European Journal of
Technology
(EJT)

System-Level Behavior Analysis for Detecting Advanced Persistent Threats (APTs)

Khaja Kamaluddin

European Journal of Technology
ISSN 2520-0712 (online)
Vol.4, Issue 2, pp1 1 - 17, 2020

www.ajpojournals.org

System-Level Behavior Analysis for Detecting Advanced Persistent Threats (APTs)

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

Article history
Submitted 21.04.2020 Revised Version Received 22.05.2020 Accepted 24.06.2020

Abstract

Purpose: Advanced Persistent Threats pose a serious threat in cybersecurity because of their stealth, long presence, and ability to hide. Most organizations placed considerable emphasis on signature-based detection techniques, which were effective against known malware but often failed to detect novel, targeted, or user-specific threats with undefined signatures. This study investigates system-level behavioral analysis as a dynamic alternative for detecting APTs, shifting focus from static indicators to the real-time behavior of processes and applications interacting with the operating system. It emphasizes the importance of identifying abnormal activities such as atypical system call usage, unauthorized process creation, memory injection, and unpredictable modifications to the registry or file system.

Materials and Methods: The research outlines several practical tools and methods used to capture behavioral data, including system call monitoring with strace and Sysmon, process and memory analysis via Process Monitor and Volatility, and registry inspection with Autoruns and Rekall. While these techniques lack automation and often require significant technical expertise, they offer valuable insights into threats that evade conventional antivirus solutions. **Findings:** The study acknowledges the challenges posed by high false positives, manual rule creation, and scalability limitations but underscores their critical role in laying the groundwork for modern cybersecurity practices.

Unique Contribution to Theory, Practice and Policy: Based on these findings, the study recommends the integration of behavioral detection capabilities into advanced, automated platforms that leverage machine learning and cloud-based analytics. It advocates for a behavior-first approach that prioritizes system-wide visibility and proactive threat hunting over reactive, signature-matching strategies. These recommendations aim to inform the development of AI-driven security solutions capable of detecting complex, evasive threats like APTs in real time and at scale.

Keywords: *Advanced Persistent Threats (APTs) (O33); Behavioral analysis (D83); Memory analysis (C63); Registry activity (C88); Cybersecurity (O33, H56); Malware detection (O33); Threat intelligence (L86).*

INTRODUCTION

Advanced Persistent Threats (APTs) represent one of the most sophisticated and dangerous forms of cyberattacks, typically executed by well-funded and highly skilled adversaries such as nation-states or organized cybercrime groups [1]. Unlike conventional attacks that aim for immediate exploitation, APTs are characterized by their stealth and persistence, often maintaining unauthorized access to a target network for months or even years [2].

The approach represents a sequence of steps: including initial access, persistence, Privilege Escalation, Lateral Movement, Data Collection, and Exfiltration [3]. At each step, attackers utilize a mix of custom malware, fileless techniques, and legitimate tools like PowerShell, WMI, or RDP to blend into regular system activity and avoid triggering conventional defenses.

Detection methods that assume signatures such as antivirus programs, firewall policies and file hash databases are often not effective against these threats because they rely much on static indicators of compromise (IoCs). APTs are skilled at bypassing these systems by continually evolving their tactics and using "living-off-the-land" binaries (LOLBins) that do not exhibit obvious malicious properties [4]

In response to these stealth threats, security professionals have begun to rely more on behavior-based detection which looks at file activities rather than dwelling on static identifiers. Table 1 shows advantages and drawbacks of both methods.

Table 1: Signature-Based vs. Behavioral Detection

Aspect	Signature-Based Detection	Behavioral Detection
Detection Method	Matches known patterns (hashes, signatures)	Monitors system behavior (e.g., syscalls, processes)
Effectiveness Against New Threats	Poor against unknown or obfuscated malware	Effective against novel or fileless attacks
Evasion Resistance	Easily bypassed by polymorphic malware	Harder to evade due to focus on system behaviors
False Positives	Low, with good signature updates	Higher, due to potential benign anomalies
Detection Timing	Post-compromise (after payload is active)	Real-time or early in attack lifecycle
APT Detection Suitability	Limited, misses stealthy APTs	Strong, detects persistent and low-noise APTs

APT detection needs to shift from the high-level scope of symptoms to a lower-level analysis of system-level actions including system calls, process related interactions, memory manipulation, and operations on the registry and file system. Signals at this level expose far more accurate and reliable indications of compromise in the form of how the OS is being used irrespective of file name, hash, or any encryption [5].

Obfuscation of system level behavior is exponentially harder for an adversary as compared to obfuscation of static artifacts. Even in polymorphism or fileless , malware cannot escape interaction with the OS in pursuit of its objectives, for example: by creating hidden processes,

injecting code into the memory, editing registry runkeys or establishing unauthorized network sessions [6].

If a document viewer (winword.exe) creates and execute a command shell (cmd.exe), and writes out PowerShell scripts, this particular operation distinguishes itself as suspicious even though it does not match any signature of known malware.

Further, system-level analysis has the effect of integrating disparate data domains. Relating observed changes to memory, registry changes, and actions in a process to a common analysis report. Such a multifaceted point of view effectively contributes to identifying APTs, which could be masked by the background of regular system behaviors.

The result is that system-level behavioral analysis supports robust, future-oriented defense not only by uncovering underlying dangers but also by providing actionable findings to combat new cyber perils.

This research studies how at the system level behavioral analysis was exploited to uncover Advanced Persistent Threats (APTs). The core objectives are:

i. Analyze system-level behavioral techniques (system calls, process analysis, and memory forensics) for APT detection.
ii. Evaluate their efficacy against APT tactics (privilege escalation, fileless attacks) and limitations (false positives, scalability).
iii. Assess their relevance in modern defenses and integration with AI/threat intelligence.

APT Tactics and Threat Landscape

Anatomy of an APT Attack

APT activity is divided into several stages with insistence on covert activities, extended residence, and persistent goal attainment.

i. **Initial Access:** Target systems are often gained access to by adversaries who employ such techniques as spear phishing, taking advantage of vulnerable systems, or taking over official entry ways such as RDP or VPN.
ii. **Privilege Escalation:** Once attackers infiltrate a system, it is known that they usually exploit vulnerabilities, or misconfigurations in order to gain elevated privileges, such as those of an administrator or a root user.
iii. **Persistence:** In order to maintain their presence, attackers regularly utilize backdoors, manipulate registry entries, or implement malware which disables their removal once the system restarts. Rootkits, malicious scripts, or reverse engineering of scheduled tasks are methods which are widely utilised by attackers.
iv. **Lateral Movement:** Attackers roam the network further to find more intelligence to extract from more systems and to further their foothold. In order for this to be continuous, some satellite constellations are made up of several satellites that operate on different orbits, in order to ensure high availability.
v. **Exfiltration:** During this final stage, hackers steal and upload-sensitive information (proprietary data or personal records) to a remote server. This segment is usually performed in secret with the purpose of avoiding security leaving encrypted channels and transferring data in small, unobtrusive increments for undetected transfer.

Because of their stealth and persistence, APT attacks are difficult to identify in a real-time basis. In order to mask their activity, attackers commonly deploy rootkits, encrypt their activities to hide their flight, and use fileless malware which executes in memory to circumvent signature detection [7]. The processes through which APTs can hide while establishing a persistent presence in the intended system over time.

Notable APT Campaigns

Prominent APT events have been very effective on awareness and knowledge of current cyber threats. Some notable ones include:

i. **Stuxnet (2010):** Stuxnet targeted industrial control systems directly by exploiting weaknesses in Programmable Logic Controllers (PLC) and employed such to malicious purposes [8]. The attackers managed a deceptive point of entry using signed drivers, which allowed the malware to circumvent protective measures and negatively affect Iran's nuclear installations.

ii. **APT28 / Fancy Bear (2015):** As well as its other name APT28, you have APT28 being better known as Fancy Bear, and this group, frequently used spear phishing and credit card dumping tools such as Mimikatz to penetrate security and steal valuable data. Some of their activities included politically inspired espionage tendencies that were mostly targeting the government organizations [9].

iii. **APT29 / Cozy Bear (2015):** APT29/Magic worm was notorious of using fileless techniques and legitimate tools such as PowerShell and WMI to execute their attack. Their main areas of activity were cyber espionage, consistent attacks on diplomatic and governmental institutions [9].

iv. **Operation Aurora (2010):** Operation Aurora (2010) targeted major tech firms like Google using a zero-day flaw in Internet Explorer (CVE-2010-0249) to gain initial access. The attack involved memory injection into legitimate processes (explorer.exe), registry modifications for persistence, and encrypted outbound communication for data exfiltration. System-level behaviors included suspicious child processes, registry changes, and anomalous network activity, aligning with techniques like process injection and C2 over HTTPS.

Table 2: Summary of Major APT Campaigns

Name	Year	Attack Vector	Key Tools	Behavior Observed
Stuxnet	2010	USB/PLC	Rootkits	Registry edits, DLL injection
APT28	2015	Spear Phishing	Powershell, Mimikatz	Credential theft, persistence
APT29	2015	Fileless Attacks	PowerShell, WMI	Cyber espionage, stealthy execution
Operation Aurora	2010	Zero-day Exploit	Custom malware	Corporate espionage, targeted attacks

According to Table 2, the campaigns used certain tactics and tools, which are a useful source of information on the characteristics of APT attacks, and their ongoing evolution.

MITRE ATT&CK Framework

The MITRE ATT&CK [10] framework provides a comprehensive, structured repository of adversarial tactics, techniques, and procedures (TTPs), enabling security practitioners to analyze and categorize malicious behavior. In the context of system-level behavior analysis, ATT&CK is especially valuable for mapping low-level system events such as process creation, registry access, and API call sequences to higher-order adversary behavior.

Tactics define the adversary's objectives at each stage of an attack, from initial access to data exfiltration. System-level signals aligned with these tactics can include:

- **Initial Access:** Process execution from email clients (e.g., outlook.exe) spawning unusual child processes (e.g., cmd.exe or powershell.exe), often observed during spearphishing with attachment payloads.
- **Persistence:** Registry modifications (e.g., HKCU\Software\Microsoft\Windows\ CurrentVersion\Run) or scheduled task creation detected through monitoring API calls like RegSetValueEx or CreateService.
- **Privilege Escalation:** Use of token impersonation or DLL sideloading, detectable through anomalies in privilege tokens or DLL load paths.
- **Defense Evasion**: Fileless malware execution using PowerShell or WMI, which may bypass traditional file-based AV detection but leave traces in script execution logs or memory usage anomalies.
- **Exfiltration:** Data staging in temporary directories followed by encrypted outbound connections, observable via system call sequences and anomalous network API activity (e.g., WinInet, WinSock).

Techniques in ATT&CK describe the specific methods attackers use, which map directly to system-level observables. For example:

- **T1055:** Process Injection: Can be detected by monitoring Write Process Memory and Create Remote Thread API usage.
- **T1086:** PowerShell: Monitored through command-line auditing or script block logging.
- **T1027:** Obfuscated Files or Information: Linked with entropy-based detection of memory-resident payloads.

Procedures reflect how real-world threat actors implement these techniques. For instance:

- Mimikatz (associated with T1003 – OS Credential Dumping) uses direct memory reads to extract credential hashes from LSASS, which triggers suspicious Read Process Memory calls on lsass.exe.
- APT29 employs PowerShell and WMI to execute fileless malware, correlating to specific system behaviors such as frequent WmiPrvSE.exe activity coupled with dynamic code execution.

By incorporating ATT&CK into system-level behavior analysis, defenders can better contextualize low-level events within a broader threat model, improving detection of stealthy, technique-driven attacks even when traditional signature-based tools fail.

Importance for APT Detection:

Threat Intelligence: By linking observed attack trends to known adversary groups, such as APT28 or APT29, the framework gives relevant insight into the tactics and techniques used by adversary groups.f

i. **Detection & Response:** With the TTPs of ATT&CK available, defenders can be more empowered to build the detection frameworks, which simplifies their effort in identification of APTs through suspect actions.

ii. **Red and Blue Team Exercises:** In Red Team drills or Blue Team simulations ATT&CK provides a shared ground for assessment and improvement of the security procedures.

APT28 may abuse Mimikatz for Credential Dumping in the course of Privilege Escalation operations, and those in defense should monitor for unusual memory access or privilege escalation activities to be able to detect threats.

Use of the Mitre Attack Matrix in Table 3 is used to discuss the real world consequences of these tactics on the adversary actions.

Table 3: Mapping of APT Tactics, Techniques, and Tools Observed in System-Level Behavioral Analysis

Tactic	Technique	Tool
Privilege Escalation	Credential Dumping	Mimikatz
Exfiltration	Encrypted Transfer	PowerShell
Persistence	Registry Modification	Custom Backdoor

System-Level Behavioral Detection Techniques

System Call Monitoring

System calls serve as the primary interface between user-space applications and the operating system kernel [11]. Any application makes a system call when it needs to communicate with system resources, including the execution of file I/O, process initiation, and network communication. Encompassing among the list of APT attackers, malicious actors often use system calls in carrying out bad acts, which include installing malicious software, stealing data and gaining higher privilege.

System call monitoring is directed towards anomaly-based detection of potentially malicious activity from system call behavior. In doing so, this technique reveals unusual behavior that could be missed by traditional systems based on signatures or static analysis.

Tools

- **Strace (Linux):** Strace is a crucial diagnostic tool for Linux, in which, analysts can see system calls and signals in real-time. Strace watches an application level of system call for capturing anomalous behaviors by logging how processes interact with the operating system[12].
- **AuditD (Linux):** A part of Linux audit system, AuditD provides the possibility to monitor system events, for instance, system calls, to be safe. The ability of the AuditD to log for every system call allows detection of unusual activities from the kernel[13].

- **Sysmon (Windows):** Sysmon (System Monitor) from Sysinternals is a Windows tool that monitors process activity in depth, network connections, and changes to the file system. Thanks to its capability of monitoring Windows activity in detail, Sysmon proves to be a useful instrument in detecting indications of malicious behavior on a system level [14].

Detection Techniques

A number of techniques have been designed to identify suspicious things via tracking system call logs. The angle here is to identify the distortions or the abnormalities in the behaviour of the normal system calls which may indicate malicious work. Some notable detection approaches include:

- **Frequency-based Syscall Anomaly Detection:** Statistics on when system calls occur is also used to identify any anomalous activity. Any high increase in system calls that utilize either process sparing or network communication is a common occurrence during intrusions or malware activity. Persistence activities of malware are frequently accompanied by anomalous system call activity.
- **Sequence Modeling (Simple Pattern Matching):** Pattern matching was used by sequence modeling to detect suspicious syscall patterns before machine learning was adopted on a large scale. For example, if a row of system calls fits in line with proven attack strategies e.g., process injection (via syscalls like NtWriteVirtualMemory) it implies a possible intrusion attempt by an attacker. It is effective in detecting the predefined attack behaviors, but it remains largely ineffective in dealing with developing and intricate tactics.
- **Mapping System Call Patterns to Specific TTPs:** Specific system calls are often linked to particular tactics, techniques, and procedures (TTPs) used in advanced attacks. For instance, an attacker using process injection might trigger system calls like NtWriteVirtualMemory or CreateRemoteThread. By correlating system call patterns with documented TTPs (e.g., from the MITRE ATT&CK framework), security teams can more effectively detect malicious activity early in the attack chain.

These techniques have been evaluated in enterprise sandbox environments and academic settings using benchmark datasets such as DARPA, ADFA-LD [21], and custom Red Team simulations, enabling consistent validation of detection accuracy and practical effectiveness.

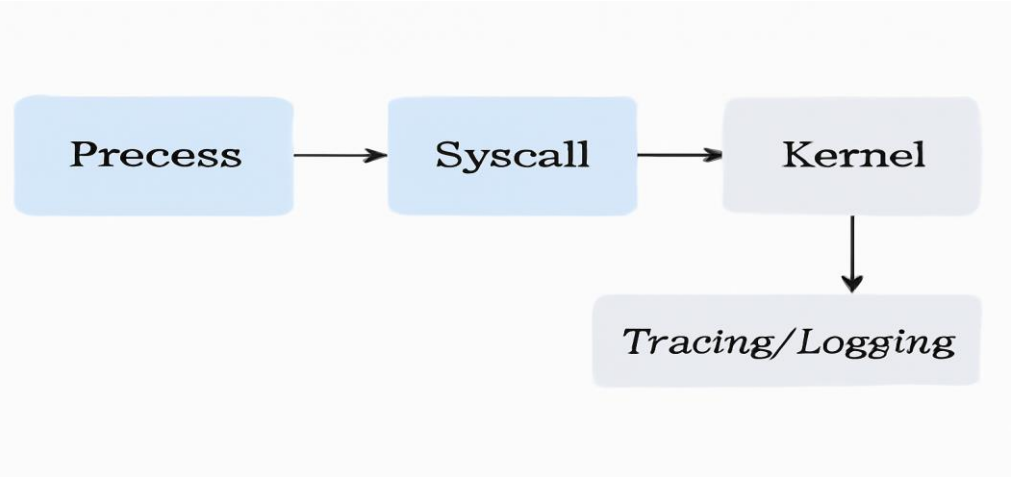

Figure 1: System Call Flow Diagram

Process Behavior Analysis

Process behavior analysis tries to identify questionable, or hostile behaviors emerging from processes in use in the system [15]. These tools provide the following detections to include: unauthorized process initiation, privilege level elevation, code injection and anomalous processes typical of Advanced Persistent Threats. Analyzing process interactions helps administrators to identify deviation from normal behavior and act against threats as it is happening.

Tools

It is possible to improve monitoring and analyzing the behavior of the process by using various tools built to monitor system activities and signal possible problems:

- **Process Monitor (Sysinternals):** This resource pays careful attention to monitoring the system in near real-time, noting down all the API calls made by the processes. It traces all system events (file system activity, registry changes, network activity and even process spawning). Monitoring these events enables Process Monitor to register abnormal process actions such as an attempt by a non-authorized process to access a system or cases where a process is compromised by an existing process.
- **pslist, pstree (Linux/Windows):**These utilities offer insight into currently running processes. pslist provides a flat list of processes on Windows, while pstree presents a hierarchical view in Linux, making it easier to identify suspicious parent-child relationships. For example, a command-line shell (e.g., cmd.exe) spawned by a document editor (e.g., word.exe) may signal process injection or abuse of trusted applications.
- **Volatility Framework:** Volatility acts as an advanced tool in memory forensics, useful in analysis once infected since it can analyze executing processes in system RAM. Volatility can show through its analysis the processes which are masked or injected, which can be in memory or hidden within other running applications, which might pass unnoticed.

Detection Features

Some of the major indications of potential dangerous process behavior security teams should be vigilant of are:

- **Parent-child Anomalies:** Attacker use trusted existing programs to run their malicious processes, thus making it difficult for security measures to detect them. If, from a word-processing utility, such as word.exe, a command line utility for example, cmd.exe is prompted it is a suspicious activity. Monitoring with Process Monitor and pslist can identify anomalies in the parent-child hierarchy that they monitor in real time by tracing their parent-child relationships.
- **Suspicious Process Names:** Sometimes, adversaries try to apply obfuscation methods in their effort to make malware appear to be trusted application. Attackers use deceptive process names that look like system or the real software to hide their thwarting activities. Using monitoring to define and verify legitimate names of processes can expose suspicious activities that indicate, a covert attack. Real-time observation using tools such as Process Monitor helps place these differences in plain sight.
- **Elevated Privileges:** APT attackers are commonly interested in privilege escalation for their malicious processes enabling them to have SYSTEM or root access. This ends up giving attackers complete power in tampering with the entire system in whatever they

want. Monitoring suspicious privilege escalation events is the essential step to determining the attempts of attackers to bypass security controls and gain more access. The technologies, such as Volatility are able to detect and trace privilege escalation actions that are hidden from more standard monitoring solutions.

- **Orphaned Processes:** An orphaned process is a process which no longer relies on the original parent process. It is a clear indication of possible intrusion if attack operators succeed to hijack legitimate procedures or to inject own code into other ones. Pstree and Volatility are used to find orphaned processes and this may indicate that the compromised the system.
- **Injected Threads or Hollowed Processes:** Code injection is a complex method that allows intruders to inject a malicious code into a real process in order to statically evade inspection. For instance, process hollowing is a process where an enemy uses legitimate process memory to be overwritten with malicious code. Through memory forensics, using like Volatility, it is possible to determine whether an injected thread or hollowed processes exist as violations in the memory structure of an active process will be recorded and raised.

Attackers often use process injection to gain elevated privileges or run malicious code within legitimate processes. Anomalies such as cmd.exe being spawned by word.exe can be detected using real-time tools like Process Monitor. If privilege escalation or hidden thread execution is suspected, forensic tools like pslist and Volatility help analyze process hierarchies and inspect memory for injected code. Combining real-time monitoring with post-mortem forensics enables layered threat detection. However, Volatility is offline and resource-intensive, while real-time tools may miss stealthy, memory-resident attacks without deeper analysis.

Registry and File System Monitoring (Windows Focused)

The goal of registry and file system monitoring is to detect persistence mechanisms and privilege abuse in Windows systems by identifying suspicious registry keys and files. Malicious actors often modify the Windows registry or manipulate files in specific directories to establish persistence and maintain access to compromised systems, even after a reboot or system restart [16]. By monitoring these areas, defenders can identify indicators of compromise (IoCs) and take preventive or corrective actions.

Detection Features:

- **Registry Keys:** To ensure that an unwanted program will automatically execute, malicious software updates registry entries associated with the system boot or user login events regularly. Registration keys that are often changed by the attackers such as:
- **HKCU\Software\Microsoft\Windows\CurrentVersion\Run:** This registry value contains records of programs that run automatically at user logon; hackers may use this facility to deploy and activate malicious files shortly after logon
- **Services:** Virtualization apps or hidden processes in Windows services can be executed by adversaries to enable malware to be initiated with administrative rights, when the machine is initialized.
- **Scheduled Tasks:** Hackers can install or modify the scheduled task to maintain their malware active and activate it to run at specific times.
- **File System Locations:** Attackers often store payloads or supporting files in less monitored directories to avoid detection. Notable locations include:

- **Temporary Directories (%TEMP %):** Frequently used for staging malware or dropping payloads due to limited oversight and automatic clearing on reboot.
- **AppData and Startup Folders**: Common persistence vectors, especially %AppData%\Roaming and %ProgramData%. Malware placed here is typically configured to run on login or boot.

Tools:

- **Sysinternals Autoruns:** This powerful tool from Sysinternals provides a comprehensive view of all auto-starting locations, including registry keys and file locations. Autoruns helps security teams detect unwanted startup items or persistence mechanisms that attackers may have left behind.
- **Windows Built-in Auditing Policies (enabled via GPO):** Windows Group Policy Objects (GPOs) can be configured to enable auditing of specific registry changes, process creation events, and file system modifications. This allows administrators to detect when malicious actors attempt to modify registry entries or files in sensitive locations.
- **Tripwire:** Tripwire is a filesystem integrity monitoring tool that can detect unauthorized changes to critical files or directories. It can be configured to alert on changes to files in specific locations (e.g., Temp directories or StartUp folders).

Tools for Monitoring and Detection

- **Unusual Timestamps:** Malicious files or registry entries often have suspicious timestamps, such as files with creation or modification times that don't align with normal system behavior.
- **Hidden Files:** Some malware attempts to hide files using attributes that prevent them from being easily detected by users or administrators.
- **Script Execution:** Monitoring for unusual script execution patterns (e.g., PowerShell or batch scripts) in specific directories can help identify malware that uses scripts to persist and execute on the system.

Memory Analysis and Forensics

Memory analysis and forensics aim to detect hidden or reflective malware running in volatile memory. Many sophisticated attacks, such as those involving advanced malware, often avoid detection by running entirely in memory or by using code injection techniques [17]. Analyzing system memory can reveal evidence of malware that has not been written to disk or that uses non-standard techniques to avoid detection by traditional file-based security tools.

Detection Features:

- **Code Injection Detection:** The widely used method of code injection detection is introducing malicious code into the memory of programs that are otherwise functioning normally. Security technologies may find it more difficult to detect the virus because the injected code may be functioning inside the framework of trustworthy, genuine processes. Since they don't match any disk-backed files, non-image-backed memory regions that could be indicators of injected code can be found using memory analysis tools.
- **Process Hollowing:** The act of an attacker establishing a suspended process and then introducing malicious code into its memory is known as "process hollowing." The malicious code executes in lieu of the legal code of the target process. By detecting

differences between a process's memory structure and the associated disk-backed image, memory analysis can reveal potential process hollowing attacks.

- **Kernel Object Discovery:** The technique of searching for odd structures or dubious kernel objects that might indicate background rootkits or kernel-level malware is known as kernel object detection.

By using these tools and techniques, memory forensics can reveal the presence of advanced persistent threats (APTs) and other sophisticated attacks that would otherwise be difficult to detect with traditional security methods.

Tools:

- **Volatility Framework:** Volatility is a well-established memory forensics tool that offers a variety of plugins for memory analysis. Volatility can scan memory dumps to detect processes, DLLs, injected code, and other signs of memory-based attacks, such as rootkits or reflective malware. Common plugins include those for scanning processes, listing DLLs, and detecting injected code in memory.
- **Rekall:** Google's Rekall is another powerful memory analysis tool. It's used to analyze volatile memory dumps and identify potential memory-based attacks. Rekall's features include memory forensic analysis for suspicious activity like as code injection or memory-based rootkits, process scanning, and kernel object identification.
- **Redline (FireEye):** Redline is a comprehensive host investigation tool developed by FireEye that enables analysts to collect and examine memory and disk artifacts. It supports acquisition of volatile memory, timeline analysis, malware detection, and indicator-based hunting. Redline can detect in-memory threats such as reflective DLL injection, unauthorized process hollowing, and registry tampering. It also includes integrity verification of key system files and supports IOC scanning, making it effective for both proactive and reactive incident response.

Rule-Based and Signature-Less Behavior Detection

The use of machine learning (ML) in security tools, rule-based and signature-less behavior detection was greatly aided by heuristics and bespoke rule engines. Unlike classic signature-based detection, which relies on known attack patterns (such as file hashes), behavior-based detection uses dynamic analysis to identify anomalous or suspicious activities [18]. This technique enables the identification of novel or unknown threats by relying on their behavior instead of their distinctive signature.

Pre-established rules are used in behavioral detection to find patterns of activity that deviate from normal system or network behavior. These rules may be based on a variety of factors, including unexpected traffic quantities, odd file system alterations, or the application of well-known attack methodologies. Security systems can identify potential risks without prior knowledge of the assault thanks to real-time system behavior monitoring and analysis.

Tools & Frameworks

Snort (Behavioral Rules)

Snort is a widely used open-source intrusion detection system (IDS) that allows for the creation of custom rules to detect patterns of behavior in network traffic. Users can build rules to identify anomalous network activity, such as surges in traffic volume or suspicious connection patterns. Because Snort allows for the creation of rules based on a variety of criteria, including source and destination IP addresses, protocols, and payload content, it is a helpful tool for

behavioral detection. One example of a behavioral rule in Snort is the detection of unusually high network traffic, which could indicate a Distributed Denial-of-Service (DDoS) assault.

Bro/Zeek IDS

Bro, now known as Zeek, is another powerful network monitoring tool that includes a behavioral scripting engine. It enables users to define custom scripts to analyze network traffic and detect unusual behaviors [19]. Zeek's ability to examine process context from logs allows it to spot odd network behavior, such as dubious file transfers or unauthorized access attempts. In large company settings, Zeek is widely utilized for comprehensive network traffic analysis and threat detection.

Symantec, McAfee AV Heuristics

Before machine learning was included into antivirus software, Symantec and McAfee used behavior-based scoring models to detect malware. These models assessed both runtime behavior (such as the actions a process performs while it is executing) and static features (such as file characteristics). The heuristic analysis would identify behaviors that might be indicative of malware, such as efforts to change system files or create persistent registry entries. These heuristic methods were crucial for locating malware versions without a recognized signature yet exhibiting dangerous behavior patterns.

Behavior-based detection systems often rely on predefined rules to trigger alerts when certain suspicious behaviors are observed. Some common rules include: Hackers usually change registry keys to remain persistent on a compromised system. Malicious behavior, such as malware generating new auto-start registry entries, may be indicated by a rule that rapidly detects several registry alterations.

Many attackers attempt to establish persistence during periods of low system activity, such as weekends or off-peak hours. A rule that monitors for any strange persistence efforts at these times (e.g., registry changes or freshly scheduled jobs) could help detect stealthy assaults.

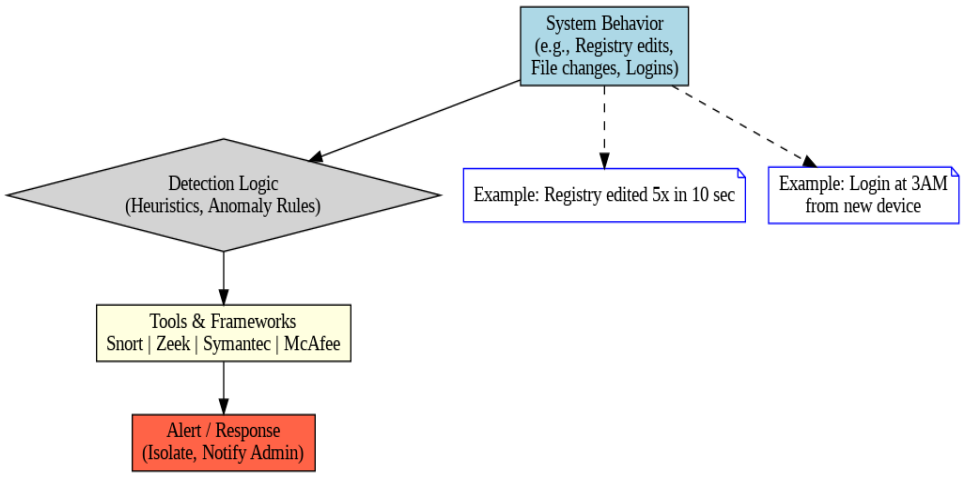

Figure 2: Sample Snort Behavioral Rule Structure

This figure shows an example of a Snort rule structure for behavioral detection. Among other patterns of behavior in network traffic, the rule is designed to identify anomalous connections or unexpected traffic quantities. It describes a number of prerequisites and actions to identify threats and determine how to respond to them. The image shows how specific behaviors that can indicate an ongoing incursion or attack can be identified using custom rules.

Table 4: Behavioral Indicators vs Static Signatures

Behavioral Indicators	Static Signatures
Detects anomalies in system/network activity	Detects known threats via predefined patterns (e.g., file hashes)
Monitors for unusual patterns (e.g., failed logins, process behavior)	Compares to known attack signatures
Examples: Failed logins, abnormal registry changes	Examples: Known malware hashes, specific attack patterns
Detects new and evolving threats	Limited to known threats
Identifies novel or zero-day attacks	Accurate for known threats, fast detection
Possible false positives, resource-heavy	Can't detect new or polymorphic threats

Behavioral indicators, as shown in Table 4, focus on spotting anomalies in system and network activity, whereas static signatures depend on known patterns (such file hashes). Behavioral signs can include weird process execution patterns, unexpected registry persistence, or multiple unsuccessful login attempts. Conversely, static signatures use preset criteria, such as a specific file hash or known malware signature, to identify dangers. The table highlights the distinctions between these two approaches as well as the advantage of behavioral detection in identifying emerging or novel risks.

Because behavior-based detection has a significant advantage over signature-based techniques and can detect novel assaults that have never been observed before, it is an essential component of modern security systems.

Limitations and Challenges Techniques

In addition to static signatures, the introduction of system-level behavioral detection techniques provided a crucial foundation for identifying malicious activity; nonetheless, these tactics had several significant disadvantages. Most tools and techniques at the time operated in silos, relied heavily on human configuration, and lacked the intelligence and scalability required for enterprise-level security.

Manual Configuration Overhead

A significant amount of manual setup was required for behavioral detection technologies. Security analysts have to develop, test, and refine detection criteria based on threat intelligence and system expertise. This method takes a great deal of experience and constant attention to evolving threat behaviors. For example, rule sets in tools like Snort or Zeek needed to be updated often to stay effective, and context was often needed for alert interpretation that was difficult to find in isolated logs or host-based outputs.

Lack of Real-Time Correlation

Another major barrier was the absence of real-time correlation between data sources. A centralized system for gathering and analyzing network and endpoint events was absent from most host-based behavioral tools, such as Process Monitor or AuditD, which operated independently. As a result, their ability to identify coordinated or dispersed attacks that impacted several systems was diminished. In the lack of integrated intelligence or correlation

engines, defenders were often reactive, responding only after indications of compromise had already materialized.

False Positives

Heuristic-based detection methods, although helpful in flagging potentially malicious activity, were notorious for generating high false positive rates. Because these systems relied on predefined rules and behavioral patterns, they often flagged benign actions as threats. For instance, a legitimate administrator running PowerShell scripts for maintenance could be misclassified as malicious activity [20]. Over time, such frequent false positives contributed to alert fatigue, where security personnel either ignored alerts or struggled to differentiate real threats from noise.

Limited ML Adoption

Perhaps the most significant technical limitation was the lack of machine learning (ML) integration. Behavioral detection systems did not leverage deep learning or unsupervised learning due to the scarcity of labeled behavioral datasets and the computational limitations of the time. While basic heuristics were in use, there was no intelligent model capable of adapting to emerging threats or generalizing across varied environments. As a result, detection systems lacked adaptability and often failed to recognize novel or obfuscated attacks.

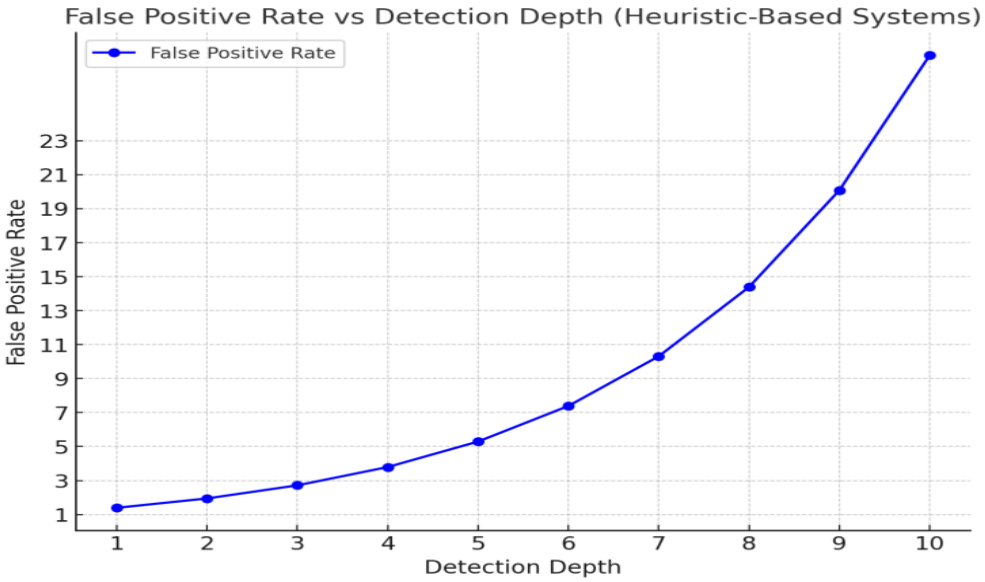

Figure 3: False Positive Rate vs. Detection Depth (Heuristic-Based Systems)

Figure 3 illustrates the trade-off between false positive rates and detection depth in heuristic-based systems. As detection methods attempt to monitor more complex system interactions and apply stronger heuristics in an attempt to probe deeper into behavior, the rate of false positives tends to increase dramatically.

This graph illustrates a fundamental flaw in early heuristic systems: while they may identify more subtle threats at higher inspection levels, they are more likely to identify acceptable activities as well, which reduces the system's practicality.

This study significantly advances cybersecurity by shifting the focus from static signature-based detection to dynamic system-level behavioral analysis, offering critical implications for theory, practice, and policy. Theoretically, it reinforces behavioral anomaly detection as a foundational principle, demonstrating that even stealthy APTs leave detectable traces through

system interactions such as process behavior and memory manipulation. By aligning these behaviors with frameworks like MITRE ATT&CK, the research enhances threat modeling and paves the way for AI-driven detection. Practically, it provides actionable guidance for using tools like Sysmon, Volatility, and Process Monitor to identify evasive techniques such as process hollowing and LOLBin abuse, promoting behavior-first detection strategies within SOCs. From a policy perspective, the study supports regulatory frameworks like NIST and ISO 27001 by advocating continuous monitoring, proactive threat hunting, and investments in behavioral EDR/XDR solutions. It also highlights the need for skilled analysts, influencing workforce development and cybersecurity training programs.

CONCLUSION AND RECOMMENDATIONS

Conclusion

System-level behavioral detection methods provided security teams with a powerful approach to identifying signs of Advanced Persistent Threats (APTs) that frequently bypassed traditional signature-based tools. By concentrating on how processes and applications interacted with the operating system, these techniques enabled defenders to spot unusual patterns and uncover hidden attacks.

Key methods involved monitoring system calls, analyzing parent-child process relationships, detecting unauthorized file or registry modifications, and inspecting memory for signs of code injection or process hollowing. Tools such as Sysmon, Process Monitor, Volatility, and Snort played a crucial role in capturing and interpreting this low-level activity. Although these tools lacked automation and real-time correlation and often required significant technical expertise, they offered critical visibility into threats that conventional antivirus solutions missed.

However, the approach had its limitations. It relied heavily on manually defined rules, frequent adjustments, and suffered from high false positive rates. Additionally, most tools operated independently without centralized aggregation or correlation, making it difficult to construct a unified, real-time view of attacks especially in large enterprise environments.

Recommendations

Despite these challenges, system-level behavioral analysis became a foundational element in the development of modern detection technologies. The experience gained from these early techniques demonstrated the value of behavior-based monitoring over static signatures and directly informed the emergence of automated detection systems using machine learning, behavioral scoring, and integrated threat intelligence. These efforts ultimately shaped the industry's shift toward more intelligent and adaptive cyber defense solutions by proving that understanding behavior at the syscall, process, memory, and file levels is essential to detecting advanced, evasive threats.

REFERENCES

[1] A. J. C. Lima, Advanced Persistent Threats, M.S. thesis, Univ. de Lisboa, Portugal, 2015.

[2] S. Singh et al., "A comprehensive study on APT attacks and countermeasures for future networks and communications: challenges and solutions," J. Supercomputer., vol. 75, pp. 4543–4574, 2019.

[3] P. Bhatt, E. T. Yano, and P. Gustavsson, "Towards a framework to detect multi-stage advanced persistent threats attacks," in Proc. 2014 IEEE 8th Int. Symp. Service Oriented System Engineering, 2014.

[4] B.I.T.S. Forensics, "SANS Institute," 2019.

[5] J. Samuel et al., "Survivable key compromise in software update systems," in Proc. 17th ACM Conf. Computer and Communications Security, 2010.

[6] F. Scrinzi, Behavioral Analysis of Obfuscated Code, M.S. thesis, Univ. of Twente, 2015.

[7] M. Ussath et al., "Advanced persistent threats: Behind the scenes," in Proc. 2016 Annu. Conf. Information Science and Systems (CISS), 2016.

[8] A. Matrosov et al., "Stuxnet under the microscope," ESET LLC, vol. 6, pp. 23, Sept. 2010.

[9] H. Mwiki et al., "Analysis and triage of advanced hacking groups targeting western countries critical national infrastructure: Apt28, Red October, and Regin," in Critical Infrastructure Security and Resilience: Theories, Methods, Tools and Technologies, 2019, pp. 221–244.

[10] B. E. Strom et al., "MITRE ATT&CK: Design and philosophy," Tech. Rep., the MITRE Corporation, 2018.

[11] R. Nikolaev and G. Back, "VirtuOS: An operating system with kernel virtualization," in Proc. 24th ACM Symp. Operating Systems Principles, 2013.

[12] G. Damri and D. Vidyarthi, "Automatic dynamic malware analysis techniques for Linux environment," in Proc. 2016 3rd Int. Conf. Computing for Sustainable Global Development (INDIACom), 2016.

[13] S. Miclea, "Windows and Linux security audit," J. Appl. Bus. Inf. Syst., vol. 3, no. 4, pp. 117, 2012.

[14] F. Nilsson et al., "SysMon–A framework for monitoring and measuring real-time properties," 2012. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:535850

[15] C. M. Anderson and D. Kincaid, "Applying behavior analysis to school violence and discipline problems: Schoolwide positive behavior support," The Behavior Analyst, vol. 28, pp. 49–63, 2005.

[16] F. Apap et al., "Detecting malicious software by monitoring anomalous Windows registry accesses," in Proc. Recent Advances in Intrusion Detection (RAID 2002), Zurich, Switzerland, Oct. 2002.

[17] A. Case and G. G. Richard III, "Memory forensics: The path forward," Digital Investigation, vol. 20, pp. 23–33, 2017.

[18] T. Liggett, Evolution of Endpoint Detection and Response Platforms, M.S. thesis, Utica College, 2018.

[19] M. Holkovič, O. Ryšavý, and J. Dudek, "Automating network security analysis at packet-level by using rule-based engine," in Proc. 6th Conf. Engineering of Computer Based Systems, 2019.

[20] W. Forstmeier, E.-J. Wagenmakers, and T. H. Parker, "Detecting and avoiding likely false-positive findings–a practical guide," Biol. Rev., vol. 92, no. 4, pp. 1941–1968, 2017.

[21] M. Xie, J. Hu, and J. Slay, "Evaluating host-based anomaly detection systems: Application of the one-class SVM algorithm to ADFA-LD," in *Proc. 11th Int. Conf. Fuzzy Systems and Knowledge Discovery (FSKD)*, Xiamen, China, Aug. 2014, pp. 978–982. doi: [10.1109/FSKD.2014.6980965]

License