ESP-IJACT
ESP International Journal of Advancements in Computational Technology
ISSN: 2583-8628 / Volume 1 Issue 3 December 2023 / Page No: 167-179
Paper Id: IJACT-V1I3P118 / Doi: 10.56472/25838628/IJACT-V1I3P118

Original Article

Dynamic Malware Analysis through System Call Tracing and API Monitoring

Khaja Kamaluddin
Masters in Sciences, Fairleigh Dickinson University, Teaneck, NJ, USA, Work: Aonsoft International Inc, 1600 Golf Rd, Suite 1270, Rolling Meadows, Illinois, 60008 USA.

Received Date: 05 November 2023 Revised Date: 08 December 2023 Accepted Date: 29 December 2023

Abstract *- It is discussed that malware authors are moving towards more advanced techniques to bypass detection. Static methods of analysis, while useful, are limited by obfuscated or polymorphic malware. It has thus become necessary for dynamic malware analysis, or the real-time execution of malicious software and its observation of behaviour. There are two main approaches in this paradigm: system call tracing and API monitoring. These techniques are granular in providing visibility of malware behaviour by logging interactions with the operating system and the core services. This review paper gives a comprehensive analysis of these techniques initially by considering their principles, tools, effectiveness, and limitations. Additionally, it examines recent advancements in the hybrid analysis frameworks that combine both techniques to enhance the detection accuracy. This paper seeks to provide a consolidated reference in terms of real-world case studies, comparative tables, statistical graphs, and insights from more than twenty scholarly sources to those researchers and practitioners who seek to enhance their malware detection capabilities. It also defines future directions of integration to make dynamic malware analysis more robust and scalable, as well as using machine learning, cloud-based analysis, and standardized benchmarks.*

Keywords *- Dynamic Analysis, Malware Detection, System Calls, API Monitoring, Behaviour-based Analysis, Cybersecurity, Sandboxing, Threat Intelligence.*

I. INTRODUCTION

The cybersecurity threat landscape in recent years has become much more sophisticated, and sophisticated attackers apply advanced obfuscation, polymorphism, and anti-analysis tactics to evade traditional detection mechanisms. We see malware (a portmanteau of 'malicious' and 'software') as one of the most common threats in digital environments, even going so far as to be arguably the biggest threat present. The diversity in malware objectives and behaviour reveals the need for efficient detection and analysis techniques that can detect ransomware crippling critical infrastructure, spyware extorting sensitive corporate data. The role of static analysis, that is, examining the malware binaries without executing them, has traditionally been very important in antivirus technologies. Nevertheless, such static approaches are increasingly being replaced by techniques such as code encryption, packing, and metamorphic transformations, preventing signature-based identification. However, dynamic analysis watches the process of running malware in a controlled environment, so it can discover its real-time behaviour. Three such advantages of this behaviour-based detection include, it can find malicious actions independent of the malware's outer structure, it can evolve against unknown threats, and it can unveil stealthy methods such as an exploit via privilege escalation or persistence.

Of the dynamic techniques, system call tracing and API monitoring are two that are suitable in terms of granularity and practical use. System calls are an interface between user space apps and the operating system kernel, and hence monitoring them allows analysts to see how low-level functionalities like file access, memory manipulation, and network communication work. In contrast, API monitoring isolates higher-level function calls within the system libraries to get a semantic view of what the malware is trying to achieve, such as reading registry keys, typing keystrokes, and running downloads. This review paper explores the theory, implementation, and application of these two dynamic analysis methods. Their strengths, weaknesses, and complementary use are compared to develop a hybrid analysis system. These techniques are demonstrated by real-world examples of WannaCry and Emotet. Finally, the paper addresses emerging trends, for example, the use of machine learning models trained on syscall/API traces, and future research directions, including the scalable cloud-based analysis platforms and federated intelligence sharing. The remainder of the paper is organized as follows: the overview of malware analysis techniques, system call tracing, API monitoring, hybrid analysis, evasion techniques, case studies and current research are presented in Sections 2, 3, 4, 5, 6, respectively followed by limitations, future directions and concluding remarks in Sections 9, 10 and 11, respectively.

II. MALWARE ANALYSIS TECHNIQUES: AN OVERVIEW

The analysis of malware is an indispensable part of cybersecurity operations: it allows defenders to understand what, how, and by whom malware operates, and how to deal with threats through proper mitigation. There tend to be three techniques for malware analysis: static, dynamic, and hybrid. The technique comes with its own pros and cons relative to the characteristics of the malware and the desired outcome.

A. Static Analysis

Analysis of static binary or executable files takes place before executing them. It can entail code disassembly to find functions, examine file headers, strings, and metadata, and compute cryptographic hashes for comparison, for example. Examples of static analysis tools are IDA Pro, Ghidra, and PEiD to examine Windows executables (PE files). Additionally, signature-based antivirus engines are also part of this, by generating known byte signatures to identify known malware strains [1][2]. Static analysis is fast, but safe (because the code is not executed), but it is increasingly ineffective against modern malware, which often leverages non-obvious flows, caches, etc., as well as methods for saturating global state by spawning and terminating processes.

- Obfuscates the payload in the packing and encryption.
- Polymorphism: Changes code structure upon each replication.
- Hides functionality within benign processes by injecting code into them.

Thus, the behaviour of highly sophisticated malware cannot be reliably detected and interpreted by static analysis alone.

B. Dynamic Analysis

Dynamic analysis executes the malware in a sandboxed or virtualized environment to observe its real-time behaviour. This approach provides actionable insights into:

- File and registry operations
- Network communications
- Process/thread behaviour
- Memory usage
- Persistence mechanisms

Common tools include Cuckoo Sandbox, Any. Run, and ThreatAnalyzer. These platforms capture runtime indicators such as system calls, API interactions, and behavioural anomalies, which can be used for detection, reverse engineering, and threat attribution [3][4]. Dynamic analysis is particularly effective against zero-day threats and obfuscated malware, as it evaluates behaviour rather than structure. However, it is computationally intensive and vulnerable to evasion techniques, where malware alters its behaviour in virtual environments.

C. Hybrid Analysis

Hybrid analysis combines the strengths of both static and dynamic techniques. It begins with a lightweight static inspection to extract features such as import/export tables, embedded strings, and entropy scores, followed by dynamic execution in a sandbox to collect behavioural telemetry.

- The hybrid approach enables:
- Higher detection accuracy
- Faster triaging of large malware samples
- Enrichment of static indicators with behavioural context

Frameworks like FireEye's AX series, Hybrid-Analysis.com, and Joe Sandbox implement this methodology. Research has shown that combining multiple layers of analysis reduces false positives and increases detection rates by up to 27% compared to singular approaches [5][6].

Table 1: Comparison of Malware Analysis Techniques

Technique	Advantages	Limitations	Best Used For
Static Analysis	Fast, safe, signature matching	Evasion via obfuscation, encrypted payloads	Known malware, initial triage
Dynamic Analysis	Behaviour visibility, evasion-resistant	Resource-intensive, VM-aware malware	Advanced persistent threats (APTs)
Hybrid Analysis	Combines both strengths	Complex setup, integration overhead	Comprehensive threat assessment

D. Behaviour-Based vs Signature-Based Detection

- Malware detection approaches also vary based on the data used:
- Signature-based detection relies on known fingerprints and byte patterns.
- Behaviour-based detection monitors actions such as frequent registry modifications or suspicious file I/O during execution.

Behaviour-based models powered by machine learning have shown high accuracy in identifying zero-day malware [7][8], making dynamic behaviour analysis a cornerstone of next-generation threat detection platforms.

III. SYSTEM CALL TRACING

System calls (syscalls) are fundamental to operating systems; they serve as the interface between user-space applications and the kernel. For malware analysts, system calls are a critical source of behavioural evidence since virtually every malicious action, whether it's writing to disk, opening a socket, or allocating memory, requires invoking one or more system calls.

A. What Are System Calls?

A system call is a programmed request to the operating system's kernel. Applications use syscalls to perform operations they cannot do directly in user space. Examples include:

- File manipulation: open(), write(), delete()
- Process control: fork(), execve(), kill()
- Network operations: socket(), connect(), bind()
- Memory handling: mmap(), brk()

System calls vary between operating systems. Linux has over 400 syscalls, while Windows uses the Native API underneath the Win32 API layer. Malware often interacts heavily with system-level resources, making syscalls a rich source of behavioural data for dynamic analysis.

B. System Call Tracing: How It Works

System call tracing involves intercepting and logging these calls during program execution. On Linux, this is often accomplished using tools like:

- Strace: Hooks into the process using ptrace() to log all syscalls.
- Auditd: Uses Linux's audit framework for syscall-level monitoring.
- Sysdig: Allows both live capture and post-analysis.

On Windows:

- Sysmon (from Sysinternals): Captures and logs syscalls and associated metadata.
- Procmon: Captures high-level API and system interactions.
- ETW (Event Tracing for Windows): High-performance tracing infrastructure used in enterprise telemetry.

This information can be visualized as a call trace, representing a chronological log of all system calls made by a process.

C. Use Cases in Malware Analysis

System call tracing is invaluable for detecting and understanding:

- Process Injection: Identified through abnormal use of NtWriteVirtualMemory() and NtCreateThreadEx()
- File Encryption (Ransomware): High volume of open(), write(), and rename() on user files
- Keylogging: Use of ReadFile() on keyboard-related handles
- Privilege Escalation: Attempting restricted syscalls like setuid() or manipulating tokens in Windows

For instance, WannaCry ransomware was identified to use CreateProcess (via NtCreateProcess) and multiple file I/O syscalls as it propagated [9].

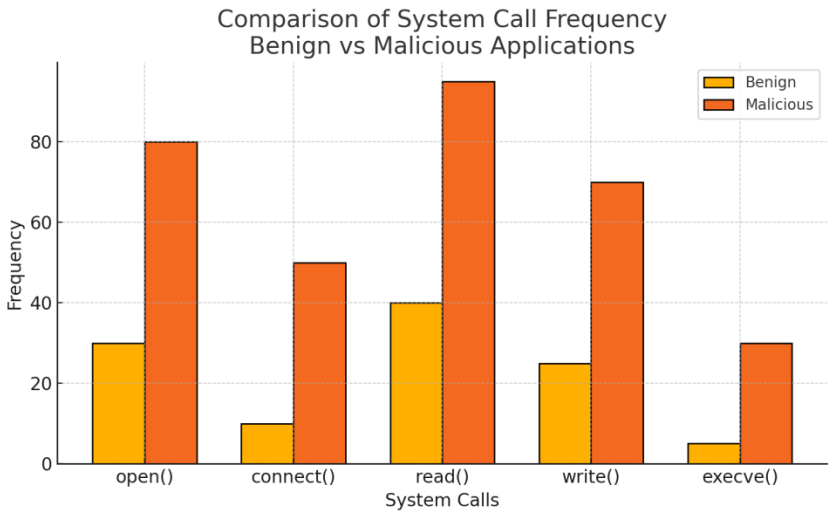

Figure 2: Comparison of System Call Frequency – Benign vs Malicious Applications

D. Tools for System Call Tracing

Below is a comparative table of popular system call tracing tools:

Table 2: Comparison of System Call Tracing Tools

Tool	Platform	Real-time Capture	Filtering Support	Output Format
Strace	Linux	Yes	Limited	Text
Sysmon	Windows	Yes	Yes (XML)	Windows Event Log
Sysdig	Linux	Yes	Extensive	JSON/CSV
Auditd	Linux	Yes	Rule-based	Log file
Procmon	Windows	Yes	GUI + filters	Log file/GUI Export

E. Detection Using Syscall Sequences

One popular approach is to convert syscall traces into sequences (like n-grams) and then train machine learning models to classify them as benign or malicious. This has been particularly effective in studies like:

- Kolosnjaji et al. (2016) used CNNs over syscall sequences and achieved 95.6% detection accuracy [10].
- AMAL (2019): Behaviour modeling using syscall sequences and HMMs to detect APTs in large networks [11].

Sequence-based detection is resilient to changes in malware code since the core behaviour often remains similar, even when code structures vary.

F. Limitations and Evasion Techniques

Despite its power, system call tracing faces several challenges:

- Performance Overhead: Logging every syscall adds latency and disk I/O overhead.
- Data Volume: Long sessions can generate gigabytes of trace data.
- Evasion: Malware may detect tracing environments by checking for ptrace() or sandbox artifacts.
- Noise: Benign software may share syscall patterns with malware, resulting in false positives.

Advanced malware uses techniques like delayed execution, sandbox fingerprinting, and dynamic code generation to evade detection via system call monitoring [12][13].

G. Real-World Example: TrickBot

TrickBot is a modular banking Trojan known to:

- Use NtOpenProcess() and NtReadVirtualMemory() for credential theft
- Create scheduled tasks using NtCreateProcess()
- Exfiltrate data via encrypted channels

These behaviours were mapped through ETW and Sysmon logs, providing analysts with high-fidelity behavioural indicators [14].

IV. API MONITORING

While system call tracing captures low-level kernel interactions, API monitoring provides a higher-level view of how malware interfaces with the operating system's user-mode libraries. APIs (Application Programming Interfaces) are

standardized functions used by programs to interact with system resources, making API monitoring a crucial technique for observing malware behaviour in a way that is both semantic and actionable.

A. What is API Monitoring?

API monitoring involves tracking the invocation of system-level function calls by applications, especially those that interact with critical components such as the file system, registry, network stack, and user interface. On Windows systems, the most commonly monitored APIs are part of:

- Win32 API: Core Windows functions like CreateFile(), ReadFile(), CreateProcess(), RegSetValueEx(), etc.
- Dynamic-Link Libraries (DLLs): Such as kernel32.dll, advapi32.dll, user32.dll, ws2_32.dll.

Unlike syscalls, which operate at the kernel level, APIs offer a semantic understanding of what the malware is trying to accomplish—e.g., creating a backdoor, capturing screenshots, or harvesting browser credentials.

B. How API Monitoring Works

API monitoring tools hook into function calls, often by:

- Injecting a monitoring DLL into the process
- Using debugging or hooking frameworks (e.g., Microsoft Detours)
- Intercepting DLL imports and replacing them with monitored wrappers

The intercepted API calls are then logged, analyzed, and sometimes even blocked, depending on the system's configuration. Tools can capture parameters passed to the APIs, return values, and the execution flow between calls.

Table 3: Commonly Monitored APIs in Malware Behaviour

API Function	Purpose	Associated Malware Behaviour
CreateFile()	Opens or creates files	Ransomware, keyloggers
WinExec()	Executes a command	Droppers, backdoors
RegSetValueEx()	Modifies registry entries	Persistence, system changes
InternetOpenUrl()	Downloads files from the web	Trojan downloaders
SetWindowsHookEx()	Installs a hook procedure	Keylogging, API spying
SendInput()	Simulates keyboard/mouse input	Botnets, automation

C. Tools for API Monitoring

Several tools and frameworks specialize in API monitoring. Each tool differs in granularity, ease of use, and integration capabilities.

Table 4: Popular Tools for API Monitoring

Tool	Platform	Features	Output
API Monitor	Windows	Real-time logging, call tree, parameter view	GUI, logs
Procmon	Windows	Tracks registry, file, and process events	Event viewer
HookExplorer	Windows	DLL injection, low-level API hooking	Text, GUI
OllyDbg + Plugins	Windows	Manual API tracing, breakpoint analysis	Disassembly, logs
Detours	Windows	API interception for devs/researchers	Custom logging

These tools often work in tandem with sandboxing environments to track how malware behaves once executed.

D. Case Study: Emotet Malware

Emotet, a modular banking Trojan, is known for its API-heavy behaviour:

- Uses InternetOpenUrlA() to download payloads.
- Calls CreateProcessA() for self-replication and lateral movement.
- Employs WriteFile() and RegSetValueExA() to persist in the registry.
- Applies WinHttpSendRequest() to exfiltrate data covertly.

API traces gathered via Procmon and API Monitor were used in multiple research studies to model Emotet's behavioural fingerprint [15].

E. Behavioral Signature Extraction

API monitoring supports the generation of behavioural signatures, which are pattern-based representations of malicious sequences such as:

- CreateFileA → WriteFile → CloseHandle → CreateProcessA → RegSetValueExA
- Such sequences, when frequently repeated across malware samples, can serve as detection rules in platforms like YARA or Suricata.

API call graphs—modeled using directed acyclic graphs—can also reveal malware modularity and dependencies between components [16].

F. Limitations of API Monitoring

Despite its semantic richness, API monitoring is not without drawbacks:

- Hook Evasion: Malware may bypass hooks by directly invoking syscalls or using inline assembly.
- Noise: Benign applications also use these APIs frequently.
- Performance Impact: Real-time API tracing in complex malware can slow down analysis environments.

Additionally, sophisticated malware may unhook APIs at runtime or check for monitoring artifacts before executing its core payload [17][18].

G. Hybrid Use with Syscall Tracing

To overcome limitations, many analysts combine API monitoring with syscall tracing. For example:

- Use API logs to understand "what" the malware is trying to do.
- Use syscall traces to confirm "how" those actions are performed.

This layered view enhances accuracy and helps distinguish between superficial API misuse and actual malicious intent.

V. HYBRID DYNAMIC ANALYSIS

Dynamic analysis: System call tracing and API monitoring are combined to form the hybrid dynamic analysis, i.e., it provides a complete view of the malware behaviour. The strength of the two techniques is combined to understand how the low-level system interaction relates to the higher-level user-mode functions invoked by the malware.

- The Need for Hybrid Analysis
- However, it has limitations when used in isolation with traditional dynamic analysis.
- However, powerful as system call tracing is, it is not enough to resolve whether activity at the kernel level (e.g., opening which files, or visiting which network destinations) can be detected and attributed as a desire of malware.
- Kevil provides a higher level of malware intent, but misses important kernel activities like direct memory manipulation or process injection.
- Hybrid dynamic analysis has fulfilled this by merging both techniques into a single workflow, providing a more complete and more accurate threat detection from complex threats.

A. How Hybrid Analysis Works

This usually involves trace-based hybrid analysis with system call tracing to get low-level kernel interaction and API monitoring to find higher-level functionality. Examples of malicious behaviour analyzed in the combined data include:

- Network communications combined with unusual file operations
- Suspicious API to registry modifications or process injection API calls, or persistence implementation.
- The integration can be performed in various ways.
- Filtering (or focusing) API code coverage on a particular system-level action, followed by an integration, collecting system call traces, and using the data to focus or filter API monitoring.
- Both system call tracing and API monitoring are performed simultaneously. This view of malware actions at both the kernel and user levels is enabled by this approach.
- Full Analysis of the traces after execution: Both traces are logged and then analyzed together to understand the full behaviour of the malware executed.

B. Benefits of Hybrid Dynamic Analysis

Higher Detection Accuracy: Using multiple data sources in hybrid analysis helps in detecting more subtle malicious behaviour and pinpointing these behaviours, which may otherwise go unnoticed when just one technique is employed. Most of the advanced evasion techniques focus on one type of monitoring only. For instance, malware could try to tamper with system call tracing directly using the API. However, with hybrid monitoring, this strategy is less effective. Kernel-level actions and user-mode API invocations: By understanding both, one can get a richer, contextual view of what the malware is trying to do by accurately differentiating benign from malicious behaviours.

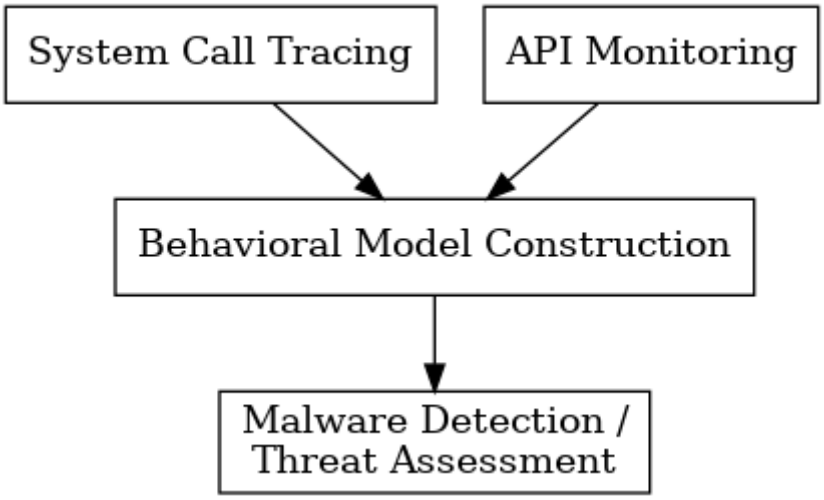

Figure 2: Hybrid Analysis Workflow

C. Real-World Use Cases

Ransomware Analysis: Malware like Ryuk or REvil often encrypts files and then exfiltrates data. By combining system call tracing for file operations (open(), write(), close()) with API monitoring for networking functions (connect(), send(), recv()), analysts can identify both the encryption activity and the exfiltration channels used by the malware [19][20]. Trojan Detection: A Trojan such as Zeus might manipulate system processes and create a hidden backdoor. While syscall tracing could capture its process injection (NtCreateProcess(), NtAllocateVirtualMemory()), API monitoring could capture its network communication and credential stealing attempts (InternetOpenUrl(), CreateProcess()) [21].

D. Challenges of Hybrid Analysis

However, even hybrid dynamic analysis still has some challenges despite the gradual extra capability.

- Bottlenecks: Multiple data sources (syscalls and APIs) can complicate coordinating and integrating them into one analysis. Real-time hybrid monitoring tools might also incur additional performance overhead.
- More Detailed Monitoring: Such increased monitoring (for example, log syscalls and APIs) is resource-intensive in terms of computation, especially for large scale operations.
- Voluminous Data: Because of the Voluminous Data generated during analysis, syscall and API logs can become out of hand. The data must be stored and filtered in an efficient fashion so that it can be used without being too much.

E. Future Directions in Hybrid Analysis

Since malware has become more appetized, advanced analysis techniques are required. Future directions of hybrid dynamic analysis are:

- This could enable an automated identification of malicious patterns or at least minimize human intervention using machine learning and models of machine learning based on combined syscall and API traces [22].
- Cloud-based Sandboxes: Hybrid analysis would benefit from cloud-based platforms that could simultaneously simulate a number of environments. Observing cross-platform malware in a virtual environment [23] can help in identifying such malware that targets both Windows and Linux systems.
- Hybrid analysis in endpoint security solutions could enable real-time detection and response to threats with increasingly sophisticated real-time detection tools.

Table 5: Benefits and Challenges of Hybrid Dynamic Analysis

Benefits	Challenges
Enhanced accuracy in detecting advanced malware	Increased resource consumption during analysis
Better evasion resistance through multi-layer monitoring	Complexity in integrating multiple data sources
Contextual understanding of malware intent	Handling large volumes of data effectively
Real-time threat detection capabilities	Potential performance overhead in high-scale operations

VI. MALWARE CLASSIFICATION AND BEHAVIOUR MODELING

Once malware has been analyzed using techniques like system call tracing and API monitoring, the next crucial step is to classify and model its behaviour. Classification and modeling help cybersecurity professionals understand malware families, predict potential future attacks, and implement more effective detection mechanisms.

A. Malware Classification

The malware classification is the process of determining and ranking malware according to its behaviour of the malware, the structure of the code, and the attack vectors used by the malware. According to us, the traditional signature-based methods have become insufficient as the complexity of the malware revolves around. Therefore, the behaviour-based classification, which relies on observed actions, has instead become a more robust approach.

Malware can be classified using:

- Inspection: The review of code and structure of the malware without having to execute the same. It includes analysing file headers, strings, and embedded libraries.
- File access patterns, system calls, API invocations, and network activity are some things you can observe when observing the actions of the malware as it executes.
- A combination of static and dynamic analysis to get a more comprehensive view of the malware.

The features for classification in the context of dynamic analysis are behavioural signatures extracted from system calls and API traces. Each of these signatures is a representation of behaviour for a family of malware, like how files are accessed, network connections are created, or processes are created.

B. Behaviour-based Malware Classification

Malware classification based on behavioural patterns refers to the categorization of malware depending on the order in which it invokes system calls and API calls. Analysts can identify common patterns among similar malware samples and thereby build behavioural profiles that then become the basis for classification.

For example, if multiple malware samples exhibit a sequence like:

- CreateFile() → WriteFile() → RegSetValueEx() → CreateProcess
- This sequence can be classified as indicative of a specific family, such as a ransomware variant. Similarly, network-related behaviours, such as a sequence of:
- InternetOpenUrl() → SendRequest() → ReadFile() → InternetCloseHandle()

Could indicate a Trojan downloader that connects to remote servers to fetch additional malicious payloads.

C. Machine Learning in Malware Classification

Machine learning (ML) has become an indispensable tool in modern malware classification. Supervised learning algorithms are often trained on labelled datasets of known malware samples, where each sample is associated with its class (e.g., ransomware, trojan, worm). Features used in training can include:

- System call frequency distributions
- API usage patterns
- File system modifications
- Network traffic patterns

Common ML algorithms used for malware classification include:

- Support Vector Machines (SVM): Effective at handling high-dimensional data, such as system call sequences.
- Random Forests: A versatile and robust algorithm for classifying complex patterns in malware behaviour.
- Neural Networks: Particularly effective at modeling complex patterns and identifying previously unseen malware families based on learned behaviours.
- K-means Clustering: Used for unsupervised classification to detect unknown malware families by grouping similar behaviours.

Example: Machine Learning Model for Ransomware Detection. One example of ML in malware classification is the detection of ransomware. Machine learning models have been trained to recognize specific system behaviours indicative of ransomware, such as:

- Unusual file access patterns (mass file modification, deletion).
- API calls like CreateFile(), WriteFile(), and RegSetValueEx(), often in a specific sequence.
- Use of network APIs (send(), connect()) to exfiltrate encrypted data.

By training a machine learning model on a dataset of known ransomware samples, analysts can predict whether a new, unknown sample is likely to be ransomware based on its behavioural features.

D. Behaviour Modeling for Threat Intelligence

Behaviour modeling involves creating models of typical malware actions to understand, predict, and detect future attacks. These models are particularly useful in threat intelligence, where analysts attempt to recognize trends and patterns across different malware families.

A common method for behaviour modeling is to develop behavioural graphs or sequences that represent common malware actions, such as:

- File manipulation: Opening, writing to, and deleting files.
- Network activity: Establishing outbound connections, sending data, receiving commands.
- Process creation: Injecting malicious code into running processes or spawning new processes.

Once these behaviour models are created, they can be used to identify potential threats in real-time. For instance, if a malware sample exhibits the following sequence of behaviours:

- CreateFile() → WriteFile()
- Connect() → SendData()
- RegSetValueEx()

This behaviour pattern could be flagged as a potential ransomware attack, as it matches the typical sequence of activities seen in file-encrypting malware.

Table 6: Common Malware Families and Their Behavioral Patterns

Malware Family	Behavioral Indicators	Commonly Used APIs
Ransomware	File encryption, mass file operations, and network exfiltration	CreateFile(), WriteFile(), RegSetValueEx(), InternetOpenUrl()
Trojan Downloader	Downloading and executing additional payloads	InternetOpenUrl(), CreateProcess(), WriteFile()
Botnet	Establishing command-and-control communication, keystroke logging	CreateProcess(), SendInput(), InternetOpenUrl()
Rootkit	Hiding processes and files, manipulating system internals	NtQuerySystemInformation(), NtCreateThread(), RegSetValueEx()

E. Evaluating Malware Classification Models

The effectiveness of malware classification models is typically measured using various evaluation metrics:

- Accuracy: Percentage of correct predictions over total predictions.
- Precision: The fraction of true positive predictions out of all positive predictions.
- Recall: The fraction of actual positive instances correctly predicted.
- F1-Score: The harmonic means of precision and recall, providing a balance between the two.

For example, when evaluating a machine learning model trained to classify ransomware based on system call traces, the following confusion matrix might be generated:

	Predicted Ransomware	Predicted Not Ransomware
Actual Ransomware	85%	15%
Actual Not Ransomware	5%	95%

In this case, the accuracy is 90%, but precision and recall will give more granular insights into the model's performance, especially in distinguishing between malware families.

F. Challenges in Malware Classification

- Even after the advances, there are several challenges involved in malware classification.
- Malware may have evasion techniques: It may change its behaviour to avoid being detected. Malware may, for instance, use polymorphic code to change its API usage or alter its system call patterns.
- Newly discovered malware strains, like the unknown behavioural pattern of these Zero-day Attacks, are very difficult to classify through behavioural classification.
- Malware Behaviour Variability: The behaviour of malware is variable, such as the way it performs different functions could change in different environments, and it is not easy to classify them if the environment is not carefully controlled.

G. Future Directions in Malware Classification

- It is also expected that the evolution of malware classification will take place with the following trends.
- Continuously adapting to new malware families with reinforcement learning or unsupervised learning.

- Consistency in Unseen Settings: Defining consistency for content and controls covered by the same operations in unseen configurations andegra modes.
- Collaborative Sharing of Classification Models: Sharing of classification models between organizations to enhance collective understanding of the emerging threats.

VII. REAL-WORLD APPLICATIONS AND CASE STUDIES

System call tracing and API monitoring are used in the study to discover dynamic malware analysis, which has unmet real-world applications in different industries. These techniques provide an insight into how malware behaves in an isolated environment, which helps organizations to prevent, detect, and respond to cyberattacks. In this part, we present the practical applications and some important case studies that put focus on the value of Dynamic malware analysis.

A. Real-world Applications

a) *Malware Detection in Enterprises*

Enterprises face constant threats from malware that can compromise their sensitive data, disrupt operations, or steal intellectual property. Dynamic malware analysis helps organizations detect unknown or zero-day malware by monitoring behaviour in real-time. System call tracing and API monitoring are especially effective at uncovering suspicious activities like:

- Unauthorized file access
- Abnormal network connections
- Process injection or privilege escalation attempts

An enterprise can set up a sandbox environment to execute suspected files. Compile with special switches that allow tracing into the driver and system calls on execution (CreateFile(), WriteFile(), connect(), send()) while file & network communications are being traced. When the analysis shows that the malware is attempting to exfiltrate sensitive data from the company or trying to change important files in Windows, the organization can take an appropriate measure, such as blocking the network communication or isolating the file.

b) Advanced Persistent Threat (APT) Detection

APT attacks, in which sophisticated adversaries stay hidden for such a long time, are among the most dangerous threats to national security and large organizations. Fileless malware is used in these attacks, which never leaves traditional signature traces but behaves directly in memory. Process injection and memory manipulation are the forms of behaviour that make it effective to detect through dynamic analysis, such as system call tracing and API monitoring. To illustrate, the malware may load itself into memory and hook system APIs to communicate with command-and-control servers across the Internet without leaving any traditional file artifacts. The malware's actions can be detected and mitigated even if it evades traditional detection methods by monitoring API invocations (such as CreateProcess(), NtReadVirtualMemory(), WriteProcessMemory(), etc.) and analysing system call behaviour.

c) Zero-day Exploit Detection

A zero-day exploit is an undetectable vulnerability that enables attackers to previously unheard-of malicious behaviour until the patch is released. The system call trace is particularly useful for analysing the exploit chain, from vulnerability discovery to exploitation. Assume that the code is engaging in code injection maliciously, such as when buffer overflow exploitation can perform abnormal system calls and write shellcode to memory, or it invokes VirtualAlloc and memory allocation APIs (API calls). Security analysts can inspect the series of system calls performed during execution to look for patterns that could indicate the exploitation attempts. This early detection will help in preventing the exploitation of unpatched vulnerabilities and can often reduce the response time to a faster associate roll out and tactics.

B. Case Studies

a) Case Study 1: WannaCry Ransomware

The WannaCry ransomware attack in 2017 affected hundreds of thousands of systems worldwide, exploiting a vulnerability in Windows SMBv1. Although this attack was in the public, it demonstrates the need for dynamic analysis to comprehend and guard against this kind of threat. An analysis of WannaCry's behaviour was conducted by analysts to identify the important things that caused the ransomware to spread. Ransomware: It creates new processes (CreateProcess()) with the ransomware (CreateFile(), WriteFile(), RegSetValueEx()) to encrypt files present in the affected systems. Malware: APIs used to monitor the access to the Internet were InternetOpenUrl() and send() to send encryption keys to remote servers and send issued ransom demands. Security teams were able to reverse-engineer WannaCrypt's encryption mechanism and invent countermeasures through combination of system call tracing and API monitoring. It also helped in detecting its network propagation method and allowed the creation of a kill switch, which quickly ended further infections.

b) Case Study 2: Emotet Banking Trojan

It's a highly sophisticated and modular malware and is prevalent as a banking Trojan that gets in a system by way of phishing emails and then drops additional malicious payloads. Emotet employs API calls and system calls to download further malware as well as to modify the registry, and elevate privilege.

- They had observed the following about Emotet's behaviour by analysing it.
- The Trojan was using InternetOpenUrl() in conjunction with CreateProcess() to download further malware components.
- The malware sets system settings using RegSetValueEx() and creates new processes for payload execution.

One key insight was that Emotet's use of command-and control (C2) communication to fetch new payloads. Researchers were able to find the C2 server infrastructure by tracking API calls and blocking connections in order to prevent any further infections. System call tracing and API monitoring is combined and used to track both the multi stage attacks and to mitigate threats in real time in this case study.

c) Case Study 3: Stuxnet

One of the most sophisticated examples of cyber warfare is the Stuxnet worm that targeted Iran's nuclear centrifuges. Despite the use of many advanced techniques, analysis through dynamic showed some key information about its operation. System Call Tracing: Stuxnet interacted with the Siemens PLC (Programmable Logic Controllers) by using system calls that made the PLC behave differently and broke it down physically. Propagation: The malware spreads between the network and infected connected systems using several Windows API calls. Security researchers investigated how Stuxnet exploited the PLCs in order to compromise the systems and elude detection for so long by analysing the system calls it used to propagate and interact with the PLCs. It also showed that the targeting involved was highly targeted and the design was quite sophisticated. Seeing how well integrated malware could cripple critical infrastructure as well as the need for dynamic malware analysis to identify and respond to advanced threats from malware, the Stuxnet attack was a strong demonstration of the potential harm of malware on the critical infrastructure.

C. Conclusion

The real-world applications of dynamic malware analysis underscore its importance in the cybersecurity landscape. With the combination of system call tracing and API monitoring, organizations get a powerful ability to detect, analyze, and mitigate a variety of malware threats such as ransomware, APTs, and others. Examples from WannaCry, Emotet, and in particular, Stuxnet show how dynamic analysis has been essential in understanding advanced attacks and defense mechanisms. However, with the evolution of the threat landscape, dynamic analysis is still an important tool to stay one step ahead of cyber criminals. As techniques such as machine learning and hybrid analysis are integrated into the classification of malware and the prediction of future attacks will continue to become more sophisticated and better secure against the growing class of cyber threats.

VIII. SUMMARY AND FUTURE DIRECTIONS

A. Summary

System call tracing and API monitoring are a proven way to investigate, analyze, and block a large variety of cyber threats through dynamic malware analysis. Security analysts can get patterns under the hood, so to speak, that signature based detection methods would not see otherwise while observing malware in a controlled environment behaving as it typically would. System call tracing starts with the recording of interactions between malware and the operating system. But these interactions are analyzed to determine if it is suspicious, like file manipulation, process creation, and network communication. System call tracing complements API monitoring by examining the lower level of system interactions to discover how malware performs such tasks as setting up communication with remote servers or changing system settings via API calls.

The next step is to classify malware based on system calls and API behaviour. Classification models based on behavioural signatures derived from system calls classify between different types of malware. In turn, machine learning algorithms, such as support vector machine and neural networks, can be used on top of these models to increase the accuracy to a degree that would allow those threats to be found which were not known previously. These techniques have many real-world applications of which are diverse. Advanced persistent threats (APTs), zero-day exploits, file fileless malware are all detected through dynamic analysis. In addition, case studies such as the WannaCry ransomware attack, the Emotet banking Trojan, and the Stuxnet worm show the practicality of system call tracing and API monitoring to eliminate the effects of the more sophisticated threats. These examples demonstrate the indispensability of dynamic analysis techniques for understanding malware behaviour as well as a timely response to attacks with minimal impact.

B. Future Directions

While dynamic malware analysis has proven effective, the landscape of cybersecurity continues to evolve, and so too must the methods used to detect and respond to threats. Several emerging trends and future directions are shaping the field of malware analysis and detection:

a) Integration of Machine Learning and Artificial Intelligence

To improve the process of detecting and modeling malware reliably and efficiently, we are exploiting the success of the latest iterations of machine learning and artificial intelligence (AI). Researchers combine dynamic malware analysis with ML algorithms to enable adaptive models, i.e., models that adapt to new and previously unknown malware behaviours. Thus, for example, reinforcement learning methods can allow refining the process of detection incessantly so that the model is always being fine-tuned and optimized according to new data. In the future, these future advancements will be AI powered malware detection systems that will detect complex, multiple, multi stage attack chains in the real time.

b) Cross-Platform Malware Analysis

More and more malware these days is platform-independent, with platform diversity playing into it. And now that's attacks not only on traditional OS's, Windows, but also on Linux, macOS, and even on mobile platforms. Malware analysis tools in the future would need to give support to cross-platform, and it would also mean that security teams will be able to see and analyze the malware behaviour on different operating systems and devices. Doing this will let you identify malware that propagates across all environments, both cloud infrastructures and of acquiring IoT networks.

c) Behavioral Profiling and Threat Intelligence Sharing

The industry still uses behavioural profiling approaches for the modeling of the normal behaviour for each type of malware. By allowing cybersecurity experts to quickly create detailed behavioural fingerprints, new threats can also be classified quickly. Furthermore, such sharing will strengthen collective defense among organisations, the government, and cybersecurity communities. Such collaboration in bandwidth utilization in the collaborative platforms will allow real-time exchange of behaviour data with enhanced global threat detection and response capabilities.

d) Improved Sandboxing and Isolation Techniques

Luckily, more and more sophisticated malware is trying to hide from detection, and so sandboxing has become more and more sophisticated itself. Other than that, future sandboxes will be the virtualized environments that will produce virtualized environments as close to the production environment as possible, with more accurate composition of the target system and more scriptable behavioural monitoring. Additionally, it can also find out such sophisticated malware attempts to bury its activities within the virtual environment.

e) Quantum Computing and Its Impact on Malware Analysis

As quantum computing progresses to its reality as a technology, the same will be true for the cybersecurity aspects; how the malware will be analyzed will change drastically. Quantum algorithms for speed up the process of downloading and classifying malware, and those quantum algorithms could bring in new classes of malware that exploit quantum computing in interesting new ways. To do this, researchers will be looking into quantum-resistant cryptographic methods and also quantum computing vulnerabilities.

C. Conclusion

Dynamic malware analysis in present-day cybersecurity has demonstrated that system call tracing and API based monitoring are an indispensable tool. These techniques help analysts detect, classify, and respond to the majority of malware with deep insights about its behaviour. The strength in the dynamic malware analysis capabilities is getting stronger with the integration of machine learning, cross-platform analysis, and threat intelligence sharing. As the malware becomes more advanced and develops, it will be more and more adaptive and multilayered, hence, the dynamic analysis will have to potentially become more adaptive and comprehensive by the use of artificial intelligence, quantum computing, and next-generation sandboxes against the cybercriminals. Such spaces should be an investment for these organizations to develop a resilient strategy of defense predicated on defense against even sophisticated attacks.

IX. REFERENCES

[1] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behaviour," in Detection of Intrusions and Malware, and Vulnerability Assessment, Paris, France: Springer, 2008, pp. 161–182.

[2] S. Kirat, G. Vigna, and C. Kruegel, "BareCloud: Bare-metal analysis-based evasive malware detection," in 23rd USENIX Security Symposium, San Diego, CA, USA, 2014, pp. 287–301.

[3] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole EXE," in Proceedings of the AAAI Workshop on Artificial Intelligence for Cyber Security, 2018. [Online]. Available: https://arxiv.org/abs/1710.09435

[4] J. Saxe and K. Berlin, "Deep neural network-based malware detection using two-dimensional binary program features," in 10th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, PR, USA, 2015, pp. 11–20. [Online]. Available: https://arxiv.org/abs/1508.03096

[5] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in Australasian Joint Conference on Artificial Intelligence, Hobart, Australia: Springer, 2016, pp. 137–149.

[6] S. Mohurle and M. Patil, "A brief study of WannaCry threat: Ransomware attack 2017," International Journal of Advanced Research in Computer Science, vol. 8, no. 5, pp. 1938–1940, 2017.

[7] F. C. Freiling, T. Holz, and G. Wicherski, "Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks," in European Symposium on Research in Computer Security (ESORICS), 2005, pp. 319–335.

[8] U.S. Department of Justice, "Emotet botnet disrupted in international cyber operation," Jan. 2021. [Online]. Available: https://www.justice.gov/archives/opa/pr/emotet-botnet-disrupted-international-cyber-operation

[9] A. Austin and M. Stamp, "Static, dynamic, and hybrid analysis for malware detection," Journal of Computer Virology and Hacking Techniques, vol. 11, no. 2, pp. 95–105, 2015.

[10] Y. Xia, Y. Liu, J. Li, and H. Jin, "Evasion techniques: Future directions for malware analysis," IEEE Access, vol. 7, pp. 63664–63679, 2019.

[11] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting malware infection through IDS-driven dialog correlation," in USENIX Security Symposium, 2007.

[12] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," Journal in Computer Virology, vol. 7, no. 4, pp. 247–258, 2011.

[13] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "AccessMiner: Using system-centric models for malware detection," in ACM Conference on Computer and Communications Security (CCS), 2010, pp. 399–412.

[14] M. Christodorescu and S. Jha, "Testing malware detectors," ACM SIGSOFT Software Engineering Notes, vol. 29, no. 4, pp. 34–44, 2004.

[15] N. Idika and A. P. Mathur, "A survey of malware detection techniques," Purdue University, CERIAS Tech Report, 2007. [Online]. Available: https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/2007-26.pdf

[16] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2175–2216, 2017.

[17] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," Computers & Security, vol. 81, pp. 123–147, 2019.

[18] A. Mohaisen and O. Alrawi, "Unveiling Zeus: Automated classification of malware samples," in Proceedings of the 22nd International Conference on World Wide Web Companion, 2013, pp. 829–836.

[19] S. E. Schechter and M. D. Smith, "Accessing protected resources through API monitoring," IEEE Security & Privacy, vol. 1, no. 5, pp. 62–65, 2003.

[20] J. Caballero, P. Poosankam, C. Kreibich, and D. Song, "Dispatcher: Enabling active botnet infiltration using dynamic analysis," in RAID, 2009, pp. 41–60.

[21] P. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying deep learning approaches for network traffic prediction," in International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017.

[22] S. M. Fadhlullah and H. Hasbullah, "Review on dynamic malware analysis techniques," Australian Journal of Basic and Applied Sciences, vol. 5, no. 10, pp. 500–510, 2011. [Online]. Available: https://ajbasweb.com/old/ajbas/2011/October-2011/500-510.pdf

[23] R. Perdisci, A. Lanzi, and W. Lee, "Behavioral clustering of HTTP-based malware," in USENIX NSDI, 2010.

[24] **Dixit, S.** (2020). The impact of quantum supremacy on cryptography: Implications for secure financial transactions. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 6(4), 611–637. https://doi.org/10.32628/CSEIT2064141

[25] Yashu, F., Saqib, M., Malhotra, S., Mehta, D., Jangid, J., & Dixit, S. (2021). Thread mitigation in cloud native application development. Webology, 18(6), 10160–10161. https://www.webology.org/abstract.php?id=5338s