

In this thesis, *Scikit-learn* is the primary library used for machine learning experiments.

Key metrics for malware detection models include Accuracy, Precision, Recall, and F1 Score. Accuracy measures the proportion of correctly classified samples, but can be misleading with imbalanced datasets. Precision indicates how often benign samples are misclassified as malware, and Recall measures how effectively the model identifies actual malware. The F1 Score balances Precision and Recall, providing a more holistic view of the model's performance.

2.6 Classic Machine Learning Models

In this thesis, we experimented with Naive Bayes, Logistic Regression, and Random Forest models for malware detection.

Naive Bayes classifiers use Bayes' theorem with the assumption of conditional independence between features. Despite its simplicity, this method has proven effective in tasks like malware classification and spam filtering.

Logistic Regression is a linear model that predicts the probability of a binary outcome using a logistic function. Scikit-learn’s implementation of logistic regression supports L1 and L2 regularisation. In malware detection, where many features are used, L1 regularisation helps by reducing less important features' coefficients to zero, assisting in feature selection.

Decision Trees (DTs) are used to predict a target variable based on decision rules learned from the data. Random Forests extend decision trees by creating multiple trees from bootstrap samples and selecting splits from random feature subsets. This randomness reduces variance, improving overall model accuracy despite slightly increased bias.

2.7 Deep Learning and Neural Networks

Deep learning is a subset of machine learning that uses neural networks with many layers to automatically learn representations from data for tasks like classification and object detection. Unlike traditional models that rely heavily on feature engineering, deep learning models learn features directly from raw data.

A deep neural network (DNN) typically consists of an input layer, multiple hidden layers, and an output layer. The input layer receives a feature vector, while the output layer generates class probabilities based on the processed input.

Hidden units, the core of neural networks, process feature vectors and pass the result to the next layer. Each hidden layer refines the data representation, progressively extracting more abstract and meaningful features.

Random projection is a dimensionality reduction technique that preserves the structure of high-dimensional data by projecting it into a reduced space using a sparse, randomly initialised matrix. This helps manage large feature sets efficiently.

2.8 State of the Art in Applying Machine Learning to Detect Malware

Machine learning has been widely applied to malware detection, focusing on dynamic behaviour analysis. Key developments in classic and deep learning techniques have shaped current approaches.

Early work, such as Schultz et al.[5], pioneered the use of classifiers like Naive Bayes and decision trees to detect malware, outperforming traditional signature-based methods. Over time, techniques such as unsupervised learning, clustering, and *N*-grams analysis were developed, leading to more advanced models like Support Vector Machines (SVMs)[6]. Ucci's 2017 survey identified challenges like feature selection and anti-analysis techniques, which this thesis addresses through improved datasets and methods to counter malware evasion[7].

Deep learning, emerging between 2010 and 2015, has since become a key approach in malware detection. Researchers like Dahl et al.[8] and Kolosnjaji et al.[9] used neural networks to classify malware based on system call sequences, while hybrid models combining convolutional and feedforward layers offered enhanced detection. Techniques such as stacked autoencoders (SAEs) and recurrent neural networks (RNNs) further advanced the field[10][11].

Companies such as Carbon Black[12], CrowdStrike[13], and Microsoft[13] integrated machine learning into their next-gen anti-malware products, focusing on behavioural detection. Microsoft's Windows Defender Advanced Threat Protection combines local and cloud-based models for enhanced real-time malware detection[15].

Recent real-time detection systems use sandbox environments to collect logs and train machine learning models. Systems like AMAL[16] and advanced techniques, such as Gated-CNNs and Bi-LSTMs, provide real-time capabilities, though techniques like sliding windows remain vulnerable to malware evasion strategies[17].

2.9 Datasets for Malware Detection

Data is essential for machine learning in malware detection. Three major datasets used in this field are the Microsoft Malware Classification Challenge, EMBER, and SOREL-20M.

The Microsoft Malware Classification Challenge dataset (2015)[18] contains 20,000 malware samples with bytecode, disassembly, and metadata like function calls and strings. EMBER (2018)[19] includes features from 1.1 million binary files, split evenly between malicious and benign samples, focused on static analysis of PE files. SOREL-20M (2020)[20] offers 20 million PE files, with disarmed malware samples, designed for robust training with low false positive rates.

However, none of these datasets support dynamic analysis, making them suitable for static classifiers but not for dynamic classifiers. Behaviour traces must be collected through sandbox detonations.

2.10 Malware Detonation in Sandboxes

Sandboxes are key tools in malware analysis, offering safe environments to observe malware behaviour. CWSandbox (2007)[21] used API hooking to monitor system calls. ANUBIS (2009)[22] improved on this by running malware in an emulated Windows XP and providing reports on system calls, APIs, and network traffic.

Today, Cuckoo Sandbox[23] is the most widely used, offering detailed behavioural reports for various operating systems. However, sandboxes have limitations, like relying on pre-installed monitoring solutions. To address this, we developed our own sandbox, Curator, for more flexibility.

Though effective, sandboxes can be fingerprinted and evaded[24]. To mitigate this, we also emulate malicious behaviour based on forensic reports.

2.11 Adversary Emulation

Adversary emulation emerged as a research focus in the late 1990s, beginning with taxonomies for attackers, attacks, and defences. Cohen's work on attack classification and simulation[25][26] was foundational. Park et al. extended this with SECUSIM[27], a simulator for defining attacks and analysing defences. Both early models focused on high-level events without producing detailed sensor output.

Kotenko et al.'s "Attack Simulator"[28] introduced attack emulation using probabilistic steps, such as Enumeration and Privilege Elevation, which foreshadowed the MITRE ATT&CK framework.

Today, MITRE ATT&CK is central to adversary emulation. Ajmal et al.[29] introduced a threat-based approach, simulating real-world APTs. Oh et al.[30] applied deep reinforcement learning (DRL) to train AI agents for cyber-attack simulations.

Several emulation tools exist, including MITRE's CALDERA[31] and CASCADE[32], which automate attack simulations and post-attack investigations. Atomic Red Team[33] and APTSimulator[34] offer simple, ATT&CK-mapped tests, while Breach and Attack Simulation (BAS) tools[35] automate security assessments.

In this thesis, we propose our own system, SpecRep, for modelling and reproducing cyberattacks.

2.12 Multi-process Attacks against Malware Detectors

The security research community has long studied malware evasion techniques, including multi-process and out-of-process execution to avoid detection.

Ji et al.[36] demonstrated a multi-process evasion mechanism that separates command-and-control (C&C) connections from malicious behaviours, reducing detection rates. Ispoglou et al.[37] created malWASH, which hides malware in benign processes, ensuring continuity even if some processes are terminated. Ma et al.[38] developed a tool to split malware code across "shadow" processes to maintain its behaviour.

Previous approaches mainly relied on detectable code injection. Our work explores using COM, WMI, and RPC for multi-process execution, leveraging native Windows functionalities for greater

stealth. COM, first developed by Microsoft in 1993, allows programs to interact with components across processes. COM's potential for anti-malware evasion has been discussed extensively, including works like Thompson's[39], Eddon's[40], and more recent research by Forshaw[41]. Additionally, frameworks like Koadic[42] have shown how COM can be weaponised for defence evasion.

WMI is another key technology used for post-reboot persistence and evasion, as highlighted in Graeber's[43] work. Defence strategies against WMI attacks focus on post-compromise detection[44][45], but WMI's interaction through COM complicates this process.

2.13 Performance Tests

Testing anti-malware performance is key to balancing security and usability. While security focuses on detecting and neutralising threats, usability measures the software's resource impact and user experience.

Absolute performance impact is the measurable effect on system resources like CPU, memory, and disk usage, tested via benchmarks. User-perceived impact reflects how noticeable any slowdowns are during daily tasks like browsing or gaming.

For this thesis, we tested our solution on an Intel Core i7-11700, 32 GB RAM, NVMe SK Hynix 512GB disk running Windows 11. Background processes were minimised by disabling non-essential services and Windows Defender.

We used two tests: micro-benchmarks and PCMARK scores. Each micro-benchmark was run 12 times, with results averaged after discarding the highest and lowest. The error margin was +/- 0.5%.

3. Modelling Process Behavior for Malware Detection

Modelling process behaviour for malware detection identifies anomalies by comparing real-time actions to a baseline of normal and malicious activities, such as file access, network usage, and memory patterns. Data is first gathered in a sandbox, though challenges like missing dependencies or sandbox detection can hinder sample execution.

Relevant features are extracted from the data, and a model is built using machine learning to differentiate between normal and malicious behaviour. This model is then integrated into real-time monitoring for proactive threat detection.

We introduce LEDA (Layered Event-based Malware Detection Architecture) and our sandbox Curator for data collection. For samples that can't run in virtual machines, we developed SpecRep, which uses large language models to create executable attacks based on adversary reports.

3.1 LEDA - Layered Event-based Malware Detection Architecture

We propose a novel detection system architecture called Layered Event-based Malware Detection Architecture (LEDA) (Figure 1). It collects data from an operating system, processes it, and uses the extracted information to determine if a monitored process is malicious.

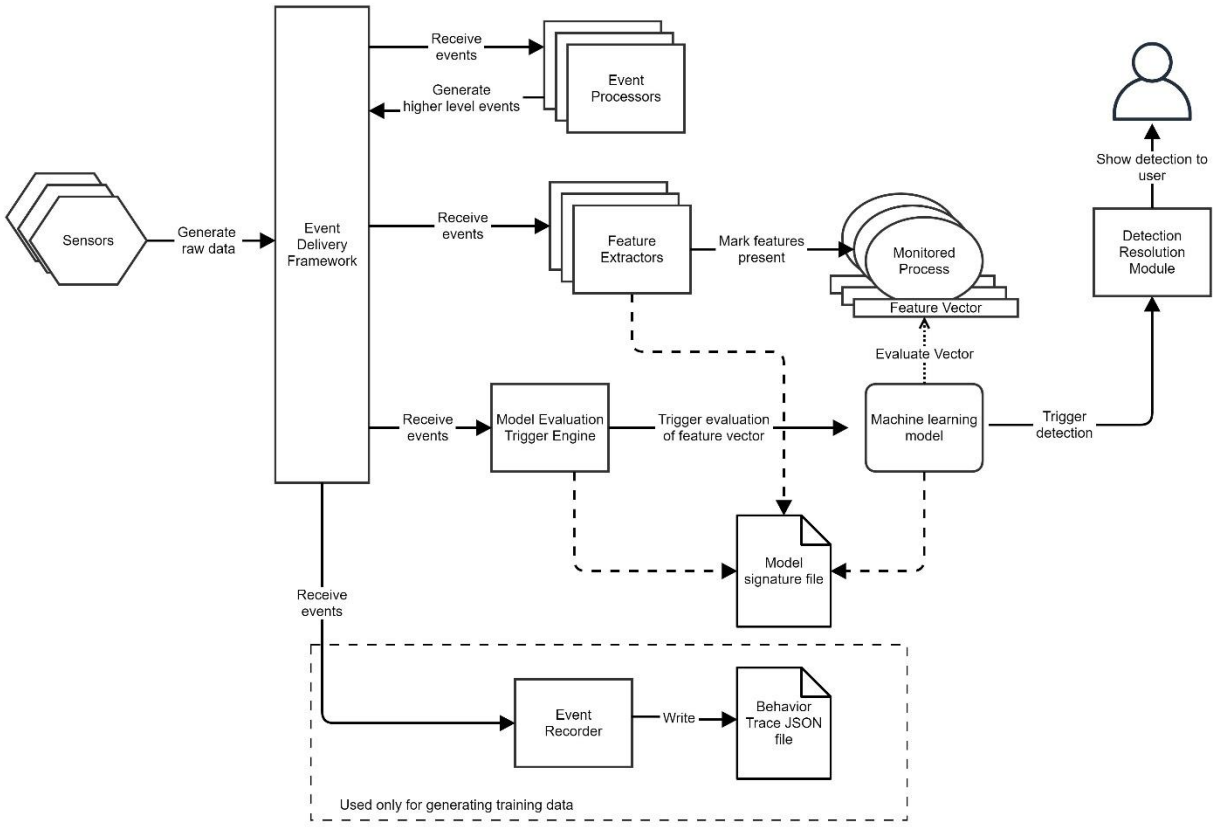


Figure 1:Leda System Architecture

LEDA operates on an **event delivery framework** where decoupled **event producers** generate events, and event consumers process them. The framework supports two dispatch policies: synchronous and asynchronous, the latter helping reduce performance impact. There are two types of events: infrastructure events, which support system functions, and behavioural events, which represent process behaviours.

Sensors gather raw event data from the operating system and submit it to the framework. **Event processors** consume these events, aggregate them, and may generate more complex events. **Feature extractors** are a special class of event processors that evaluate events and generate feature vectors for processes.

Predicates are simple logical operations applied to event fields to generate features. Once the system accumulates enough features, machine learning models (e.g., random forests, Markov chains, or neural networks) evaluate the feature vectors. These models are triggered based on specific event conditions.

LEDA is highly configurable. Processors can be enabled or disabled via configuration files, and the system avoids unnecessary overhead by skipping event dispatching for unregistered events. Finally, a detection resolution module represents typical security actions like blocking malware or starting disinfection processes.

3.2 Processes as Series of Events

LEDA represents each monitored process as an ordered set of events generated by sensors or event processors:

$$P = \{e_0, e_1, \ldots, e_n \mid e_i \text{ is an event captured by LEDA for the process } \}$$

Events are ordered by timestamp, with the first event, e_0, being the process creation event. We categorise behavioural events into three layers based on complexity:

- **Layer 0**: Raw sensor data capturing how actions were performed (e.g., API calls).
- **Layer 1**: Processed data revealing what actions were performed (e.g., file copying).
- **Layer 2**: Contextual events identifying the impact and intent (e.g., copying a file to the startup folder).

For example, if a process copies a file to the Windows startup folder, sensors generate Layer 0 events, such as API calls or file reads/writes. These are processed into a Layer 1 event, like FILE_COPY, and further evaluated to produce a Layer 2 event, such as FILE_COPY_TO_STARTUP_FOLDER.

Machine-specific data in events, such as file paths or process IDs, can hinder detection. We address this by applying a NORMALIZE function that replaces machine-specific values with standardised ones, allowing us to define a normalized process representation P':

$$P' = \{e' \mid e' = \text{NORMALIZE}(e),\ e \in P\}$$

One key challenge in building LEDA was filtering out noise generated by the detonation environment, which could otherwise interfere with the detection process. We define a denoised process P as:

$$\hat{P} = P' - Noise$$

Our goal is to automatically compute Noise for each operating system version to reduce unnecessary noise in the detection process.

3.3 Extracting Behaviour Data Using the Curator Sandbox

The only reliable way to collect behaviour traces from malware executions is through sandbox detonation, where malware is executed in a controlled virtual environment. In our *LEDA* architecture, an **Event Recorder** logs all events generated during detonation into a Behaviour Trace JSON for further analysis.

Samples are run in a modified version of LEDA with the event recorder enabled to capture behaviour during execution.

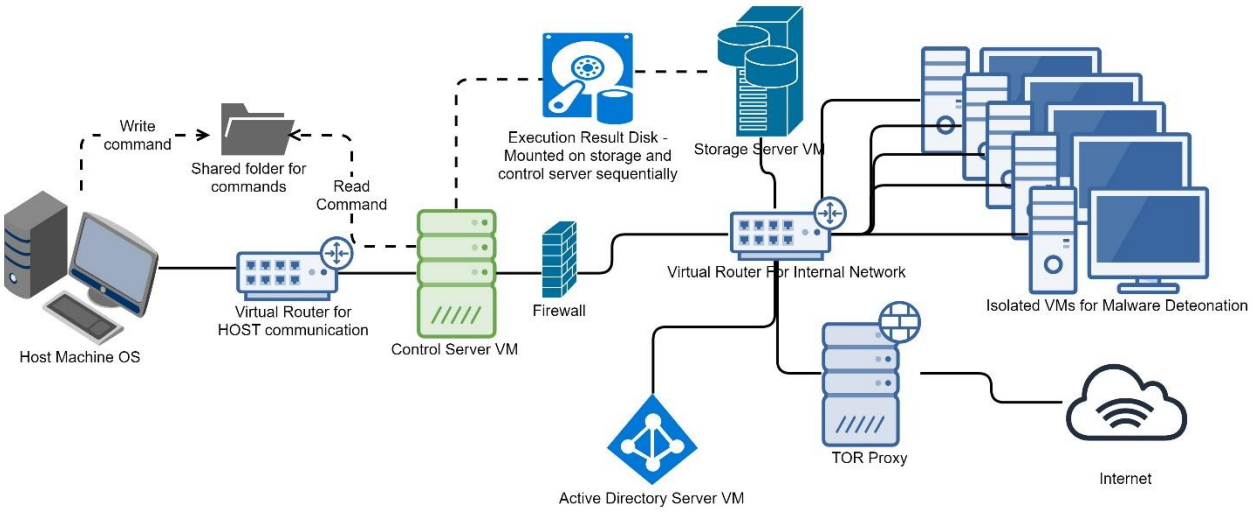


Figure 2: Curator Architecture

To support this, we developed a custom sandbox named Curator (Figure 2), designed to execute malware for at least one minute, connected to the internet through TOR to prevent exposure. Curator uses a virtualised setup with multiple VMs simulating a real network environment, including Active Directory (AD) servers and storage servers. This setup allows malware to interact with network services, enhancing the realism of the detonation environment.

3.3.1 Labelling Behaviour and Removing Noise

Accurately labelling behaviour and removing noise is crucial for building reliable malware detection models. Two key challenges are the *oracle problem* and the *malfunctioning sample problem*.

Oracle Problem
This problem refers to how to accurately label malware in large datasets. While VirusTotal serves as a primary source for obtaining malware labels, we must be cautious about relying solely on its results to avoid false positives.

Malfunctioning Sample Problem
Malware may not exhibit its full behaviour during detonation due to various issues, such as missing dependencies or disabled command-and-control servers. Even if a file is labelled as malicious, its behaviour during detonation may resemble clean processes.

Solutions
To handle these challenges, we developed a series of calibration programs to generate "normal" behaviour profiles and filter out noise. Using these profiles, we apply a denoising process to all events generated by LEDA during detonation.

Machine Calibration Algorithm: This algorithm runs each calibration program in the sandbox to generate a set of known normal events. These events are processed and normalised, and the shared events across all programs are identified as noise. The remaining behaviour is stored as "known behaviour profiles," which help filter out irrelevant events.

Behaviour Labelling Algorithm: After denoising the process behaviour, this algorithm checks if the denoised events match any known behaviour profiles. If no match is found, the algorithm searches for indicators of compromise (IOCs) in the form of file hashes or URLs to assign a label to the process based on its malicious activity.

IOC Labelling Algorithm: This algorithm extracts IOCs from processes and queries VirusTotal to identify potential malware families. It filters out unreliable detection labels (e.g., "Generic" or "Heuristic") and consolidates the remaining labels to determine the most likely malware type and family.

We modified the AVClass2 algorithm to handle false positives and consolidate detection labels. This approach allows us to maintain accuracy when dealing with large datasets of malware samples.

3.3.2 Building and Evaluating Datasets Using Curator

We collected 100,000 potentially malicious samples and 10,000 clean executables. After checking with VirusTotal, we discarded 23.5% due to low detection confidence and 6.2% classified as Potentially Unwanted Applications (PUA). After five months, we re-scanned the samples and found that 28.2% of the remaining samples malfunctioned during detonation. This left us with 50,346 samples suitable for creating malware detection datasets.

3.3.3 Measuring the Impact of Curator's Labelling Strategy

We trained *Random Forest* models using four datasets:

1. **Curator**: 50,346 valid samples.
2. **Oracle**: Curator samples + samples with oracle issues (80,217).
3. **Malfunctioning**: Curator samples + malfunctioning samples (70,129).
4. **Complete**: All 100,000 samples.

Each dataset was split 70% for training and 30% for validation. Results showed that models trained on the Curator dataset outperformed the others.

Table: RandomForestClassifier Results

Dataset	*Testing Set (M+C)*	*FP*	*FN*	*FP+FN*	*Accuracy*
Complete	30,000 + 3,000	426	10,683	11,109	0.66336
Oracle	24,065 + 3,000	104	7,943	8,047	0.70268
Malfunctioning	21,038 + 3,000	354	8,421	8,775	0.63495
Curator	50,346 + 3,000	0	230	230	**0.99569**

Conclusions

- The **Curator** dataset consistently outperformed all other datasets across models, achieving the highest accuracy (99%).
- The **Complete**, **Oracle**, and **Malfunctioning** datasets resulted in poorer performance due to the inclusion of mislabeled or malfunctioning samples.
- This experiment validates *Curator's* effectiveness in building reliable datasets for behaviour-based malware detection.

3.4 SpecRep Adversary Emulation System

Curator's main limitation is its inability to generate behaviour for malfunctioning samples and those requiring commands from a C&C server, such as advanced backdoors. These backdoors allow remote access and modify their behaviour based on external commands, making them critical for detection.

To address this, we developed *SpecRep*, an adversary emulation system. Initially, SpecRep used manually crafted Python scripts to replicate attack steps, generating different behaviours for backdoors. However, scripting every variation of an attack was impractical, given the numerous methods attackers can use to achieve the same goal.

We shifted to an objective-based approach, focusing on what attackers aim to achieve rather than how they act. This made SpecRep more flexible, allowing it to emulate a wider range of attacks and fully explore backdoor behaviours within varied attack contexts.

3.4.1 Emulation System Architecture

SpecRep's architecture is built around achieving attacker objectives in multiple ways, with its flow illustrated in Figure 3.

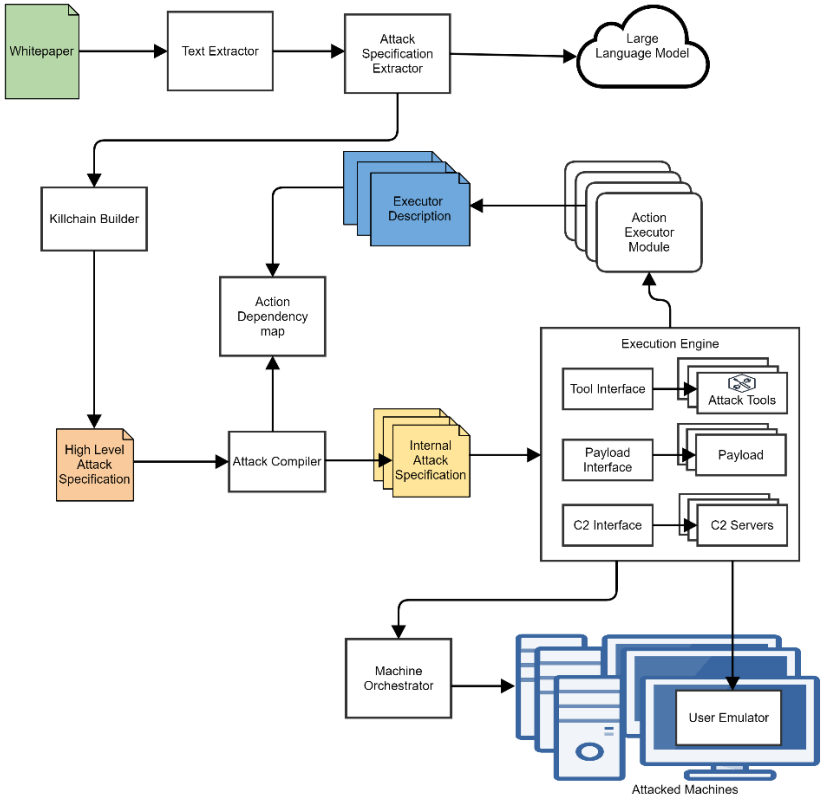


Figure 3: SpecRep System Architecture

System Overview:

1. **High-level Attack Specifications (HLAS)**: These specifications, written in a custom metalanguage, describe attack objectives rather than methods. Text extractors process

white papers to convert relevant information into HLAS using the OpenAI API, which extracts attacker objectives mapped to MITRE techniques. The *Attack Specification Extractor* and *Killchain Builder* then create one or more plausible attack scenarios.

2. **Internal Attack Specifications (IAS)**: The second phase compiles HLAS into executable IAS format, creating all possible combinations of actions that achieve the attack objectives. These actions are defined in *Executor Description* files.
3. **Execution Engine**: The final part executes the IAS on test machines, using open-source tools like Metasploit and PowerShell Empire, or custom tools. Actions requiring user interaction (e.g., opening an email) are performed by a *User Emulator*.

3.4.2 Formalised Attack Specification

SpecRep's attack descriptions are written in a simplified metalanguage. This language specifies machines, users, stages, and objectives.

- The **Machines** section defines the devices involved in the attack, such as computers or IoT devices, along with properties like IP address and operating system.
- The **Users** section describes users performing actions during the attack, which may be tied to different roles in an Active Directory.
- The **Stages** section models the timeline of the attack, ensuring the logical sequence of objectives. For example, an attacker must gain initial access before registering persistence.

```
<Specification>  ::= <Machines> [<DefaultMachine>] <Users>
    [<DefaultUser>]
                     <Stages>

<Machines>       ::= <Machine> | <Machine> <Machines>
<Machine>        ::= <Name> <Ip> <Type> <Operating System>
<DefaultMachine> ::= <Name>

<Users>          ::= <User> | <User> <Users>
<User>           ::= <Name> [<Type>]
<DefaultUser>    ::= <Name>

<Stages>         ::= <Stage> | <Stage> <Stages>
<Stage>          ::= <Objectives> | <UserActions>

<Objectives>     ::= <Objective> | <Objective> <Objectives
   >
<Objective>      ::= <ObjectiveCategory> <Type> [<
   TargetMachine>]
                     [<TargetUser>] [<PayloadName>] [<
   TargetPayloadName>]
<Type>           ::= <TypeName> | Any

<UserActions>    ::= <UserAction> | <UserAction> <
   UserActions>
<UserAction>     ::= <Action> [<TargetMachine>] [<
   TargetUser>]
                     [<TargetPayloadName>]
```

Figure 4: SpecRep Attack specification Language

Each objective has a category (high-level goal) and a type (specific implementation). When compiled, all combinations of objectives within a stage are executed, ensuring varied attack flows.

3.4.3 Compiling Metalanguage to Attack Specification

The *Attack Compiler* translates high-level attack objectives into internal attack specifications. These are interpreted by the execution engine to perform the actions. The compiler builds an *action dependency map* (example in Figure 5) by identifying relationships between actions and their dependencies (i.e., what is required to achieve each objective).

The compilation process uses pathfinding in the dependency map to generate all possible attack flows based on the specified stages. For each valid path, the compiler outputs an internal attack specification.

Example:
If an attacker wants to perform user discovery on vulnerable machines, the compiler generates multiple flows depending on available actions like scanning for vulnerabilities and using tools like Metasploit or Koadic for lateral movement.

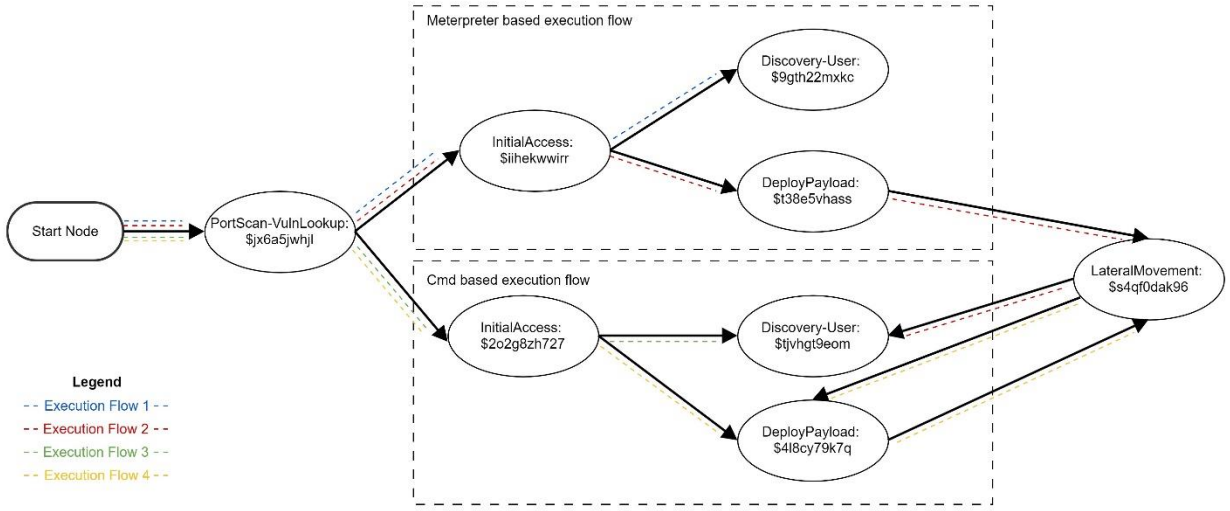


Figure 5: Example of an Action Dependency Map

3.4.4 Execution of Internal Attack Specifications

Once compiled, the internal attack specification is passed to the execution engine, which orchestrates the test machines and performs the attack. Each machine is equipped with a *User Emulator* to simulate user actions if required.

4. Multi-Stage Detection of Malicious Processes During Execution

Detecting malicious processes in real-time is challenging as traditional models rely on post-execution logs and lack full behavioural context during execution.

We developed a machine-learning model integrated with LEDA to detect malicious activity during execution without requiring complete logs. Inspired by MtNet[11], we combined dynamic features from LEDA with static features from process memory to counter malware evasion techniques, like encryption.

LEDA introduces the concept of denoised event representation, defined as:

$$\widehat{P} = \{e'_0, e'_1, \ldots, e'_n \mid e'_i \text{ is a normalized event}\}$$

and uses "detection triggers" to evaluate a process P's behaviour dynamically:

$$\text{Trigger}(P) = \{t_0, t_1, \ldots, t_n \mid t_i \in \widehat{P} \text{ when evaluating } P\}$$

Initially, we set Trigger(P)=P , but this caused performance issues due to excessive memory processing. We then applied the concept of "reasonable doubt" by manually selecting critical events, which performed well in tests. Later, we dynamically learned these triggers and tested them over time and across infection sources.

This chapter details the architecture and results of our approach.

4.1 Multi-Stage LEDA based on Reasonable Doubt

We present a high-level overview of the proposed solution in Figure 6. ocusing on the key components for detection. LEDA integrates both user mode and kernel mode **sensors** (yellow in the diagram). In kernel mode, a Mini-Filter driver registers callbacks for changes in the file system, registry, or process creation. User mode uses API hooking via a DLL, and also intercepts actions using COM, WMI, and process injection. Network events are hooked, avoiding the need to decrypt communication.

Event processes consume data from sensors, providing information to users and detecting actions matching the MITRE matrix. High-level events are sent to a custom evaluation trigger engine, the **Reasonable Doubt engine**, **behavioural feature extractors**, and a **LOG Collector**. The Reasonable Doubt engine determines when to evaluate a machine learning model. Feature extractors build a feature vector *B[]* for the detection model. The log collector gathers memory data during sample detonation and dumps memory as needed.
In detonation mode, LEDA extends the process view $\hat{H}P$ with memory snapshots *MemorySnapshots(P)* taken at moments *Trigger(P)*. Features are extracted from $\hat{H}P$ and *MemorySnapshots(P)* for model training.

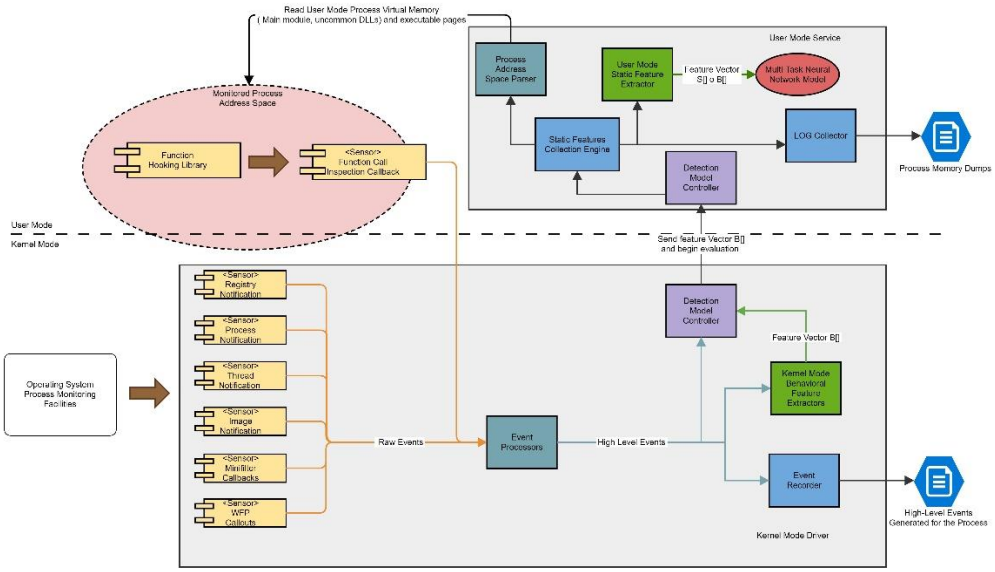


Figure 6:Multi-Stage LEDA

4.1.1 Dynamic Feature Extraction

LEDA uses the concept of **reasonable doubt**, relying on expert knowledge to select triggers *Trigger(P)* manually. Dynamic features, based on MITRE ATT&CK, correlate low-level events into high-level indicators of behaviour or techniques. The system generates 1241 high-level events, most tied to MITRE tactics.

4.1.2 Static Feature Extraction

Static features of process *P* are extracted from *MemorySnapshots(P)*, which capture decrypted code in memory. The **Process Address Space Parser** extracts features by analyzing memory layout, filtering out DLLs and injected monitoring components. The parsing process is shown in Algorithm 1.

4.1.3 Machine Learning Models

We use a classical machine learning model (Naive Bayes, Logistic Regression, or Random Forest) alongside a deep neural network (DNN). Features from N-grams of MITRE techniques and strings from memory are used as input. The DNN architecture, depicted in Figure 2, is trained using random projection to reduce input features from 50,000 to 4,000. The hidden layers employ ReLU activations, and the loss function is cross-entropy. The model uses multi-task learning, with a weighted multi-task loss.

4.1.4 Evaluation Based on "Reasonable Doubt"

LEDA can evaluate models at various stages, including a "reasonable doubt" strategy. This strategy triggers model evaluation during significant MITRE ATT&CK tactics like Persistence, Privilege Escalation, and Lateral Movement. This approach is efficient and matches the performance of full-log evaluations.

4.1.5 Experiment: Multi-Stage LEDA Detection Capabilities

We selected 50,000 malicious and 10,000 clean Windows PE samples for this experiment. Malicious samples were sourced from VirusTotal, VX Heaven, Malshare, VirusShare, and a private collection. .NET samples were excluded due to complexities in scanning just-in-time compiled code.
Using Curator, we labeled 79.2% of the malware samples, identifying 288 malware families. The remaining malware was labeled *GenericMalware*, and all clean samples were labeled *GenericClean*. The dataset was split as follows:

- **Training:** 42,000 (35,000 malicious, 7,000 clean)
- **Validation:** 9,000 (7,500 malicious, 1,500 clean)
- **Test:** 9,000 (7,500 malicious, 1,500 clean)

We compared detection accuracy between full log evaluation after 1 minute and the *Reasonable Doubt* approach. The results are summarized below.

Table: Reasonable Doubt Evaluation vs. Full System Logs

Model	*1min*	*rdF*	*rdL*	*1min*	*rdF*	*rdL*	*1min*	*rdF*	*rdL*
Naive Bayes									
N = 1	0.99	0.56	0.85	0.92	0.56	0.89	0.91	0.67	0.89
N = 2	0.99	0.61	0.89	0.91	0.69	0.89	0.92	0.78	0.95
N = 3	0.99	0.64	0.89	0.90	0.74	0.87	0.93	0.79	0.95
N = 4	0.99	0.61	0.89	0.91	0.73	0.85	0.92	0.73	0.91
N = 5	0.99	0.61	0.89	0.89	0.73	0.84	0.89	0.73	0.85

Random Forest with N=3 performs best under *Reasonable Doubt*, achieving 0.95 accuracy on the final prediction.
We then tested family classification and the full system (classical model + deep neural network).

Table: Full System Evaluation

TP	*FP*	*TN*	*FN*	*Accuracy*	*Precision*	*Recall*	*F1*
7231	49	1451	269	0.96	0.99	0.96	0.97

Adding process memory scanning improved accuracy to 0.96. Family classification accuracy was 73%, with errors only between malware families.

Performance Test: Multi-Stage LEDA with Reasonable Doubt vs Other Trigger Configurations

We tested performance using various real-world tasks on a system with Intel Core i5-4460 @ 3.20 GHz, 16 GB RAM, and a Samsung 850 EVO SSD. Each test was run 12 times, and average results after removing outliers are displayed below.

Table: Test Results for User Activity Benchmarks

Task	*Baseline (ms)*	*S1min (ms)*	*S1min vs Baseline (%)*	*Srd (ms)*	*Srd vs Baseline (%)*
Copy files	11576.53	12512.88	8.09%	11966.51	3.36%
Encode video	2236.29	2417.02	8.08%	2266.37	1.34%
Execute program	1015.19	1035.60	2.01%	1024.43	0.91%
Install program	6655.35	7840.25	17.80%	7240.61	8.79%
Unzip archive	9851.94	10179.72	3.33%	10139.92	2.92%
Zip files	44542.71	45332.42	1.77%	44832.94	0.65%

The *Reasonable Doubt* configuration caused less performance impact than the 1-minute evaluation.
We also tested day-to-day performance using *PCMARK 10*.

Table: Test Results for PCMARK

Task	*Baseline*	*S1min*	*Srd*
Home score	3984	3983	3982
Office score	4412	4201	4369
Storage score	5069	5041	5053
Work score	5160	5140	5155

The *Reasonable Doubt* configuration resulted in a lower performance impact, particularly due to fewer triggered evaluations for clean applications.

4.2 Automated Learning of Detection Triggers

LEDA detects malware by evaluating feature vectors during specific trigger events, marked by “reasonable doubt.” Although effective, the manual selection of triggers is a weakness. To address this, we researched automatic learning of detection models and their optimal triggers, focusing on the behavioural detection model.

4.2.1 Training Methodology

Our training process begins by detonating both malware and clean applications in Curator. For each detonated sample, we compute its normalized event form and denoised form. We then extract the full feature vector from the sample. Since we originally defined "reasonable doubt" based on explainable features, we used all available feature extractors to ensure comprehensive feature extraction.

After obtaining the feature vectors for all samples and labelling them as Malicious or Clean, we compute feature vectors for any known behaviour profiles. These behaviour profiles help the model understand that certain malfunctioning samples are not necessarily malicious.

Given the large number of distinct features in the dataset, we applied feature selection using Logistic Regression with L1 Regularization to reduce the feature set. The selected features are relabeled consecutively, and a new model configuration is generated to extract only the relevant features.

After feature selection, the new feature vectors are computed for each sample. We split the dataset into training and testing subsets and proceed with training the detection model. The parameters of the model are stored in the corresponding configuration file.

Finally, we select the optimal list of trigger points that determine whether a process is classified as malicious or clean. This list aims to minimize computational overhead while maximizing the detection metrics, particularly the F1 score, as it balances both false positives and false negatives.

4.3 Metrics for the Current Version of LEDA

At the end of each chapter, we evaluate LEDA against previous system versions by extracting 1,241 high-level events related to MITRE techniques. We use a Random Forest detection model trained on 3-grams of these techniques, evaluated using the "Reasonable Doubt" method with an extra forced trigger at process termination.

For the detection test, we use data from the LEDA Benchmark Dataset. After selecting the 500 most important features using Logistic Regression with L1 regularization, we train, validate, and test the model on the same dataset.

4.3.1 Experiment: Detection on the LEDA Benchmark Dataset

Since this is the first LEDA version, there are no previous results to compare. Below are the detection results for this configuration.

Table: Detection Metrics for LEDA

Name	*FP*	*FN*	*TP*	*TN*	*Accuracy*	*Precision*	*Recall*	*F1 Score*
LEDA	143	2,857	22,143	7,357	0.9077	0.9936	0.8857	0.9366

LEDA achieved an accuracy of 0.9077 and an F1 score of 0.9366. These scores are slightly lower than results from the dataset used to introduce "Reasonable Doubt," likely due to the broader variety of malware families in the benchmark dataset.

4.3.2 Performance Test: Evolution of Performance Impact

We tested the performance of LEDA using micro-benchmarks and PCMark, comparing the results with a system running no security solution (Baseline).

Table: Performance Impact Measured Using Micro Benchmarks (Lower is Better)

Set Name	*Copy*	*Encode*	*Execute*	*Install*	*Unzip*	*Zip*
Baseline	11,638	2,328	1,125	6,758	9,751	43,639
LEDA	12,946	2,449	1,220	7,293	10,705	47,954

LEDA introduces a slight performance impact, as expected due to the additional code execution. The impact is minimal and acceptable from the user's perspective.
We confirmed these results using the PCMARK benchmark.

Table: Performance Impact Measured Using PCMark (Higher is Better**)**

Set Name	*AppStartup*	*VideoConference*	*WebBrowsing*	*Spreadsheets*	*Writing*	*PhotoEditing*	*Rendering*	*VideoEditing*
Baseline	10,871	7,592	9,558	10,483	7,189	13,847	13,031	6,474
LEDA	10,277	6,864	8,752	9,801	6,415	12,755	12,072	5,793

Although LEDA causes some performance decay, the overall system performance remains high.
Overall, LEDA achieves strong detection metrics with minimal performance degradation, making it a practical and efficient solution for malware detection.

5. Multi-Process Attacks Against Malware Detectors

In the previous chapter, we established that LEDA's detection models, which analyse denoised process representations, effectively train malware detectors. These representations, defined as:

$$\widehat{P} = \{e'_0, e'_1, e'_2, e'_3, e'_4, e'_5 \ldots, e'_n \mid e'_i \text{ is a normalized event}\}$$

capture the essential behaviours within a process. This chapter examines how adversaries can manipulate processes to evade detection, specifically through code injection and inter-process communication techniques, which fragment process behaviour and challenge LEDA's detection capabilities.

5.1 Code Injection for Defence Evasion

Code injection alters the behaviour of a process by injecting parts of it into other processes. This creates fragmented denoised representations for the attacked process P, as well as the victim processes V1 and V2:

$$\widehat{P} = \{e'_0, e'_1, i'_1, i'_2, e'_5 \ldots, e'_n\}$$

$$\widehat{V1} = \{v1'_0, v1'_1, \ldots v1'_i, e'_2, v1'_{i+1}, e'_3, v1'_{i+2} \ldots, v1'_n\}$$

$$\widehat{V2} = \{v2'_0, v2'_1, \ldots v2'_i, e'_4, v2'_{i+1}, v2'_{i+2} \ldots, v2'_n\}$$

This fragmentation makes it difficult for LEDA to detect the full behaviour of process P, reducing detection performance.

We previously addressed this issue by introducing process groups for score-based detectors, which we now adapt to LEDA's machine learning framework. This concept restores LEDA's ability to detect malicious behaviour, even under code injection attacks.

5.2 Group-Aware LEDA

To mitigate the effects of code injection, we introduce process groups in LEDA. The **Entity Manager** manages process groups by categorizing processes into group creators (processes that spawn new groups) or group inheritors (processes that join existing groups when related processes inject code into them). The Entity Manager assigns processes to groups through the following mechanisms:

1. *Process Creation*: When a process PPP is created by a group inheritor, it is added to the parent process's group. If created by a group creator, it starts a new group.
2. *Code Injection*: If a process III injects into process PPP, PPP joins all groups of III, unless III is an allowed injector, typically designated system processes.

By grouping processes, LEDA ensures that fragmented behaviours from multiple processes can still be collectively analysed, enhancing the system's ability to detect malicious actions distributed across different processes.

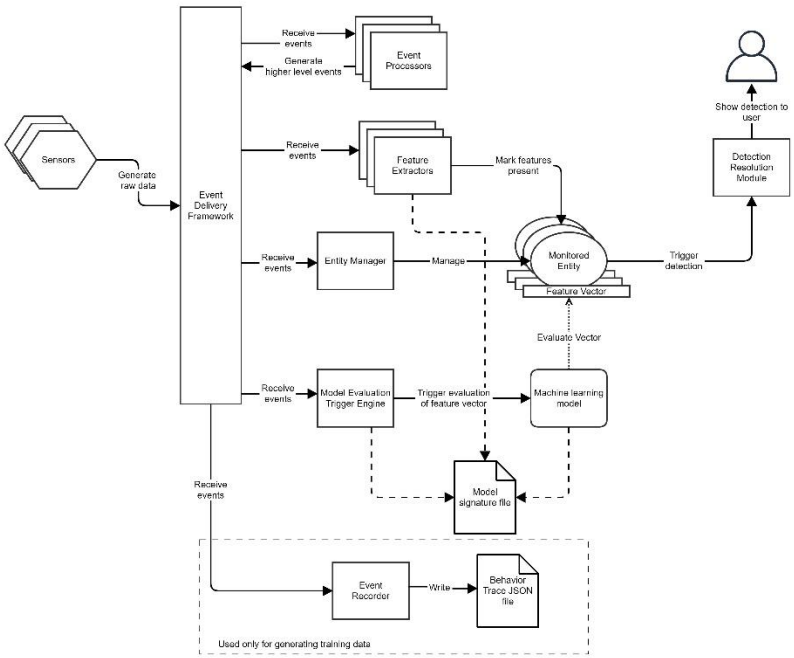


Figure 7: Group Aware LEDA

5.3 Monitoring Techniques for COM, WMI, and ALPC-Based Multi-Process Malware

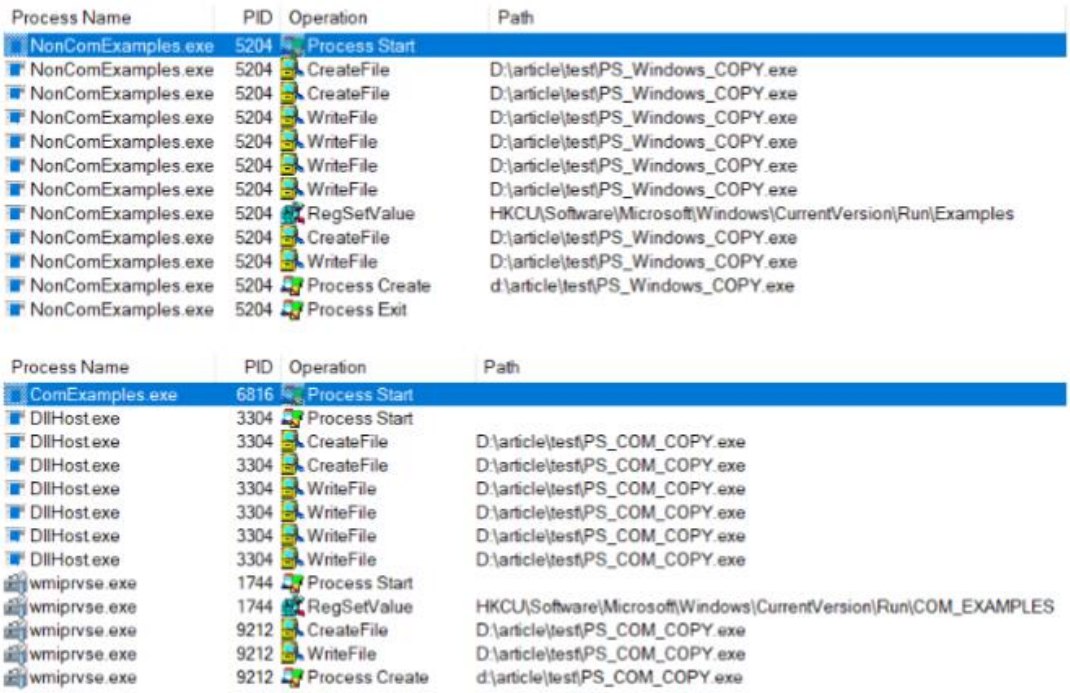
Another evasion technique involves offloading tasks to server processes using inter-process communication mechanisms like COM and WMI. In this case, the representations of the attacker process P and the server processes S1 and S2 become fragmented:

$$\widehat{P} = \{e'_0, e'_1, r'_1, r'_2, e'_5 \ldots, e'_n\}$$

$$\widehat{S1} = \{s1'_0, s1'_1, \ldots s1'_i, e'_2, s1'_{i+1}, e'_3, s1'_{i+2} \ldots, s1'_n\}$$

$$\widehat{S2} = \{s2'_0, s2'_1, \ldots s2'_i, e'_4, s2'_{i+1}, s2'_{i+2} \ldots, s2'_n\}$$

We exemplyfy this easion technique in Figure 8.



Process Name	PID	Operation	Path
NonComExamples.exe	5204	Process Start	
NonComExamples.exe	5204	CreateFile	D:\article\test\PS_Windows_COPY.exe
NonComExamples.exe	5204	CreateFile	D:\article\test\PS_Windows_COPY.exe
NonComExamples.exe	5204	WriteFile	D:\article\test\PS_Windows_COPY.exe
NonComExamples.exe	5204	WriteFile	D:\article\test\PS_Windows_COPY.exe
NonComExamples.exe	5204	WriteFile	D:\article\test\PS_Windows_COPY.exe
NonComExamples.exe	5204	WriteFile	D:\article\test\PS_Windows_COPY.exe
NonComExamples.exe	5204	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Run\Examples
NonComExamples.exe	5204	CreateFile	D:\article\test\PS_Windows_COPY.exe
NonComExamples.exe	5204	WriteFile	D:\article\test\PS_Windows_COPY.exe
NonComExamples.exe	5204	Process Create	d:\article\test\PS_Windows_COPY.exe
NonComExamples.exe	5204	Process Exit	

Process Name	PID	Operation	Path
ComExamples.exe	6816	Process Start	
DllHost.exe	3304	Process Start	
DllHost.exe	3304	CreateFile	D:\article\test\PS_COM_COPY.exe
DllHost.exe	3304	CreateFile	D:\article\test\PS_COM_COPY.exe
DllHost.exe	3304	WriteFile	D:\article\test\PS_COM_COPY.exe
DllHost.exe	3304	WriteFile	D:\article\test\PS_COM_COPY.exe
DllHost.exe	3304	WriteFile	D:\article\test\PS_COM_COPY.exe
DllHost.exe	3304	WriteFile	D:\article\test\PS_COM_COPY.exe
wmiprvse.exe	1744	Process Start	
wmiprvse.exe	1744	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Run\COM_EXAMPLES
wmiprvse.exe	9212	CreateFile	D:\article\test\PS_COM_COPY.exe
wmiprvse.exe	9212	WriteFile	D:\article\test\PS_COM_COPY.exe
wmiprvse.exe	9212	Process Create	d:\article\test\PS_COM_COPY.exe

Figure 8: COM and WMI effects

In such cases, process groups alone are insufficient. Therefore, LEDA's sensors need to be enhanced to trace actions back to their originating process, ensuring that even when inter-process communication techniques are used, the system can maintain its detection accuracy.

We explored various strategies to trace the source of actions performed by a COM server, either by monitoring the client, the server, or by using existing instrumentation through Event Tracing for Windows (ETW). The table below provides a comparison of these methods based on intrusion level, ability to monitor COM and WMI, resistance to attacks without elevated privileges, and implementation difficulty.

Table: Comparison of Monitoring Strategies

Solution	Level of Intrusion	Can Monitor COM	Can Monitor WMI	Known Attacks Without Elevated Privileges	Implementation Difficulty
Client: Object Proxies	Medium - requires setup of proxy	YES	YES	YES - load original library	Easy - Built into COM
Client: Binary Call Interception	High - requires hooks in client	YES	YES	YES - remove hook	Medium - Requires setting up hooks
Server: RPC Proxy	High - requires hooks in server	YES	YES, but with limitations	NO	Very hard - Requires protocol parsing
Server: Per Thread OLE Structures	None	YES	NO	NO	Easy - Requires reading data from TEB
Instrumentation: COM-Perf	None	YES	YES, but with limitations	NO	Medium - Monitoring window algorithm is difficult to implement
Instrumentation: WMI-Activity-Provider	None	NO	YES	NO	Medium - Requires the same monitoring window algorithm as COM-Perf

After reviewing these strategies, binary call interception emerges as a more viable option than object proxies for client-side monitoring, as both methods share similar weaknesses in terms of defence evasion, but object proxies do not support WMI monitoring.

On the server side, parsing per-thread OLE structures offers a practical way to identify the client requesting an operation, avoiding the complexity and limitations associated with using RPC proxies, particularly when monitoring WMI.

Finally, leveraging ETW-based instrumentation proves to be an effective solution. Building a monitoring window using Com-Perf for COM and WMI-Activity-Provider for WMI provides a robust approach that does not require altering the client's address space and is difficult to bypass. This strategy offers a reliable method for tracing actions back to their source while maintaining low levels of intrusion and minimizing implementation complexity.

5.4 Metrics for the Current Version of LEDA

In this chapter, we assess the latest version of LEDA, which incorporates process groups and enhanced sensors to address multi-process malware. LEDA extracts 1,241 high-level events related to MITRE techniques, and the detection model used is a Random Forest trained on 3-grams of these techniques. The evaluation follows the "Reasonable Doubt" standard with an additional forced trigger on process termination.
For all detection tests, we use data from the LEDA Benchmark Dataset, selecting the top 500 features using Logistic Regression with L1 regularisation. The results highlight the improvements from incorporating process groups, COM, WMI, and ALPC monitoring.

5.4.1 Experiment: Detection on the LEDA Benchmark Dataset

The detection results are presented in the table below. Introducing process groups increased the F1 score from 0.93 to 0.94. Further improvements in COM, WMI, and ALPC monitoring raised it to 0.95.

Table: Detection Metrics for LEDA

Name	*FP*	*FN*	*TP*	*TN*	*Accuracy*	*Precision*	*Recall*	*F1 Score*
LEDA	143	2,857	22,143	7,357	0.9077	0.9936	0.8857	0.9366
GROUPS	185	2,394	22,606	7,315	0.9206	0.9919	0.9042	0.9460
GROUPS+COM	238	2,125	22,875	7,262	0.9273	0.9897	0.9150	0.9509

While adding groups and COM monitoring led to increases in both false positives and false negatives, the overall F1 score improvement demonstrates the effectiveness of these methods.

5.4.2 Performance Test: Evolution of Performance Impact

The performance impact of these changes was measured using micro benchmarks. As shown in the table below, while there is a slight performance decrease with the introduction of groups and COM monitoring, the system still performs within acceptable limits.

Table: Performance Impact Measured Using Micro Benchmarks (Lower is Better**)**

Set Name	*Copy*	*Encode*	*Execute*	*Install*	*Unzip*	*Zip*
BASELINE	11,638	2,328	1,125	6,758	9,751	43,639
LEDA	12,946	2,449	1,220	7,293	10,705	47,954
GROUPS	13,336	2,534	1,268	7,574	11,145	50,203
GROUPS+COM	13,355	2,527	1,256	7,587	11,118	49,543

We confirmed these findings using the PC Mark benchmark, which registered some performance degradation caused by the added security measures. However, the overall system performance remains acceptable.

Table: Performance Impact Measured Using PCMARK (Higher is Better)

Set Name	*AppStartup*	*VideoConference*	*WebBrowsing*	*Spreadsheets*	*Writing*	*PhotoEditing*	*Rendering*	*VideoEditing*
BASELINE	10,871	7,592	9,558	10,483	7,189	13,847	13,031	6,474
LEDA	10,277	6,864	8,752	9,801	6,415	12,755	12,072	5,793
GROUPS	9,773	6,567	8,456	9,461	6,222	12,257	11,684	5,576
GROUPS+COM	9,337	6,350	8,056	8,994	5,959	11,702	11,132	5,347

Overall, the inclusion of process groups and enhanced monitoring strategies in LEDA demonstrates improved detection capabilities, with only minor performance trade-offs.

6. Techniques to Detect and Neutralize Multi-Component Malware

Malware detection often analyses process behaviour to spot malicious activity. LEDA translates this behaviour into event streams and applies machine learning for detection.

In previous chapters, we enhanced LEDA to handle code injection and the use of COM, WMI, and RPC. However, advanced attackers can still evade detection, as seen in the SolarWinds attack, where the Solorigate backdoor was embedded in legitimate software, making it difficult to trace.

To address this, we propose a persistent entity map that tracks the origin of malicious executables, even across system reboots. This map links system components used by attackers, improving visibility and detection accuracy by making it harder to hide the initial infection point.

6.1 Persistent Entity Maps for Malware Detection

6.1.1 Introducing Entity Maps

An **entity map** is a dynamic graph-based model that captures relationships between system components like processes, files, and registry keys. It is built in real-time by monitoring these interactions to trace the origin of malicious activity and determine how an attacker gained control over the system.

The entity map tracks code execution across processes. Each process runs code from executable files and libraries, but interpreters like *PowerShell* or *Microsoft Word* execute commands or scripts. For these interpreters, the entity map focuses on the scripts or commands executed, not just the executable file.

Processes in Windows are created in various ways:

- Direct methods: System calls like *NtCreateUserProcess*, service calls via *svchost.exe*, or out-of-process COM calls using *ShellBrowserWindow* or the *WIN32_Process* WMI class.
- Indirect methods: These include process creation through services, tasks scheduled via Task Scheduler, registry keys set to start processes, or WMI event notifications that trigger process launches.

Additionally, processes can manipulate others through code injection or hijacking legitimate modules, exploiting Windows' library load order or modifying registry keys. These interactions form the backbone of the entity map, which models system state and traces potential compromises.

6.1.2 Extension to LEDA to Accommodate Entity Maps

We extended LEDA to support entity maps by adding new components to its architecture(Figure 9). The **Entity Map Manager** constructs and analyzes the relationships between processes, files, and registry keys, while the **Real-Time Entity Map Manager** maintains these maps by monitoring system activity. A **Historical Entity Map Query Engine** retrospectively examines if past malware activity was previously recorded, using persistent data stored in a database.

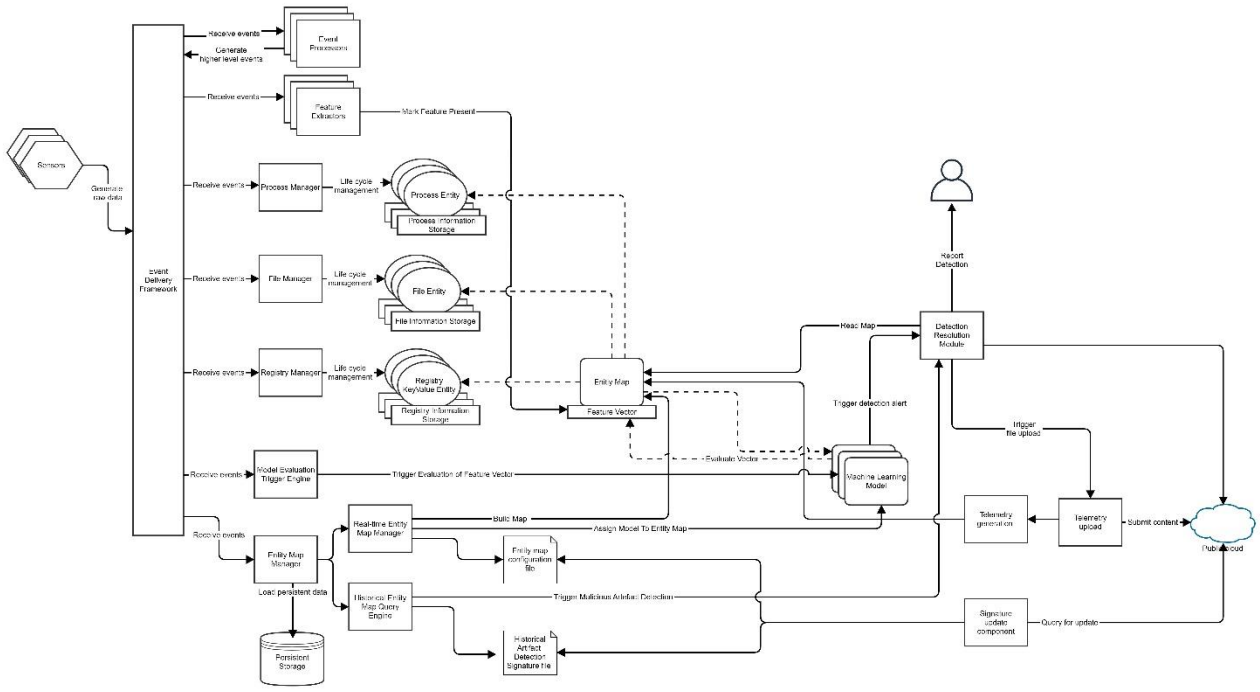


Figure 9: LEDA with support for Entity Maps

Supporting components like **Process**, **Registry**, and **File Managers** handle process, registry, and file events, respectively, while abstracting information for the system. A **Signature Update Component** ensures that detection signatures are kept current by updating the entity map configuration and historical signatures.

The extended LEDA system also features telemetry submission and a detection resolution module to neutralize detected malware and present information to the user.

6.1.3 Building the Entity Map

Entity maps are maintained using algorithms that track entity interactions. For instance, when a file or registry key is accessed, the system checks if an associated entity exists. If not, a new

entity is created, possibly reconstructed from persistent data. Similar steps are followed during process creation and code injection events, ensuring the accurate reflection of system states.

Entity maps are disrupted after system reboots, but by storing relevant data, they can be reconstructed. The system ensures that entity relationships, such as file modifications or process spawns, are preserved even after a restart.

6.1.4 Modified LEDA Algorithms for Entity Maps

LEDA's algorithms are adapted to work with entity maps. Each event is assigned to an entity map snapshot, and feature extraction is performed on all relevant entity maps. Model evaluations are triggered when necessary, based on these snapshots. Events generated before a process is added to the entity map are not considered.

The architecture ensures that entity maps can track complex interactions between system components, providing a more comprehensive detection system for malware.

6.1.5 Dealing with Persistence

To ensure the entity map persists beyond reboots, we propose storing the maps tied to critical events like file writes or registry key settings. However, this persistent storage requires careful management to avoid data overload. Storage policies are implemented with rules that dictate when data should be deleted or retained, depending on usage frequency and priority.

6.1.6 Entity Description Language

Entity maps are managed through a JSON-based structure. The formalized language includes descriptions of entities (processes, files, registry keys), relationships between them, and actions triggered when specific conditions are met. This setup allows for flexible handling of complex system interactions, ensuring efficient detection and response to malware activities.

6.1.7 Signature Rules and Matching

Entities contain features that can be matched through predicates like *Contains* or *Equals*. When a relationship between entities is recognized, the system checks if the entity matches predefined signatures. Signature matching is done by monitoring entity creation and relationships, with actions executed when a match is found.

The matching process works as a subgraph problem, where the entity map is the larger graph, and the signature is the subgraph we are trying to detect. Partial matches are updated in real-time until a full match is confirmed, triggering the corresponding actions.

These algorithms allow real-time graph signature matching, ensuring entity maps stay current and reflect the latest system events. This approach enhances LEDA's ability to detect and respond to sophisticated malware attacks.

6.1.8 Examples: Entity Map Management

In the example scenario, we illustrate how malware can abuse OneDrive for persistence across reboots. The attack follows these steps:

1. A user navigates to an attacker-controlled website using *Internet Explorer*.
2. The website exploits a vulnerability to trigger remote code execution in *Internet Explorer*.
3. *Internet Explorer* drops a malicious DLL (*OneDriveTelemetryExperimental.dll*) into the OneDrive directory.
4. The user restarts the computer.
5. After the reboot, a legitimate instance of *OneDrive* starts and loads the malicious DLL.
6. *OneDrive* begins deleting volume shadow copies and encrypting user files.

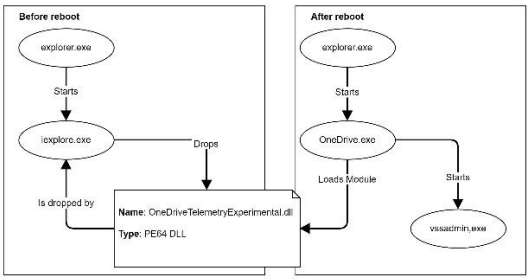


Figure 10: Attack Flow

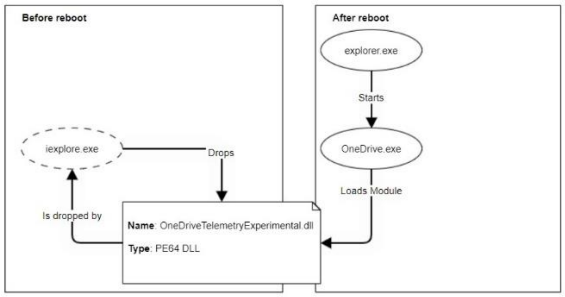


Figure 11: Entity Map for Attack

This example shows how the initial infection logic is separated from the ransomware activity by a reboot. LEDA uses entity maps to track the relationship between the malicious file dropped by *Internet Explorer* and the subsequent activities of *OneDrive* after the reboot. The system persists information about files dropped by *iexplore.exe* and recognizes external executables. Once *OneDrive* starts and loads the malicious DLL, LEDA ties the malware's actions back to the external code, despite the system reboot.

6.2 Neutralizing Malicious Applications After Execution

Neutralizing malicious applications after execution is a complex task that requires a comprehensive approach to containment and mitigation. Once LEDA identifies a threat, the system has already been compromised, necessitating swift action to prevent further damage.

LEDA's role goes beyond detection. It must restore the system to its original, secure state. This involves repairing or reinstalling compromised components, removing malicious code, and ensuring no lingering threats remain. Challenges like ransomware, which encrypts critical files, demand that LEDA has access to uncorrupted backups. Similarly, file infectors, which embed themselves in legitimate files, require a careful recovery process to avoid damaging valid files.

Post-incident, it's essential for the security team to document the threat response thoroughly. This includes detailing the steps taken to neutralize the threat, evaluating their success, and learning from the incident to strengthen future defences. Understanding how the malware

was executed, its damage, and the exploited vulnerabilities is key to developing better preventive measures.

For LEDA to support this, it must log every application's behaviour and system changes. These logs are essential for both real-time threat response and post-incident analysis. They help reconstruct the attack, assess the damage, and identify areas for improvement. Additionally, analyzing historical data helps detect patterns of ongoing or potential future threats.

To address these challenges, we have enhanced LEDA with a **System Disinfect** module, designed to neutralize malicious activities and restore system integrity after malware detection. Central to this is the **Process Activity Logger**, which tracks critical system actions—such as file, registry, and WMI modifications—in an **Activity Journal**. This log is carefully managed to ensure it captures only the most relevant events, preventing malicious processes from overwhelming the system.

A **Backup Manager** complements the system by handling the creation, modification, and deletion of backups for key files. These backups are stored in a protected **Backup Storage** with a fixed capacity and managed through a FIFO policy, ensuring that the oldest backups are overwritten first when needed.

When LEDA detects malicious activity through an entity map, the **System Disinfect** module identifies and terminates all related processes. It then rolls back their actions, restoring files from the Backup Storage if necessary, to ensure the system is fully cleansed.

LEDA's entity map also tracks the entire attack chain, preserving this information even across reboots. This ensures continuity in tracking and neutralizing the threat, even if the system restarts. By compiling this data, LEDA can generate a comprehensive report that provides a detailed analysis of the attack, helping users understand the threat and improve future security measures.

6.3 Metrics for the Current Version of LEDA

At the conclusion of this final contributions chapter, we compare all versions of LEDA, focusing on 1,241 high-level events associated with MITRE techniques. For detection, we use a Random Forest model trained on 3-grams of MITRE techniques, evaluated with the 'Reasonable Doubt' standard and a forced trigger on process termination.

Detection testing was performed using the LEDA Benchmark Dataset, with the top 500 features selected via Logistic Regression with L1 regularization. In this chapter, process groups were replaced with entity maps, and we introduced a system disinfect module to neutralize malware post-detection.

6.3.1 Detection Experiment on the LEDA Benchmark Dataset

The detection results, shown below, demonstrate that entity maps outperform groups, as evidenced by the improvement in the F1 score. Each successive LEDA version results in fewer false negatives, albeit with a slight increase in false positives. However, the rise in false

positives is outweighed by the consistent improvement in the F1 score, indicating that the trade-off is acceptable.

Table: Detection Metrics for LEDA

Name	*FP*	*FN*	*TP*	*TN*	*Accuracy*	*Precision*	*Recall*	*F1 Score*
LEDA	143	2,857	22,143	7,357	0.9077	0.9936	0.8857	0.9366
GROUPS	185	2,394	22,606	7,315	0.9206	0.9919	0.9042	0.9460
GROUPS+COM	238	2,125	22,875	7,262	0.9273	0.9897	0.9150	0.9509
ENTITY_MAP+COM	274	1,984	23,016	7,226	0.9305	0.9882	0.9206	0.9532

6.3.2 Performance Test: Evolution of Performance Impact

We conducted a final performance test on all versions of LEDA. The results reveal that the performance impact of groups and entity maps is comparable, with differences falling within the test error margin.

Table: Performance Impact Measured Using Micro Benchmarks (Lower is Better)

Set Name	*Copy*	*Encode*	*Execute*	*Install*	*Unzip*	*Zip*
BASELINE	11,638	2,328	1,125	6,758	9,751	43,639
LEDA	12,946	2,449	1,220	7,293	10,705	47,954
GROUPS	13,336	2,534	1,268	7,574	11,145	50,203
GROUPS+COM	13,355	2,527	1,256	7,587	11,118	49,543
ENTITY_MAP+COM	13,181	2,476	1,232	7,405	10,829	48,591

The PC Mark benchmark confirms a similar trend, with each version of LEDA introducing a slight performance decay. However, the overall system performance remains high, even when incorporating entity maps and the disinfect module.

Table: Performance Impact Measured Using PC Mark (Higher is Better)

Set Name	*AppStartup*	*VideoConference*	*WebBrowsing*	*Spreadsheets*	*Writing*	*PhotoEditing*	*Rendering*	*VideoEditing*
BASELINE	10,871	7,592	9,558	10,483	7,189	13,847	13,031	6,474
LEDA	10,277	6,864	8,752	9,801	6,415	12,755	12,072	5,793
GROUPS	9,773	6,567	8,456	9,461	6,222	12,257	11,684	5,576
GROUPS+COM	9,337	6,350	8,056	8,994	5,959	11,702	11,132	5,347
ENTITY_MAP+COM	9,226	6,272	7,973	8,843	5,887	11,512	10,959	5,268

The introduction of entity maps into LEDA provides a significant improvement in detection performance, as demonstrated by the consistently higher F1 scores. While there is a slight increase in false positives, this is balanced by the reduction in false negatives. The performance impact is minimal, with overall system performance remaining high, even in the most advanced configurations.

7. Conclusions

Cybersecurity has a history spanning over 50 years, yet protecting users and infrastructures from threat actors remains a critical challenge. This thesis aims to strengthen defenses that prevent attackers from disrupting essential aspects of daily life.

Chapter 1 introduces the context, objectives, and structure of the thesis, highlighting the problem of detecting malicious applications during execution but before they cause irreparable harm.

Chapter 2 reviews the theoretical background and state of the art in behavioural detection. We examined machine learning as a solution for malware detection, considering practical

aspects like data collection, attacks against detection models, and balancing security with system usability.

Chapter 3 describes the Layered Event-based Detection Architecture (LEDA). Here, we introduced a model where processes are represented as an ordered set of events generated by sensors:

$$P = \{e_0, e_1, \ldots, e_n \mid e_i \text{ is an event captured by LEDA for the process } \}$$

By normalizing process events, we defined a denoised process:

$$\hat{P} = P' - \text{Noise}$$

Chapter 4 focuses on detecting malware, where we extended a detection model to evaluate features during execution. We introduced the concept of "reasonable doubt" to identify suspicious actions as trigger points for model evaluation.

In Chapter 5, we addressed multi-process attacks, such as code injection and technologies like COM, WMI, and ALPC, which challenge single-process detection models. We introduced the concept of process groups and detection based on events generated by the group:

$$G = \{e \mid e \text{ is generated by a process in the group}\}$$

We improved sensors to detect malware using COM, WMI, and ALPC.
Chapter 6 generalizes LEDA for multi-component malware, which splits execution across boot cycles. We introduced **entity maps** as a generalization of process groups to track malware across multiple components:

$$EM = \{e \mid e \text{ is generated by a process in the entity map}\}$$

This chapter also discusses how entity maps help revert malicious actions and find the entire kill chain of an attack.

Throughout its evolution, LEDA showed consistent improvements in detection performance, as measured by the F1 score. Starting with 0.9366 in its single-process version, it improved to 0.9460 with process groups, then to 0.9509 with sensor enhancements, and finally to 0.9532 with entity maps. Despite the increased performance impact, the system remained efficient according to benchmarks.

Cybersecurity is a continuous race between attackers and defenders. We hope the concepts presented in this thesis contribute to raising the cost for attackers and lowering it for defenders through innovations shared with the academic community.

7.2 Contributions of the Thesis

The contributions of this thesis can be decomposed into two categories: theoretical contributions and practical ones.
From a theoretical perspective, we modelled process behaviour and cyber attacks and designed algorithms that help reproduce and detect malware. The main theoretical contributions are:

- **TC1**: Definition of process behaviour as a series of events, with this definition extended to account for code injections (using process groups) and multi-component malware (using entity maps).
- **TC2**: Algorithm to automatically determine noise in event representations of processes.
- **TC3**: Identification of problems affecting datasets for malware behaviour detection, termed the Oracle problem and Malfunctioning sample problem, with algorithms developed to address them.
- **TC4**: Proposal of a new perspective on cyber attacks as a set of adversary objectives.
- **TC5**: New strategy for evaluating process behaviour efficiently, based on a set of trigger points defined using expert knowledge, named "reasonable doubt."
- **TC6**: Methodology for automatically learning optimal model evaluation trigger points.
- **TC7**: Measurement of decay of detection model metrics over time.
- **TC8**: Model for multi-component applications using "entity maps" built from processes, files, and registry keys.
- **TC9**: Algorithms to store "entity maps" across multiple boot cycles.

From a practical perspective, we implemented several systems that can safely run malware, reproduce cyber-attacks, detect malicious applications, and neutralize them. The main practical contributions are:

- **PC1**: Creation of a malware detection architecture named LEDA.
- **PC2**: Development of a new malware detonation environment (sandbox) called Curator.
- **PC3**: Establishment of an adversary emulation framework named SpecRep.
- **PC4**: Alterations to LEDA to combine behavioural and static features extracted from memory for detecting malicious applications with minimal impact on system performance.
- **PC5**: Extension of LEDA to detect multi-process malware by organising processes into groups.
- **PC6**: Alterations to LEDA sensors to account for potential detection evasion methods based on COM, WMI and ALPC.
- **PC7**: Modification of LEDA to use "entity maps" to address multi-component malware.
- **PC8**: Implementation of a method to neutralise malicious processes detected by LEDA.

7.3 Correlation of Contributions with Objectives

We present the correlation between our research objectives, contributions, the chapters they were presented in, and the papers in which they were presented to the community below.

Table: Correlation between objectives and contributions of the thesis

Objective	***Contribution***	***Chapter***	***Published Work***
1.1	TC1	2, 3, 6	P1, A3
1.2	PC1, PC2	3	A8
1.3	TC2, TC3	3	A8
1.4	TC4, TC7	3, 6	A1, P3

1.5	PC3	3	A1
2.1	PC4	3, 4	A7, P1, A3
2.2	PC4	4	A7
2.3	PC4	4	A7
2.4	TC5, TC6	4	P1, A7
2.5	TC7	4	A3
3.1	PC5	5	A10, P6
3.2	PC5	5	A10, P6
3.3	PC6	5	A1
3.4	PC6	5	A1
4.1	TC8, TC9	6	P6, P3
4.2	PC7	6	P6, P3, P4
4.3	PC7	6	P3
4.4	PC8	6	P6

8. List of Publications

8.1 US Patents

1. **Radu-Marian Portase**, Botond Fülöp, and Gheorghe Florin Hăjmășan. *Antimalware Systems and Methods Using Optimal Triggering of Artificial Intelligence Modules.* US Patent Application No. US-20240296222-A1, 2024.
2. Gheorghe Florin Hăjmășan, **Radu-Marian Portase**, and Alexandra Hăjmășan. *Entity Profile Signatures for Improving Malware Detection.* US Patent Application No. 18/601,012, Filed 03/11/2024.
3. **Radu-Marian Portase**, Gheorghe Florin Hăjmășan, and Alexandra Hăjmășan. *Systems and Methods for Countering Persistent Malware C1.* US Patent Application No. 18/461,251, Filed 09/05/2023.
4. **Radu-Marian Portase**, Gheorghe Florin Hăjmășan, and Alexandra Hăjmășan. *Systems and Methods for Countering Persistent Malware.* US Patent Application No. 18/461,134, Filed 09/05/2023.
5. Gheorghe Florin Hăjmășan, Alexandra Mondoc, and **Radu-Marian Portase**. *Dynamic Reputation Indicator for Optimizing Computer Security Operations.* US Patent US10237293B2, 2019. URL: https://patents.google.com/patent/US10237293B2
6. Gheorghe Florin Hăjmășan and **Radu-Marian Portase**. *Systems and Methods for Tracking Malicious Behavior Across Multiple Software Entities.* US Patent No. US10089465B2, 2018. URL: https://patents.google.com/patent/US9323928B2

8.2 Publications in ISI Indexed Journals

1. **Radu-Marian Portase**, Adrian Colesa, and Gheorghe Sebestyen. *SpecRep: Adversary Emulation Based on Attack Objective Specification in Heterogeneous Infrastructures.* Sensors 24, no. 17 (2024): 5601. https://doi.org/10.3390/s24175601. (Q2 Journal)
2. **Radu-Marian Portase**, Andrei Marius Muntea, Andrei Mermeze, Adrian Colesa, and Gheorghe Sebestyen. *Detection Strategies for COM, WMI, and ALPC-Based Multi-Process*

Malware. Sensors 24, no. 16 (2024): 5118. https://doi.org/10.3390/s24165118. (Q2 Journal)

3. **Radu-Marian Portase**, Raluca Laura Portase, Adrian Colesa, and Gheorghe Sebestyen. *LEDA: Layered Event-Based Malware Detection Architecture. Sensors 2024, 24, 6393. https://doi.org/10.3390/s24196393*

8.3 Publications at International Conferences

4. Marian Sandor, **Radu-Marian Portase**, and Adrian Colesa. *Enhancing ML Model Resilience to Time-Evolving Data in Malware Detection Systems with Adversarial Learning.* Accepted at 2024 IEEE 20th International Conference on Intelligent Computer Communication and Processing (ICCP).
5. Andrei-Marius Muntea, **Radu-Marian Portase**, and Gheorghe Sebestyen. *Monitoring of RPC Messages with ALPC-Level API Hooking.* 2024 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), 2024, pp. 1-6. IEEE.
6. Marian Sandor, **Radu-Marian Portase**, and Adrian Colesa. *Ember Feature Dataset Analysis for Malware Detection.* 2023 IEEE 19th International Conference on Intelligent Computer Communication and Processing (ICCP), 2023, pp. 203-210.
7. **Radu-Marian Portase**, Adrian Colesa, and Gheorghe Sebestyen. *Efficient Multi-Stage Detection of Malicious Windows Executable During Execution.* 2023 IEEE 19th International Conference on Intelligent Computer Communication and Processing (ICCP), 2023, pp. 211-218. https://api.semanticscholar.org/CorpusID:267203008
8. **Radu-Marian Portase** and Adrian Colesa. *Curator - A System for Creating Data Sets for Behavioural Malware Detection.* 2021 20th International Symposium on Parallel and Distributed Computing (ISPDC), 2021, pp. 57-64. https://api.semanticscholar.org/CorpusID:237328775
9. Gheorghe Florin Hăjmăşan, Alexandra Mondoc, **Radu-Marian Porta**se, and Octavian Creţ. *Performance Improvements in Behaviour-Based Malware Detection Solutions.* IFIP International Conference on ICT Systems Security and Privacy Protection, 2018, pp. 370-384.
10. Gheorghe Florin Hăjmăşan, Alexandra Mondoc, **Radu-Marian Portase**, and Octavian Creţ. *Evasive Malware Detection Using Groups of Processes.* IFIP International Conference on ICT Systems Security and Privacy Protection, 2017, pp. 32-45.

References

[1] Nwokedi Idika and Aditya Mathur. “A survey of malware detection techniques”. In: Purdue University (Mar. 2007)

[2] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick D¨ussel, and Pavel Laskov. “Learning and Classification of Malware Behavior”. In: Detection of Intrusions and Malware, and Vulnerability Assessment. Ed. by Diego Zamboni. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 108–125.

[4] Tobias Wuchner, Martın Ochoa, and Alexander Pretschner. “Malware Detection with Quantitative Data Flow Graphs”. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. ASIA CCS ’14. Kyoto, Japan: ACM, 2014, pp. 271–282. isbn: 978-1-4503-2800-5. doi: 10.1145/2590296.2590319. url: http://doi.acm.org/10.1145/2590296.2590319.

[4] Gheorghe Florin Hăjmășan, Alexandra Mondoc, **Radu-Marian Portase**, and Octavian Creț. *Evasive Malware Detection Using Groups of Processes.* IFIP International Conference on ICT Systems Security and Privacy Protection, 2017, pp. 32-45.

[5] Schultz, M.G., Eskin, E., Zadok, F., & Stolfo, S.J. (2001). Data mining methods for detection of new malicious executables. Proceedings 2001 IEEE Symposium on Security and Privacy, pp. 38-49. doi: 10.1109/SECPRI.2001.924286.

[6] Ucci, D., Aniello, L., & Baldoni, R. (2017). Survey on the Usage of Machine Learning Techniques for Malware Analysis. Computers and Security, 81, 123-136.

[7] Wüchner, T., Ochoa, M., & Pretschner, A. (2014). Malware Detection with Quantitative Data Flow Graphs. Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, pp. 271-282. doi: 10.1145/2590296.2590319.

[8] Dahl, G.E., Stokes, J.W., Deng, L., & Yu, D. (2013). Large-scale malware classification using random projections and neural networks. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 3422-3426.

[9] Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2017). Empowering convolutional networks for malware classification and analysis. 2017 International Joint Conference on Neural Networks (IJCNN), pp. 3838-3845. doi: 10.1109/IJCNN.2017.7966340.

[10] Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., & Yagi, T. (2016). Malware Detection with Deep Neural Network Using Process Behavior. 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), vol. 2, pp. 577-582.

[11] Huang, W. & Stokes, J.W. (2016). MtNet: A Multi-Task Neural Network for Dynamic Malware Classification. DIMVA.

[12] Wachdorf, D., & Lundgren, S. (2017). Cybersecurity systems and techniques. US Patent Application US20170329968A1.

[13] Martin, D.T., & Diehl, D.F. (2013). Event Model for Correlating System Component States. US Patent Application US20150101044A1.

[14] Microsoft. (2017). Antivirus evolved. Retrieved from https://cloudblogs.microsoft.com/microsoftsecure/2017/05/08/antivirus-evolved/.

[15] Microsoft. (2017). Windows Defender Antivirus cloud protection service: Advanced real-time defense against never-before-seen malware. Retrieved from https://cloudblogs.microsoft.com/ microsoftsecure/2017/07/18/windows-defender-antivirus-cloud-protection-service-advanced-real-time-defense-against-never-before-seen-malware/.

[16] Mohaisen, A., Alrawi, O., & Mohaisen, M. (2015). AMAL: High-fidelity, behavior-based automated malware analysis and classification. Computers and Security, 52, 251-266.

[17] Maniriho, P., Mahmood, A.N., & Chowdhury, M.J.M. (2023). API-MalDetect: Automated malware detection framework for windows based on API calls and deep learning techniques. Journal of Network and Computer Applications, 218, 103704.

[18] Ronen, R. et al. (2018). Microsoft Malware Classification Challenge. https://arxiv.org/abs/1802.10135.

[19] Anderson, H., & Roth, P. (2018). EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. https://arxiv.org/abs/1804.04637.

[20] Harang, R., & Rudd, E.M. (2020). SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection. https://arxiv.org/abs/2012.07634.

[21] Willems, C., Holz, T., & Freiling, F. (2007). Toward Automated Dynamic Malware Analysis Using CWSandbox. IEEE Security and Privacy, 5, 32-39. doi: 10.1109/MSP.2007.45.

[22] Bayer, U., Mandl, T., & Nentwich, F. (2009). ANUBIS: ANalyzing Unknown Binaries. Virus Bulletin Conference 2009.

[23] Cuckoo Sandbox Team. (2024). Cuckoo Sandbox: Automated Malware Analysis System. https://cuckoosandbox.org.

[24] Yokoyama, A., et al. (2016). SandPrint: Fingerprinting Malware Sandboxes. RAID Conference, pp. 165-187.

[25] Cohen, F. (1997). Information system attacks: A preliminary classification scheme. Computers & Security, 16(1), 29-46.

[26] Cohen, F. (1999). Simulating cyber attacks, defences, and consequences. Computers & Security, 18(6), 479-518.

[27] Park, J.S., et al. (2001). SECUSIM: A Tool for the Cyber-Attack Simulation. International Conference on Information, Communications and Signal Processing.

[28] Kotenko, I., & Man'kov, E. (2003). Experiments with Simulation of Attacks against Computer Networks. Computer Network Security, pp. 183-194.

[29] Ajmal, A.B., et al. (2023). Toward Effective Evaluation of Cyber Defense: Threat Based Adversary Emulation Approach. IEEE Access, 11, 70443-70458.

[30] Oh, S.H., et al. (2023). Applying Reinforcement Learning for Enhanced Cybersecurity against Adversarial Simulation. Sensors, 23(6), 3000.

[31] MITRE. (2024). CALDERA - Automated Adversary Emulation Platform. https://github.com/mitre/caldera.

[32] MITRE. (2024). CASCADE - Server Framework for Emulating Adversaries. https://github.com/mitre/cascade-server.

[33] Red Canary. (2024). Atomic Red Team - Small Tests for Adversary Emulation. https://github.com/redcanaryco/atomic-red-team.

[34] Nextron Systems. (2024). APTSimulator - A Windows Attack Simulation Tool. https://github.com/NextronSystems/APTSimulator.

[35] Chuvakin, A. (2024). Breach and Attack Simulation (BAS) Tools Reviews and Ratings. https://www.gartner.com/reviews/market/breach-and-attack-simulation-bas-tools.

[36] Ji, Y., et al. (2014). A Multiprocess Mechanism of Evading Behavior-Based Bot Detection Approaches. ISPEC, pp. 75-89. doi: 10.1007/978-3-319-06320-1_7.

[37] Ispoglou, K.K., & Payer, M. (2016). malWASH: Washing Malware to Evade Dynamic Analysis. WOOT.

[38] Ma, W., et al. (2012). Shadow attacks: Automatically evading system-call-behaviour based malware detection. Journal in Computer Virology, 8(1), 1-13. doi: 10.1007/s11416-011-0157-5.

[39] Thompson, D., et al. (1997). Distributed Component Object Model (DCOM). Department of Software Development, Monash University.

[40] Eddon, G., & Platt, D.S. (1999). Inside COM+ Base Services. Microsoft Press.

[41] Forshaw, J. (2017). COM in 60 seconds!. InfoCon.

[42] Koadic. (2024). Koadic GitHub Repository. https://github.com/zerosum0x0/koadic.

[43] Graeber, M. (2015). Abusing Windows Management Instrumentation (WMI) to Build a Persistent, Asynchronous, and Fileless Backdoor. Black Hat USA.

[43] Dizon, J., et al. (2017). *Understanding WMI Malware*. Trend Micro.

[44] SANS. (2019). *Investigating WMI Attacks*. https://www.sans.org/blog/investigating-wmi-attacks/.

[45] Ionescu, I.A. (2021). *RPC Call Interception*. US Patent Application US20150163109A1.