A Detection Method for COM-Based Malicious Code Using Dynamic Behavior Analysis

Jing Lu
B21031417
Nanjing University of Posts and Telecommunications
Nanjing, China

***Abstract*—To address the widespread and highly latent nature of malicious code utilizing the Component Object Model (COM) interface, which makes it difficult to detect, this paper proposes a discovery method based on dynamic behavior analysis. The approach begins by executing samples in the Cuckoo sandbox to capture system audit logs, from which a behavior provenance graph is constructed. A reachable subgraph directly associated with the sample's behavior is then extracted. This graph structure is converted into an abstract behavior sequence according to predefined class information such as "process," "file," or "operation," including key operations like ¡RpcCall¿ which represents COM call behavior. Utilizing a Bag-of-Words (BoW) model combined with Uni-gram, Bi-gram, and Tri-gram methods, a Random Forest classifier is employed to identify malicious behavior. The results of a 5-fold cross-validation experiment on a dataset of 18,000 samples (9,000 malicious and 9,000 benign) show that the proposed model achieves an average accuracy and F1-score of 0.9421. These experimental results validate the effectiveness and feasibility of the method for analyzing and detecting malicious activities that leverage the COM interface, offering a valuable approach and reference for research in this field.**

***Index Terms*—COM interface, Malware Detection, Dynamic Behavior Analysis, Behavior Provenance Graph, Bag-of-Words, N-gram, Random Forest.**

I. INTRODUCTION

The Component Object Model (COM) is a software component technology by Microsoft for its Windows OS, valued for cross-language operation and reusability in technologies like ActiveX and OLE. However, its flexibility, especially the out-of-process execution model, is often exploited by malware for stealthy execution, persistence, and privilege escalation, posing a significant security threat.

Traditional detection methods struggle with the complexity and stealth of COM-based attacks, which are often lost in a sea of legitimate system activities. This research addresses this gap by proposing a novel detection framework that leverages dynamic analysis and machine learning to distinguish malicious COM usage from benign behavior.

The main contributions of this paper are as follows:

- We design a systematic workflow to capture and represent complex COM-related behaviors using system-level provenance graphs.
- We propose a feature engineering pipeline that converts structured graph data into abstract behavioral sequences and then into effective numerical feature vectors using a hybrid Bag-of-Words and N-gram approach.
- We demonstrate that a Random Forest model trained on these features achieves high accuracy and a strong F1-score, validating the effectiveness of our approach on a large, balanced dataset.

II. BACKGROUND

Research into COM-based threats has been ongoing. Internationally, security researchers have noted the abuse of COM and its related technology, Windows Management Instrumentation (WMI). The Empire framework (2015), a fileless backdoor demonstrated by Matt Graeber (2015), and the Koadic post-exploitation tool (2017) are prominent examples of COM/WMI exploitation. The WikiLeaks Vault 7 disclosures further confirmed that APT groups leverage these techniques. On the defensive side, early work on COM interception frameworks and recent use of Event Tracing for Windows (ETW) have provided monitoring capabilities.

In China, research has also begun to focus on complex, multi-process attacks that involve COM cross-process calls. However, a systemic methodology for modeling and accurately identifying malicious COM calls in a high-noise environment remains a significant challenge. Our work builds upon these foundations, aiming to create a more robust and automated analysis system.

III. PROPOSED METHODOLOGY

Our proposed detection framework consists of four main stages: (1) Dynamic Data Acquisition, (2) Behavior Provenance Graph Construction, (3) Feature Engineering, and (4) Classification. The overall workflow is depicted in Fig. 1.

A. Data Acquisition and Graph Construction

We first collected a dataset of 9,000 malicious and 9,000 benign samples. Each sample was executed in an isolated Cuckoo Sandbox environment running Windows 10 to capture its dynamic behavior. During execution, detailed system audit logs recording events like process creation, file I/O, network connections, and API calls were collected.

From these raw logs, we construct a behavior provenance graph for each sample. To focus only on the actions originating from the sample, we first identify the main process node of the sample and then extract the "reachable subgraph" by traversing all nodes and edges accessible from this starting point. This filters out irrelevant system background noise and provides a focused, causal map of the sample's behavior.

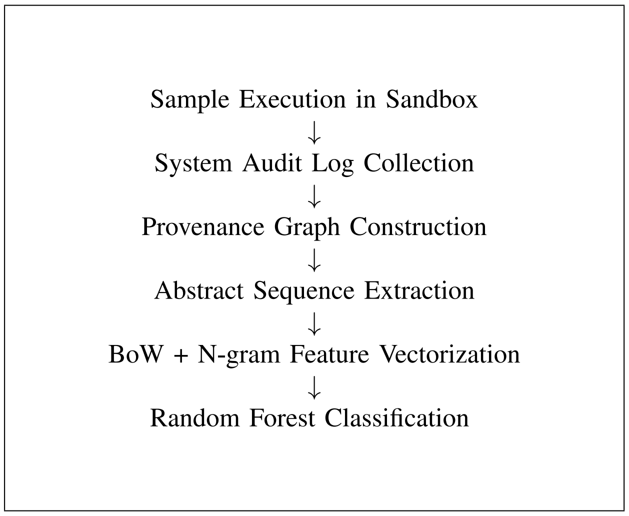
Sample Execution in Sandbox
System Audit Log Collection
Provenance Graph Construction
Abstract Sequence Extraction
BoW + N-gram Feature Vectorization
Random Forest Classification

Fig. 1. The overall workflow of the proposed detection model.

B. Feature Engineering

This stage transforms the structural graph data into a format suitable for machine learning.

1) Abstract Behavior Sequence Generation: We traverse the nodes in the reachable subgraph, following the causal and temporal order of events, to generate a raw event sequence. This sequence contains fine-grained details. To generalize, we abstract this raw sequence by mapping specific entities and actions to a predefined vocabulary. For example, any process creating another is mapped to the token ChildProcess, and a COM call is represented by the crucial token RpcCall . This process converts a complex series of events into a concise, symbolic sequence (e.g., [user_process, ChildProcess, RpcCall, ...]).

2) Vectorization: A purely sequential representation still isn't numerical. We use a Bag-of-Words (BoW) model to convert the abstract sequence into a feature vector based on term frequency. To overcome BoW's limitation of ignoring word order, we enhance it with N-grams. We extract not only single terms (uni-grams) but also pairs (bi-grams) and triplets (tri-grams) of consecutive tokens. The final feature vector for each sample is a concatenation of the BoW vectors for uni-grams, bi-grams, and tri-grams, capturing both the frequency of individual actions and the context of local action sequences.

C. Classification Model

We chose the Random Forest algorithm as our classifier. It is well-suited for this task due to its strong performance on high-dimensional and sparse data (like our feature vectors), its inherent resistance to overfitting, and its ability to be trained in parallel. For our experiments, the model was configured with 100 decision trees (n_estimators=100).

IV. Behavior Provenance Graphs

The raw logs obtained from dynamic behavior monitoring often consist of a large number of discrete event records. To better understand the relationships between these events and the overall context of malicious behavior, the Behavior Provenance Graph is introduced as a powerful data representation and analysis tool.

A. Definition and Structure

In the context of cybersecurity, a behavior provenance graph is a directed graph (which can be a Directed Acyclic Graph, DAG, or a general directed graph allowing cycles) that describes the causal or information flow relationships between system entities and system events.

Nodes in the graph represent various entities within the system. Examples include:

- **Process Nodes:** Containing attributes like process name and PID.
- **File Nodes:** Containing attributes like filename and path.
- **Network Nodes:** Containing attributes like IP address, port, and protocol.
- **Registry Key Nodes:** Containing the key path.
- **COM Object Nodes:** Potentially containing CLSID and IID information.

Edges represent the actions, events, or relationships between these entities. They are typically directed to indicate the flow of information or influence. Edges can also be augmented with attributes such as timestamps, specific API call names, parameters, and return values. For instance, an edge might represent that ”Process A created Process B” or ”Process C wrote to File D.”

B. Construction and Application

The construction of a provenance graph generally involves normalizing raw logs, extracting key events, identifying the entities involved, and inferring the dependency relationships between them. This structured representation offers several advantages over simple event lists. It provides a holistic, system-wide view of malware execution, enables the tracing of complex malicious activity chains (e.g., how a file was downloaded, then executed by a process, which then established network connections), and helps in identifying attack patterns that would be obscured in raw logs.

Common applications include malware detection, threat hunting, digital forensics, incident response, and the analysis of Advanced Persistent Threats (APT). However, challenges remain, such as the potential for massive graph scale and complexity, the ”semantic gap” in mapping low-level events to high-level attack tactics, and the need to distinguish malicious signals from the noise of normal system activities.

V. Experimental Results and Discussion

The model's performance was evaluated using 5-fold cross-validation on the 18,000-sample dataset.

A. Performance Metrics

The average accuracy across the five folds was a robust 0.9421, indicating strong generalization ability. Table I provides a detailed breakdown of the average precision, recall, and F1-score.

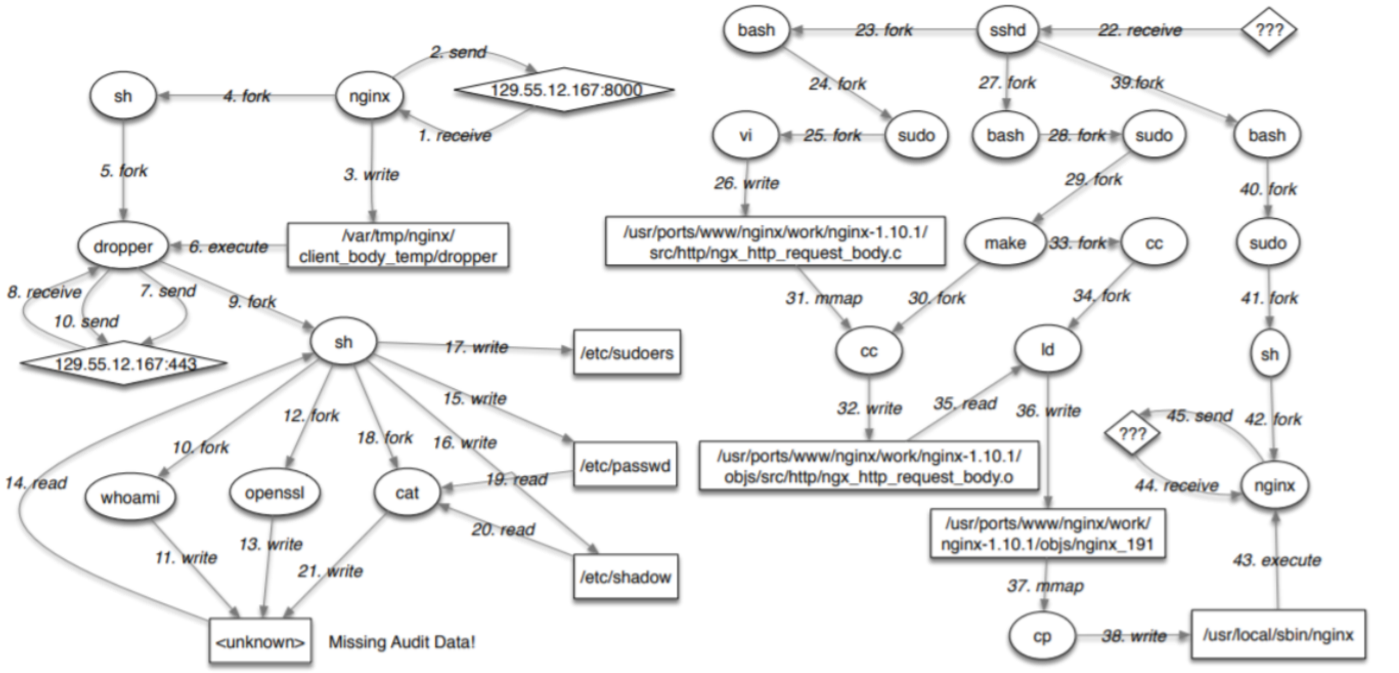

Fig. 2. Case of provenance graph

TABLE I
AVERAGE CLASSIFICATION REPORT FROM 5-FOLD CROSS-VALIDATION

Class/Average Type	Precision	Recall	F1-Score
Benign (0)	0.9386	0.9461	0.9423
Malicious (1)	0.9457	0.9381	0.9419
Macro Avg	0.9422	0.9421	0.9421
Weighted Avg	0.9422	0.9421	0.9421

B. Discussion of Results

The model achieved an F1-score of 0.9419 for the malicious class, built upon a high precision of 0.9457 and a high recall of 0.9381. The high precision is critical in a practical setting, as it means there is a very low chance of falsely flagging a benign application as malicious (low false positive rate), which avoids disrupting legitimate users. The high recall demonstrates the model's effectiveness in successfully identifying the majority of actual threats (low false negative rate).

The balanced performance across both malicious and benign classes, as shown by the nearly identical macro and weighted averages, confirms that the model is not biased towards one class. This is likely due to the balanced dataset and the feature representation's ability to capture the distinguishing characteristics of both types of software. The use of N-grams proved particularly effective, as malicious COM usage often forms specific, recognizable short sequences of operations (e.g., creating a process, then immediately making a remote procedure call), which are captured as discriminating features.

VI. Conclusion and Future Work

This paper proposed and validated a robust framework for detecting COM-based malware using dynamic behavior analysis, provenance graphs, and a Random Forest classifier. By converting complex behavioral graphs into abstract, sequential feature vectors, our method successfully bridges the gap between system-level event data and effective machine learning classification, achieving an average accuracy of 0.9421 on a large dataset.

While the results are promising, several avenues for future research exist.

- **Advanced Representations:** Exploring more sophisticated graph embedding techniques, such as Graph2Vec or Graph Neural Networks (GNNs), could capture deeper structural information from the provenance graphs directly, potentially improving performance.
- **Threat Evolution:** Future work should focus on continuously updating the training dataset with new malware samples, especially those designed for evasion, and investigate online or incremental learning models to adapt to the evolving threat landscape in real-time.
- **Practical Integration:** We plan to work towards translating this research into a practical tool that can be integrated with existing security infrastructure like Endpoint Detection and Response (EDR) or SIEM systems to enhance their threat detection capabilities.