Hijacking Processes for Privilege Escalation: Advanced Techniques and Defensive Countermeasures

I. Introduction to Privilege Escalation and Process Hijacking

Defining Privilege Escalation: Vertical vs. Horizontal

Privilege escalation represents a critical phase within the cyberattack lifecycle, where threat actors, having gained initial unauthorized access to a system, seek to increase their access rights. This elevated privilege is fundamental for executing more impactful malicious activities, such as data exfiltration, malware deployment, or achieving complete system control.[1]

Two primary types of privilege escalation are commonly observed:

- **Vertical Privilege Escalation:** This involves an attacker moving from a lower-privileged account, such as a standard user, to a higher-privileged one, such as a superuser, administrator, or the SYSTEM account. The ultimate objective is to gain unrestricted control over the entire system, a feat often achieved by exploiting inherent system vulnerabilities, software flaws, or misconfigurations.[1] With such elevated access, an attacker can modify system configurations, install software, create new privileged user accounts, or even delete critical data.[1]
- **Horizontal Privilege Escalation:** In contrast, horizontal escalation occurs when an attacker gains access to another user's identity or resources at the *same* privilege level. While this does not directly increase their own system privileges, it enables crucial lateral movement within a network, access to sensitive peer-level information, or the ability to obscure their tracks by operating under a different legitimate identity. Tactics often include credential theft or session hijacking.[1]

The consistent portrayal of privilege escalation as a "step in the cyber attack chain" [1] and a "means to an end" [3] highlights its strategic importance. It is not merely a final objective but a crucial pivot point that unlocks broader and deeper malicious activities within a compromised environment. For instance, successfully escalating privileges can transform simple malware infections into catastrophic data breaches and network intrusions.[1] This implies that defensive strategies must extend beyond merely preventing initial access to actively detecting and containing the *subsequent* activities that elevated privileges facilitate.

Understanding Process Hijacking: A Key Enabler for Privilege Escalation

In the realm of offensive cybersecurity, process hijacking refers to a technique where an attacker seizes control of an existing, legitimate running process to execute arbitrary malicious code or alter its intended behavior.[4] While the NIST glossary defines "process hijacking" more broadly in the context of checkpointing and migration for distributed resource management [4], its malicious application is centered on manipulating system processes for unauthorized access and code execution. This technique is a specialized form of process injection, which involves running custom code within the address space of another process.[5]

The primary advantage of process hijacking, and indeed process injection generally, is its ability to camouflage malicious activity. By operating within the context of a trusted application, the injected code becomes significantly harder for conventional security solutions to detect.[5] This inherent stealth is a major draw for adversaries. Furthermore, by hijacking a process, the malicious code inherits the privileges of the compromised process, directly enabling privilege escalation.[7] The dual benefit of gaining higher privileges while maintaining a low profile makes process hijacking a highly attractive technique for adversaries. This emphasis on stealth highlights a critical trend: as traditional signature-based and file-system-centric defenses improve, attackers increasingly shift towards in-memory, fileless, and legitimate-process-bound techniques. This evolution necessitates a corresponding shift in defensive measures towards behavioral anomalies and memory forensics, as traditional detection methods are often insufficient.

Importance in the Cyberattack Lifecycle

Privilege escalation is consistently identified as "one of the most exploited stages in the cyberattack lifecycle".[2] Its position in the attack chain is typically after initial access—which might be gained through methods like phishing, exploiting external vulnerabilities, or stolen credentials—and before the final "actions on objectives," such as data exfiltration, system destruction, or establishing long-term persistence.[3]

Within established frameworks like the Cyber Kill Chain, privilege escalation commonly falls within the "Exploitation" or "Installation" phases.[11] In these stages, attackers leverage identified vulnerabilities to gain higher access levels or to install persistent backdoors and command-and-control mechanisms. A successful privilege escalation can dramatically amplify the impact of an initial compromise, transforming what might have been a simple malware infection into a catastrophic data breach or a full network takeover.[2] The consistent description of privilege escalation as a pivotal transition point in the attack chain means that it acts as a critical bottleneck for adversaries. If an attacker fails to escalate privileges, their initial foothold may be severely limited in its potential for damage.[13] This implies that successfully preventing or detecting privilege escalation can act as a critical choke point, significantly disrupting the entire attack progression. This understanding is crucial for incident response and threat hunting efforts, as focusing detection and response at this stage can prevent attackers from achieving their ultimate, more destructive objectives, even if initial access was not prevented.

II. Core Process Hijacking Techniques

A. Process Injection

General Principles and Benefits (Privilege Escalation, Defense Evasion)

Process injection is a broad category of techniques designed to execute arbitrary code within the address space of an existing, legitimate, and typically non-malicious process.[7] This method is widely favored by adversaries due to its effectiveness in evading security controls and enabling privilege escalation.[7]

The primary benefits derived from process injection are significant:

- **Privilege Escalation:** When malicious code is injected into a process that operates with elevated privileges, the injected code automatically inherits those higher permissions. This grants the attacker greater control over the compromised system, allowing them to perform actions that would otherwise be restricted, such as downloading subsequent payloads, exfiltrating sensitive data, or spreading malware across the network.[7]
- **Defense Evasion:** Executing malicious code within a legitimate process allows attackers to camouflage their activities. This makes it exceedingly difficult for traditional security tools, including signature-based antivirus and disk forensics, to detect the threat, as the malicious code is often memory-resident and operates under a trusted process identity that is typically allow-listed.[7] This technique enables malicious payloads to run unnoticed, bypassing security mechanisms designed to flag new or suspicious executables.[5]

The fundamental technical steps involved in most process injection methods typically include: acquiring a handle to the target process, allocating memory within that process's address space, writing the malicious code (often shellcode or a Dynamic Link Library (DLL)) into the allocated memory, and finally, triggering its execution.[8] Common target processes, often chosen for their high privileges or frequent execution, include explorer.exe, rundll32.exe, svchost.exe, and various browser processes.[7] The consistent emphasis on process injection operating "in memory without leaving any traces of a malicious executable file on disk" [15] highlights a critical evolution in the threat landscape: attackers are increasingly favoring memory-resident and fileless techniques. This is a direct response to the growing effectiveness of disk-based forensic tools and signature-based antivirus solutions, necessitating a shift in defensive strategies from static file analysis to dynamic behavioral analysis, memory forensics, and advanced Endpoint Detection and Response (EDR) capabilities.

Thread Execution Hijacking (T1055.003): Detailed Steps and Real-World Examples

Thread Execution Hijacking is a specific process injection technique where an attacker takes control of an *already existing* thread within a separate, running process and redirects its execution flow to a malicious payload. This method is particularly stealthy as it avoids creating new processes or threads that might trigger security alerts, instead manipulating an active component of an existing process.[5]

The general attack lifecycle for Thread Execution Hijacking involves several precise steps:

1. **Process Handle Acquisition:** The attacker's first step is to obtain a handle to the target process into which they intend to inject code. This is achieved by using the OpenProcess API function, specifying necessary access rights such as PROCESS_VM_OPERATION, PROCESS_VM_WRITE, and PROCESS_VM_READ.[5]
2. **Thread Suspension:** Once the process handle is obtained, a specific thread within that target process is identified. A handle to this thread is then acquired via the OpenThread API, and the thread is temporarily suspended using SuspendThread. This suspension is crucial to prevent the thread from executing further instructions while the injection is in progress.[5]
3. **Memory Allocation:** After successfully suspending the thread, the attacker allocates a region of executable and writable memory within the target process's virtual address space. This is typically accomplished using VirtualAllocEx, specifying MEM_COMMIT and PAGE_EXECUTE_READWRITE flags to ensure the allocated memory is suitable for code execution.[5]
4. **Writing Shellcode:** The attacker's malicious payload, often referred to as shellcode, is then written into this newly allocated memory space within the target process. The WriteProcessMemory function is used for this, copying data from the attacker's buffer to the target's process memory.[5]
5. **Hijacking Thread Context:** The execution context of the suspended thread is retrieved using GetThreadContext. This context includes the values of CPU registers. The instruction pointer (EIP for x86 architectures, RIP for x86-64 architectures) within this context is then modified to point to the starting address of the injected shellcode.[5]
6. **Context Manipulation:** The altered execution context is applied back to the suspended thread using SetThreadContext. This action effectively changes the thread's next instruction to the injected malicious code, redirecting its flow.[5]
7. **Thread Resumption:** Finally, the ResumeThread function is called to resume the thread. The thread then continues its execution from the new entry point specified by the altered EIP/RIP register, thereby executing the attacker's malicious code within the context of the legitimate process.[5]

This technique is commonly observed in the wild. For instance, Lumma Stealer, in January 2024, was reported to use thread execution hijacking. This malware performs anti-VM and anti-debug checks before injecting its decrypted payload into target processes via the SuspendThread function.[5] Zloader malware also utilized thread execution hijacking to inject CyberMesh.exe into msiexec.exe.[5] The explicit enumeration of specific Windows API functions like OpenProcess, SuspendThread, VirtualAllocEx, WriteProcessMemory, GetThreadContext, SetThreadContext, and ResumeThread in the procedural steps for Thread Execution Hijacking provides a unique behavioral signature of the attack. While individual API calls might be legitimate, their specific sequence and combination, particularly when invoked against an external process, are highly suspicious. This directly informs defensive strategies: security monitoring tools (such as EDR and Sysmon) can be configured to log or alert on the suspicious sequence or combination of these API calls, even if individual calls are legitimate, thereby moving detection from signature-based to behavior-based.

VDSO Hijacking (T1055.014): Linux-Specific Manipulation of Virtual Dynamic Shared Objects

VDSO Hijacking is a sophisticated process injection technique unique to Linux-based systems. It allows adversaries to execute malicious code by manipulating the Virtual Dynamic Shared Object (VDSO). The VDSO is a special shared library that the Linux kernel dynamically links into the address space of each user-space application upon execution. It contains a small number of frequently used functions, such as time-related functions, which optimize system calls by allowing processes to execute code stubs directly from their own memory without the overhead of a full kernel call.[18]

By hijacking this optimized mechanism, attackers can inject and execute malicious payloads without relying on traditional process creation methods, making detection significantly more challenging due to its low-level, in-memory nature.[18] The fact that VDSO Hijacking is explicitly described as "Linux-based" and leverages a Linux-specific "optimization" for system calls highlights that sophisticated privilege escalation techniques are often deeply intertwined with the unique architectural design and performance optimizations of a particular operating system. What is designed for efficiency can, if not rigorously secured, become a vulnerability. This reinforces the critical need for operating system-specific security expertise and tailored detection/mitigation strategies, as a generic, cross-platform security approach is likely to miss these highly specialized, OS-native exploitation techniques.

Adversaries can perform VDSO hijacking through two main methods:

1. **Patching Memory Address References:** In this method, an adversary patches the memory address references stored in the process's Global Offset Table (GOT). The GOT is a data structure used by dynamic linkers to resolve symbols (e.g., functions and variables) in dynamically linked libraries. By replacing the legitimate address of a function in the GOT with the address of a malicious function, the attacker redirects the process's execution flow when that function is called, allowing arbitrary code execution within the compromised process.[18]

2. **Overwriting the VDSO Page:** This method involves exploiting memory corruption vulnerabilities (e.g., buffer overflows or use-after-free flaws) to directly overwrite the VDSO page itself with malicious code. Once the VDSO page is corrupted with the attacker's payload, any subsequent VDSO system call made by the compromised process will inadvertently execute the attacker's injected code.[18]

Process Hollowing (T1055.012): Mechanism, Steps, and Notable Malware Examples

Process Hollowing is a highly stealthy malware injection technique designed primarily for evasion. It involves an attacker creating a legitimate process in a *suspended* state, then stripping out (or "hollowing out") its original code from memory. This cleared space is then replaced with malicious code, and the process is subsequently resumed.[15]

Attackers favor this technique for several compelling reasons:

- **Evasion of Security Tools:** Because the malicious code runs under the identity of a legitimate process (e.g., explorer.exe or svchost.exe), it appears benign to many traditional antivirus and behavioral detection tools, often bypassing them entirely.[15]
- **Code Execution Without Dropping a File:** The attack primarily operates in memory, leaving minimal forensic artifacts on disk. This "fileless" nature significantly complicates post-incident analysis and makes detection challenging for solutions reliant on disk-based signatures.[15]
- **Persistence & Privilege Escalation:** If the legitimate process chosen for hollowing has elevated privileges (e.g., a system service), the injected malicious code inherits these permissions, dramatically increasing the attack's potential impact and facilitating persistence.[15]

The step-by-step mechanism of Process Hollowing is as follows:

1. **Create a Suspended Process:** The attacker initiates a legitimate process (e.g., explorer.exe, svchost.exe) using APIs like CreateProcess with the CREATE_SUSPENDED flag. This critical step prevents the original code from executing immediately upon creation.[15]
2. **Unmap the Original Code:** The memory region where the legitimate executable's code is loaded is then removed or "hollowed out." This is achieved using system calls such as ZwUnmapViewOfSection or NtUnmapViewOfSection, which effectively clear the space within the process's memory for the malicious payload.[15]
3. **Inject Malicious Code:** New memory is allocated within the compromised process using VirtualAllocEx. The malicious payload (shellcode) is then written into this newly allocated space via WriteProcessMemory.[15]
4. **Modify Execution Context:** The entry point of the original process's primary thread is altered to redirect execution to the newly injected malicious code. This is typically achieved by updating the thread context using SetThreadContext, which modifies the instruction pointer (EIP/RIP).[15]
5. **Resume Execution:** Finally, the attacker resumes the suspended process using ResumeThread. The process then continues execution from the new entry point, causing

the malicious payload to execute under the guise of the legitimate process.[15]

Process Hollowing is a common technique in advanced persistent threats (APTs), ransomware, and fileless malware. Notable real-world examples include Emotet, TrickBot, Ryuk Ransomware, and Cobalt Strike, all of which have leveraged this technique to evade detection and achieve their objectives.[15] REMCOS RAT was also reported to use process hollowing to copy itself into iexplore.exe.[20] The consistent emphasis on Process Hollowing's use for "Evasion of Security Tools" and "Code Execution Without Dropping a File" [15], along with its ability to bypass "IDS/IPS systems, firewalls, etc." [19], points to a clear evolutionary trend in malware development: a shift away from easily detectable disk-based artifacts towards in-memory, fileless execution. This indicates that traditional endpoint security solutions relying heavily on file-based signatures are becoming less effective against modern threats, driving the imperative for organizations to adopt advanced behavioral analytics, memory forensics, and EDR solutions that can detect anomalous process behavior and in-memory code execution.

B. Dynamic Link Library (DLL) Manipulation

DLL Injection: Types and Escalation Pathways

DLL injection is a technique that involves forcing a running process to load a malicious Dynamic Link Library (DLL) into its address space. This manipulation enables the attacker to execute arbitrary code, bypass security mechanisms, achieve privilege escalation, or steal sensitive data within the context of the targeted process.[21] The core of the attack lies in manipulating the legitimate DLL loading process, either by replacing, supplementing, or tricking the application into loading a malicious DLL instead of or alongside a legitimate one.[21]

Common types of DLL injection include:

- **DLL Hijacking:** This is a prevalent form where a legitimate DLL is replaced with a malicious one, often by exploiting vulnerabilities in the application's DLL loading mechanism or search order.[21]
- **DLL Proxying:** An attacker creates a malicious DLL that acts as an intermediary (proxy) between the legitimate DLL and the application. This allows the attacker to intercept, modify, or manipulate sensitive data and function calls without disrupting the application's normal operation.[21]
- **AppInit_DLLs:** This method leverages a specific Windows registry key (AppInit_DLLs) that instructs the operating system to load specified DLLs into *every* process that uses the User32.dll library. By adding a malicious DLL to this key, an attacker can achieve widespread injection and persistence across multiple applications.[21]
- **Remote DLL Injection:** This technique involves injecting a malicious DLL into a process running on a remote system. It typically utilizes Windows API functions such as OpenProcess to gain a handle to the target process and WriteProcessMemory to write the DLL path or payload into its memory, followed by creating a remote thread to initiate

execution.[21]

DLL injection can lead to privilege escalation through several mechanisms:

- **Hooking:** Attackers can use function hooking to intercept calls to legitimate functions (e.g., related to authentication or authorization) within a target process. By redirecting these calls to a malicious function within the injected DLL, they can modify process behavior, bypass security checks, and gain elevated privileges.[22]
- **Code Injection:** Malicious code is directly injected into a process that runs with higher privileges (e.g., a system service or a privileged user account). Once the injected code executes, it inherits those higher privileges, allowing access to otherwise restricted resources.[22]
- **Registry Hijacking:** By modifying specific Windows registry keys responsible for loading DLLs, attackers can trick a legitimate process running with high privileges into loading their malicious DLL instead of its intended one. This grants the attacker arbitrary code execution with elevated rights.[22]
- **Image File Execution Options (IFEO) Hijacking:** This technique exploits the IFEO registry key, which is designed to allow debuggers to be specified for executables. An attacker can add a registry entry for a legitimate executable and point it to a malicious DLL as the "debugger." When the legitimate executable launches, the malicious DLL is also launched with the same privileges, enabling code injection and privilege escalation.[22]

DLL injection is a fundamental technique that can lead to privilege escalation through various sub-techniques like hooking, code injection, and registry/IFEO hijacking.[22] This demonstrates that complex attacks are often modular, combining different low-level techniques. DLL injection might serve as the "delivery mechanism" for malicious code, while hooking or code injection acts as the "escalation mechanism" that leverages the injected code. This implies a modularity in attack design, where different components (e.g., a DLL injector, a specific hooking payload) can be combined to form a complex attack chain.

Real-world examples of DLL injection are abundant. The threat group GhostWriter (UAC-0057) has utilized DLL injectors to deploy PicassoLoader and Cobalt Strike beacons. The Russian APT group RomCom employed Shellcode Reflective DLL Injection (sDRI) to bypass sandboxes and deploy their backdoor. The browser hijacker SmashJacker utilized the AppInit_DLL technique for persistence. Numerous other APT groups, including APT41, Astaroth, BlackEnergy, Cobalt Strike, and Emotet, have extensively used various forms of DLL hijacking.[25]

DLL Search Order Hijacking: Exploiting Windows DLL Search Logic

DLL Search Order Hijacking is a specific type of DLL hijacking that exploits a predictable and often insecure behavior of the Windows operating system: the predefined order in which it searches for and loads Dynamic Link Libraries when an application does not specify the full, absolute path to the required DLL.[23]

The Windows DLL search order typically prioritizes certain directories, such as the application's own directory, system directories (C:\Windows\System32), the current working directory, and directories listed in the PATH environment variable.[23] Attackers leverage this by placing a malicious DLL, named identically to a legitimate one, in a directory that Windows will search *before* the legitimate DLL's actual location. This often involves moving legitimate system binaries to non-standard, user-writable directories that also contain the malicious DLL. When the vulnerable application attempts to load the DLL, it inadvertently loads and executes the malicious version first.[23]

This technique is highly favored by adversaries for achieving persistence, escalating privileges, and evading defensive controls, as it allows them to introduce and execute malicious binaries within the context of legitimate, high-reputation executables, making it difficult to distinguish from normal system activity.[23] The success of DLL Search Order Hijacking hinges on the "publicized DLL search order of Microsoft applications" [24] and the fact that a process "will search in the directory it's executing from before iterating through other locations".[29] This is not a traditional software bug but an exploitation of a documented, predictable system design. The attacker leverages this predictability to ensure their malicious DLL is loaded instead of the legitimate one. This highlights that even seemingly benign or performance-oriented system behaviors can become significant attack surfaces if not rigorously secured, emphasizing the importance of secure coding practices (e.g., always specifying full DLL paths) and stringent file system permissions to prevent unauthorized placement of files in search paths.

C. Process Replacement

Process replacement is a malware technique that involves overwriting the *entire memory space* of a running, legitimate process with a malicious executable. Unlike process injection, which inserts code into an existing process's memory while often preserving its original functionality, process replacement completely supplants the original program's memory contents.[31]

This method is employed by malware authors primarily to disguise their malicious activities by making the malware appear as a legitimate system process (e.g., svchost.exe). This approach also minimizes the risk of crashing the host process, which can sometimes occur with more surgical process injection techniques. Crucially, the malicious process inherits the same privileges as the legitimate process it replaces, enabling privilege escalation.[31] Process Hollowing (discussed previously) can be considered a specialized form of process replacement, as it also involves creating a suspended process, unmapping its original code, and replacing it with malicious code before resuming execution.[15] The distinction between "process injection" and "process replacement" [17] indicates that attackers have a range of techniques for manipulating processes, each with its own characteristics and trade-offs (e.g., process injection might be more subtle, while replacement offers full control over the process's memory). This suggests that attackers select specific process manipulation techniques based on their priorities for a given attack scenario, such as maximizing stealth, ensuring execution stability, or bypassing particular security controls. Defenders must therefore be aware of this entire spectrum of techniques and their unique detection challenges.

Table 1: Overview of Process Hijacking and Injection Techniques

Technique Name	**Primary OS**	**Key Mechanism**	**Example API Calls/Concepts**	**Primary Benefit for Attacker**
Thread Execution Hijacking	Windows	Redirects existing thread's execution flow	OpenProcess, SuspendThread, VirtualAllocEx, WriteProcessMemory, GetThreadContext, SetThreadContext, ResumeThread	Stealth, Privilege Escalation
VDSO Hijacking	Linux	Manipulates Virtual Dynamic Shared Object (VDSO) or Global Offset Table (GOT)	GOT Patching, VDSO Page Overwrite	Stealth, Privilege Escalation
Process Hollowing	Windows	Replaces legitimate code in suspended process's memory	CreateProcess(SUSPENDED), ZwUnmapViewOfSection, VirtualAllocEx, WriteProcessMemory, SetThreadContext, ResumeThread	Stealth, Privilege Escalation, Persistence
DLL Injection (General)	Windows	Forces malicious DLL	LoadLibrary, OpenProcess,	Code Execution,

		to load into process address space	WriteProcessMemory, CreateRemoteThread	Privilege Escalation, Data Theft
DLL Search Order Hijacking	Windows	Exploits predictable Windows DLL search logic	Windows DLL Search Order, File System Manipulation	Persistence, Privilege Escalation, Defense Evasion
Process Replacement	Windows	Overwrites entire memory space of legitimate process	Memory Overwrite, Process Hollowing variants	Disguise Malware, Inherit Privileges

Table 2: Real-World Malware/Threat Group Examples by Technique

Technique	**Malware/Threat Group**	**Context/Target**	**Key Impact/Objective**
Thread Execution Hijacking	Lumma Stealer [5]	Banking Trojan	Data theft, Evade detection
	Zloader [5]	Malware Loader	Inject CyberMesh.exe into msiexec.exe
Process Hollowing	Emotet [15]	Banking Trojan, Malware Loader	Hidden execution, Download payloads, Evade detection
	TrickBot [15]	Banking Trojan, Malware Loader	Credential theft, Lateral movement, Ransomware distribution

	Ryuk Ransomware [15]	Ransomware	Encryption routines, Demanding ransom
	Cobalt Strike [15]	Penetration Testing Tool (abused)	Remote access, Privilege escalation, Lateral movement
	REMCOS RAT [20]	Remote Access Trojan	Copy into iexplore.exe, Evade detection
DLL Injection (General)	GhostWriter (UAC-0057) [25]	Threat Group	Deploy PicassoLoader and Cobalt Strike beacon
	RomCom APT Group [25]	Russian APT	Sandbox escape, Deploy RomCom Backdoor
	SmashJacker [25]	Browser Hijacker	Establish persistence, Redirect search results
DLL Hijacking	APT41 [28]	Threat Group	Search order hijacking
	BlackEnergy [27]	Malware	Inject DLL component into svchost.exe

III. Related Privilege Escalation Vectors Leveraging Process Manipulation

A. Access Token Manipulation (T1134)

Understanding Windows Access Tokens and Security Contexts

Windows operating systems fundamentally rely on access tokens to define and manage the security context for users, processes, and threads. An access token is a data structure containing critical security information, including the user's unique identifier (SID), membership in security groups, assigned privileges, and integrity levels.[32] These tokens are intrinsically linked to processes and threads, governing precisely what resources they can access and what actions they are authorized to perform within the system.[32] When a user launches an application, a copy of their access token is typically given to that application, ensuring that processes and threads operate within the user's defined permissions.[33]

Techniques: Token Privilege Modification, Process Creation with Token, Token Impersonation, PPID Spoofing, SID History Injection

Access token manipulation involves adversaries illicitly modifying or leveraging existing access tokens to bypass access controls and gain unauthorized access or elevate their privileges. This is a highly prevalent and effective strategy in Windows environments for both privilege escalation and defense evasion.[32]

Key techniques include:

- **Token Privilege Modification:** Attackers can manipulate an access token to enable specific, often disabled, privileges (e.g., SeDebugPrivilege). This is achieved using functions like AdjustTokenPrivileges().[32] SeDebugPrivilege is particularly potent as it allows debugging or manipulating processes owned by other users, including those with elevated privileges.[32] Attackers frequently aim to escalate privileges to NT Authority\SYSTEM, which represents the highest privilege level on a Windows system, equivalent to root on Unix-based systems.[32]
- **Process Creation with Token:** This technique involves using a stolen or duplicated access token to create new processes that execute under the security context of the impersonated user. Windows API functions such as CreateProcessAsUser and CreateProcessWithTokenW are commonly employed for this purpose.[32]
- **Token Impersonation/Theft:** A thread assumes the identity of another user or security principal. This is typically done to perform actions that require higher privileges than the current user's account. This often involves duplicating a stolen token [33] and using functions like ImpersonateLoggedOnUser or SetThreadToken.[32]
- **PPID Spoofing (T1134.004):** This sub-technique involves manipulating the Parent Process Identifier (PPID) of a newly created process. By spoofing the PPID, attackers can make a malicious process appear as a legitimate child of a trusted parent, thereby evading process monitoring defenses and potentially escalating privileges. Malware like Cobalt Strike,

DarkGate, and KONNI have been observed utilizing PPID spoofing.[32]

- **SID History Injection:** Windows Security Identifiers (SIDs) uniquely identify user, computer, or group accounts. In this technique, adversaries inject or harvest well-known SIDs into an access token's SID history. This manipulation can grant the attacker escalated privileges and access to resources that would otherwise be restricted.[34]

The reliance on access tokens for granular authentication and authorization in Windows [32] is designed for robust security and flexible privilege management. However, the research indicates that "Access token manipulation takes advantage of built-in Windows functionality" [33] and that "blocking use of the functions in question is not a viable solution".[33] This means that the very mechanisms intended to secure the system can be repurposed for malicious ends, making detection challenging. The sophisticated and flexible nature of Windows' privilege management, while a strength for system administration, creates a complex attack surface. Attackers exploit the very mechanisms designed for security. Therefore, defensive strategies must focus on detecting *anomalous patterns of behavior* and *misuse* of legitimate API calls and system functions, rather than simply blocking them. This requires deep understanding of normal system behavior and advanced monitoring capabilities.

B. Exploiting Weak Service Permissions and Misconfigurations

Unquoted Service Paths Vulnerability

This vulnerability arises in Windows environments when a service is installed with an executable path that contains spaces but is not enclosed in double quotes. Windows processes such paths sequentially, attempting to execute each segment as a program until it finds a valid executable.[35]

For example, if a service path is C:\Program Files\My Service\service.exe and is unquoted, Windows might first attempt to execute C:\Program.exe. If an attacker has write permissions to C:\, they can place a malicious Program.exe there, which will then be executed with the privileges of the vulnerable service. This can lead to local privilege escalation, and potentially even domain administrator privileges if the service runs under a domain account.[35] These types of vulnerabilities often stem from "user or installer error" [35] or applications that "configure services incorrectly, by default".[37] This suggests that a significant portion of privilege escalation vulnerabilities arise from simple configuration oversights rather than complex technical exploits.

Weak Service Binary and Registry Permissions

- **Weak Service Binary Permissions:** This vulnerability exists when the executable file of a Windows service has insufficient permissions, allowing a low-privileged user to modify or

replace the service binary itself.[38] If the service is configured to run with high privileges (e.g., as SYSTEM), the modified malicious binary will then execute with those elevated rights upon the service's next start, leading to privilege escalation.[37]
- **Weak Service Permissions:** This refers to incorrect or overly permissive access control lists (ACLs) on the service configuration itself. If a low-privileged user can modify the service's properties (e.g., its binpath or startup type), they can redirect the legitimate service to execute a malicious program, thereby inheriting its elevated privileges.[37]
- **Weak Registry Permissions:** Misconfigured access controls on specific Windows registry keys or entries can allow unauthorized users to manipulate critical system configurations.[40] Attackers can exploit these weaknesses to inject malicious code into registry keys, which can then be executed by privileged processes, leading to unauthorized access. The Image File Execution Options (IFEO) registry key is a notable target for attaching malicious binaries to legitimate system binaries.[41]

The causal link is clear: seemingly minor permission flaws on services, their binaries, or associated registry keys can lead to full system compromise. This underscores the principle of least privilege (PoLP) as a fundamental defense. Overly permissive access controls, even on seemingly innocuous files or registry keys, create significant attack surfaces.

AlwaysInstallElevated Policy Abuse

The AlwaysInstallElevated policy is a Windows Installer setting that, when enabled, allows any user to install Windows Installer packages (.msi files) with elevated (system) privileges, regardless of their actual user account control (UAC) settings or administrative rights.[42] Microsoft strongly advises against enabling this setting due to the massive security risk it poses, as it is equivalent to granting full administrative rights to any user.[42] If this policy is active (by setting the AlwaysInstallElevated value to "1" under both HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE registry keys), any user can install unmanaged applications with SYSTEM permissions, creating a direct path for malicious software installations or system compromise.[42] The AlwaysInstallElevated policy, while potentially designed for ease of software deployment in managed environments, is explicitly stated to be a "massive security risk" because it grants "full administrative rights".[42] This illustrates how features intended to simplify user experience or IT management can inadvertently create severe security vulnerabilities if misconfigured or misunderstood. This highlights the inherent tension between usability/convenience and security, and organizations must carefully weigh these trade-offs and understand the security implications of such policies.

C. Linux-Specific Process-Related Escalations

SUDO Misuse and Configuration Flaws

The sudo command in Linux allows authorized users to execute commands with the privileges of another user, most commonly the root user.[43] While powerful, misconfigured sudoers files (which define sudo privileges) can create significant security weaknesses. Attackers actively seek out and exploit overly permissive sudo configurations, where a user might be allowed to run specific commands as root without proper password prompts or with wildcards that grant excessive power. This allows them to execute arbitrary commands with root privileges.[2] For instance, if sudo rights are granted to programs that allow shell escapes (e.g., vi, nmap, perl), an attacker can leverage these to gain a root shell.[45] The consistent emphasis on sudo "misuse" or "poor configuration" [2] directly leading to privilege escalation underscores the principle of least privilege: granting only the necessary permissions. This highlights the importance of meticulously reviewing sudoers files and understanding the capabilities of binaries granted sudo privileges.

Misconfigured Cron Jobs

Cron jobs are Linux's mechanism for scheduling tasks to run automatically at specified intervals. These tasks can be configured to run with various user privileges, including root.[44] A common misconfiguration occurs when a script executed by a root-privileged cron job is writable by a low-privileged user. An attacker can then modify this script, injecting malicious commands. When the cron job executes, the malicious code runs with root privileges, leading to privilege escalation.[44] Cron jobs, while beneficial for system automation, introduce new attack surfaces if not secured. Any automated process running with elevated privileges must have its inputs, scripts, and execution environment tightly controlled.

PATH Variable Manipulation

The PATH environment variable in Linux specifies the directories where the shell searches for executable commands.[23] When a user types a command, the system iterates through the directories listed in PATH until it finds the executable. If a user's PATH includes a user-writable directory (especially the current directory, .) *before* legitimate system directories (e.g., /usr/bin), an attacker who has write access to that directory can place a malicious binary there, named after a common command (e.g., ls, cat). When the user executes the command, the malicious binary is run instead, potentially with the user's current privileges or exploiting other vulnerabilities.[45] The PATH variable, seemingly an innocuous environment setting, can lead to command hijacking and privilege escalation if misconfigured. This demonstrates that attackers look beyond obvious vulnerabilities to exploit fundamental operating system behaviors and configurations that are often overlooked in security audits.

Linux Process Capabilities Abuse

Linux capabilities provide a more granular approach to privilege management than the traditional all-or-nothing root user. They split the extensive powers of the root user into distinct, smaller units (capabilities) that can be assigned individually to processes or executable files.[43] Attackers can exploit misconfigurations where a binary has unnecessary setuid or setgid capabilities assigned. By executing such a binary, they can gain elevated privileges associated with those specific capabilities, potentially leading to full root access if the capability is powerful enough (e.g., CAP_SYS_ADMIN).[43] This vector includes exploiting vulnerable setuid/setgid binaries or gaining control of files that have elevated capabilities.[43] Linux capabilities are designed to *enhance* security by providing more granular control than traditional root/non-root distinctions.[43] However, if misconfigured (e.g., granting setuid binaries unnecessary capabilities), this granularity becomes a new attack surface. This is a direct example of a security feature becoming a vulnerability if not properly implemented and managed. This highlights that new security mechanisms, while powerful, introduce complexity and new points of failure if not understood and configured correctly.

A comparative analysis of privilege escalation techniques reveals a fundamental architectural difference between Linux and Windows. Linux techniques heavily revolve around file-system permissions (e.g., SUID/SGID, writable cron jobs, PATH manipulation) and direct configuration of user privileges via sudo.[43] In contrast, Windows privilege escalation frequently leverages the intricacies of access tokens, DLL loading mechanisms, and registry settings.[32] This divergence highlights that while the *goal* of privilege escalation is universal, the *attack surfaces and methodologies* are highly OS-specific. This implies that security professionals need specialized knowledge for each environment, and cross-platform solutions must account for these distinct attack surfaces.

Table 3: Common Privilege Escalation Attack Vectors and Their Exploitation (Beyond Direct Process Hijacking)

Attack Vector	**Primary OS**	**Mechanism/Vulnerability**	**Example Exploitation Method**
Credential Exploitation	Both	Weak/stolen credentials, password reuse	Credential stuffing, Password spraying, Pass-the-Hash [1]
Vulnerabilities/Exploits	Both	Unpatched software flaws, design bugs	Kernel Exploits (Dirty COW, Dirty Pipe), Buffer Overflows [1]
Misconfigurations	Both	Overly permissive settings, default credentials	Unquoted Service Paths, Weak Service/Registry Permissions, AlwaysInstallElevated Policy Abuse, Overly permissive

			SUDO [1]
Malware	Both	Malicious software designed for PE	Rootkits, Spyware, Ransomware with PE features [1]
Social Engineering	Both	Human manipulation to gain info/access	Phishing, Spear phishing, MFA fatigue attacks [1]
OS-Specific Exploits (Windows)	Windows	Windows-specific flaws/features	Access Token Manipulation (Token Impersonation, PPID Spoofing, SID History Injection), Sticky Keys, UAC Bypass [1]
OS-Specific Exploits (Linux)	Linux	Linux-specific flaws/features	SUDO Misuse, Misconfigured Cron Jobs, PATH Variable Manipulation, Linux Process Capabilities Abuse [2]

IV. Detection and Mitigation Strategies

A. Foundational Security Practices

Effective defense against process hijacking and privilege escalation requires a multi-layered approach, starting with robust foundational security practices.

- **Principle of Least Privilege (PoLP):** This foundational security concept dictates that users, applications, and processes should be granted only the absolute minimum level of access and permissions necessary to perform their legitimate functions.[44] Implementing PoLP significantly reduces the attack surface and limits the potential impact of a successful compromise, as even if an account is breached, its restricted privileges minimize the damage an attacker can inflict.[49] Despite the complexity of modern process hijacking techniques, PoLP consistently emerges as a primary and effective mitigation, suggesting that many advanced attacks still rely on exploiting fundamental permission oversights.

- **Multi-Factor Authentication (MFA):** MFA adds a crucial layer of security by requiring users to provide at least two different forms of verification before granting access. This significantly hardens accounts against credential-based attacks—a common initial access vector for privilege escalation—as stolen passwords alone are insufficient for unauthorized access.[2] MFA is especially critical for privileged accounts, which are prime targets for escalation.[54]
- **Regular Patch Management and System Updates:** A cornerstone of cybersecurity, regular patching and updating of software, operating systems, and firmware are essential. These updates address known vulnerabilities that attackers frequently exploit for privilege escalation.[2] A proactive approach to patch management, including monitoring for updates and prioritizing based on severity, is non-negotiable.[51]
- **Security Awareness Training:** Human error remains a significant factor in successful cyberattacks. Comprehensive security awareness training educates employees on recognizing and avoiding social engineering tactics, such as phishing emails, which often serve as initial access vectors for subsequent privilege escalation.[1] A knowledgeable workforce is a critical defense against inadvertently granting attackers a foothold.[2]
- **Network Segmentation and Application Control:** Segmenting a network into micro-perimeters isolates workloads and allows for granular security controls, limiting an attacker's ability to move laterally and escalate privileges across the network.[48] Application control, including whitelisting, restricts executable paths, preventing unauthorized scripts or binaries from facilitating privilege escalation and is highly effective against techniques like DLL hijacking.[3]

The sheer number of distinct foundational practices (PoLP, MFA, patching, training, segmentation) consistently recommended indicates that no single control is sufficient. These are presented as a "multifaceted approach" [51], highlighting the necessity of a layered, defense-in-depth strategy. Organizations cannot rely on a single "silver bullet" solution; a robust security posture requires a combination of technical controls, strong policies, and human education.

B. Technical Detection Methods

Beyond foundational practices, advanced technical detection methods are crucial for identifying sophisticated process hijacking and privilege escalation attempts.

- **Monitoring Process Creation and Behavior Anomalies:** Security teams should rigorously monitor process creation events, paying close attention to any processes initiated in a suspended state (indicated by the CREATE_SUSPENDED flag).[15] While not always malicious, this behavior warrants immediate investigation. Monitoring for unusual parent-child process relationships can also reveal suspicious activity.[58] Furthermore, detecting legitimate processes executing unexpected code or launching from unusual directories is a critical indicator of compromise.[15] Many process hijacking techniques involve subtle deviations from normal process behavior, such as suspended processes, memory modifications, or unusual API calls.[15] These are not inherently "malicious" in isolation but serve as "red flags" [56], shifting detection from static signatures to dynamic behavioral analysis.

- **Memory Allocation and API Call Monitoring:** Granular monitoring of memory allocation patterns and specific API calls frequently used in process injection and hollowing is vital. These include functions like OpenProcess, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread, SuspendThread, GetThreadContext, SetThreadContext, ResumeThread, ZwUnmapViewOfSection, and NtUnmapViewOfSection.[5] Additionally, monitoring Windows token APIs such as LogonUser, DuplicateTokenEx, and ImpersonateLoggedOnUser can help detect access token manipulation attempts.[33] The ability to detect these attacks relies heavily on collecting granular system telemetry, particularly API call traces and memory activity. This data, while potentially noisy, is critical for both real-time detection and forensic analysis. Organizations need robust logging and monitoring solutions (e.g., Sysmon, EDR, SIEM) capable of capturing low-level system events and correlating them.
- **DLL Load Point Analysis and File Integrity Monitoring:** Analyzing where applications load DLLs from is crucial, especially when looking for DLLs loaded from non-standard or user-writable locations.[29] Examining DLL imports and exports for suspicious functions can also reveal malicious intent.[56] Furthermore, detecting unsigned DLLs or those with unusual file sizes or hashes being loaded by otherwise signed processes is a significant red flag.[23] File Integrity Monitoring (FIM) solutions can alert security teams to unauthorized changes or renamings of critical DLLs on systems, which could indicate a DLL hijacking attempt.[56] Simply monitoring DLL loads from the same folder as the executable can generate "tons of false positives".[29] The recommendation is to "Target your detection toward user-writable folders like temp or appdata/roaming for the best results".[29] This indicates that raw telemetry alone is insufficient; it needs to be contextualized and filtered based on known attack patterns and legitimate system behavior to be truly effective.
- **Leveraging Endpoint Detection and Response (EDR) and Security Information and Event Management (SIEM):** EDR tools provide continuous, real-time monitoring of endpoints, tracking process creation, memory modifications, and unusual API calls to intercept privilege escalation attempts.[3] They offer deep visibility and rapid response capabilities essential for countering modern threats.[3] SIEM tools collect and analyze system and application logs from multiple sources, correlating event data to identify unusual patterns and trigger alerts for security teams.[15] User and Entity Behavior Analytics (UEBA) solutions further enhance detection by establishing baseline behavioral patterns for users and systems, triggering alerts when anomalies are detected, such as a standard user account suddenly performing administrative actions.[49] EDR and SIEM are repeatedly mentioned as key tools.[15] They provide the necessary visibility and correlation capabilities that individual monitoring tools lack, pointing to a trend towards integrated security platforms for comprehensive threat detection. Effective defense against sophisticated

attacks requires a holistic view of the environment, integrating data from various sources to detect complex attack chains rather than isolated events.

The fact that process hijacking techniques are designed to "evade detection by security tools" [15] and bypass "traditional antivirus and behavioral detection tools" [15] because they run "under the guise of a legitimate process" [15] highlights a paradigm shift. This indicates that traditional static, signature-based approaches are increasingly insufficient. Modern defense against process hijacking requires advanced detection capabilities that can identify anomalous behavior within legitimate processes, rather than just blocking known malicious files. This necessitates investment in EDR, SIEM, and UEBA solutions capable of deep visibility into system processes, memory, and API calls.

C. Defensive Tools and Frameworks

A wide array of tools and frameworks exist to aid both offensive security practitioners in testing for privilege escalation and defensive teams in detection and mitigation.

- **Vulnerability Scanners and Penetration Testing Tools:** Tools such as Metasploit [21], PowerShell [21], Process Explorer [21], and Dependency Walker [21] are used by both attackers and defenders. For Linux environments, various enumeration scripts like LinPEAS (Linux Privilege Escalation Awesome Script), LinEnum, Linux Exploit Suggester 2, and Unix Privesc Checker are invaluable for identifying common misconfigurations and potential escalation paths.[46] On Windows, tools like PowerUp, BeRoot, WinPEAS (Windows Privilege Escalation Awesome Script), and SharpUp assist in identifying privilege escalation vectors.[62]
- **Penetration Testing Frameworks:** Methodologies provided by organizations like OWASP (Web Security Testing Guide - WSTG), the Penetration Testing Execution Standard (PTES), the PCI Penetration Testing Guide, and NIST 800-115 offer structured approaches for identifying privilege escalation flaws and assessing overall security posture.[64] The extensive list of both offensive and defensive tools demonstrates a dynamic and evolving cybersecurity landscape. For every attack technique, there are tools developed to exploit it, and subsequently, tools developed to detect and mitigate it. This highlights the continuous innovation on both sides of the cybersecurity spectrum, emphasizing the need for organizations to stay updated with the latest tools and threat intelligence.

Table 4: Key Detection and Mitigation Controls

Control Category	**Specific Measure**	**Description**	**Relevance to Process Hijacking/PE**

Foundational Practices	Principle of Least Privilege (PoLP)	Grant minimum necessary access to users/processes.	Reduces attack surface; limits impact of compromise. [49]
	Multi-Factor Authentication (MFA)	Requires multiple verification factors for access.	Hardens accounts against credential theft, a common PE vector. [2]
	Regular Patch Management	Timely application of software/OS updates.	Addresses known vulnerabilities exploited for PE. [51]
	Security Awareness Training	Educates users on threat recognition.	Helps prevent initial access via social engineering. [2]
	Network Segmentation	Divides network into isolated zones.	Limits lateral movement post-compromise. [48]
	Application Control/Whitelisting	Restricts executable paths; allows only approved software.	Prevents unauthorized code execution, effective against DLL hijacking. [3]
Technical Detection	Process Behavior Monitoring	Tracks process creation, execution, and anomalies.	Detects suspended processes, unusual parent-child relationships, unexpected code execution. [15]
	Memory/API Call Monitoring	Observes memory allocation patterns and API calls.	Identifies suspicious API sequences (e.g., VirtualAllocEx, WriteProcessMemory, SetThreadContext)

			used in injection. [15]
	DLL Load Analysis	Monitors DLLs loaded by applications.	Flags unsigned DLLs, loads from non-standard paths, or suspicious imports/exports. [29]
	File Integrity Monitoring (FIM)	Detects unauthorized changes to critical files/DLLs.	Alerts on tampering with legitimate binaries or DLLs. [56]
	EDR Solutions	Continuous endpoint monitoring and response.	Provides deep visibility into process activity, memory, and API calls to detect and contain PE. [3]
	SIEM Solutions	Centralized log collection and correlation.	Identifies unusual patterns across multiple data sources indicative of PE. [15]
	UEBA Solutions	Analyzes user/entity behavior for anomalies.	Detects deviations from normal activity baselines, flagging potential PE attempts. [49]
Tools/Frameworks	Vulnerability Scanners	Automated tools to identify known flaws.	Discovers misconfigurations and unpatched vulnerabilities. [51]
	PAM Solutions	Manages and monitors privileged accounts.	Enforces granular access, session recording, and just-in-time privileges. [9]

	Penetration Testing Frameworks	Methodologies for security assessments.	Guides the identification and exploitation of PE vulnerabilities in a controlled manner. 64

V. Conclusion

Process hijacking techniques, encompassing various forms of process injection, DLL manipulation, and process replacement, represent a critical and evolving threat vector in the cybersecurity landscape. These methods are not merely isolated exploits but pivotal enablers for privilege escalation, allowing adversaries to transform an initial, limited foothold into a deeply entrenched and widespread system compromise. The primary allure of these techniques for attackers lies in their dual benefits: the ability to inherit elevated privileges and the inherent stealth gained by operating within the context of legitimate, trusted processes. This strategic advantage enables attackers to evade traditional security controls that primarily focus on detecting new or overtly malicious executables.

The analysis reveals a significant shift in offensive tactics towards memory-resident and fileless operations, directly challenging conventional signature-based defenses. Furthermore, the effectiveness of these attacks often stems not from complex, zero-day vulnerabilities, but from the exploitation of common misconfigurations, predictable system behaviors, and the nuanced intricacies of operating system privilege models. Whether it's the precise API call sequences of Thread Execution Hijacking, the manipulation of Linux's VDSO, the subtle abuse of Windows DLL search orders, or the exploitation of weak service permissions, each technique highlights how attackers leverage fundamental system designs for malicious ends.

To counter this sophisticated threat landscape, a multi-layered, proactive, and adaptive defense posture is imperative. Organizations must move beyond reactive, signature-based security to embrace comprehensive strategies that combine robust foundational practices with advanced technical controls. This includes strict adherence to the Principle of Least Privilege, mandatory Multi-Factor Authentication, rigorous patch management, and continuous security awareness training. Technically, effective defense demands deep visibility into system processes, memory, and API calls, leveraging advanced Endpoint Detection and Response (EDR), Security Information and Event Management (SIEM), and User and Entity Behavior Analytics (UEBA) solutions. These tools enable the detection of subtle behavioral anomalies within legitimate processes, which are the tell-tale signs of process hijacking attempts. The continuous innovation in offensive techniques necessitates a corresponding commitment to vigilance, regular security audits, and constant adaptation to the evolving threat landscape.

Works cited

1. What Is Privilege Escalation? - Definition, Types, Examples ..., accessed June 21, 2025, https://www.proofpoint.com/us/threat-reference/privilege-escalation
2. What is Privilege Escalation? Types, Risks, and How to Defend Against It | Huntress, accessed June 21, 2025, https://www.huntress.com/cybersecurity-education/cybersecurity-101/what-is-privilege-escalation
3. What Is Privilege Escalation? - Cynet, accessed June 21, 2025, https://www.cynet.com/network-attacks/privilege-escalation/
4. csrc.nist.gov, accessed June 21, 2025, https://csrc.nist.gov/glossary/term/process_hijacking#:~:text=Process%20hijacking%20makes%20it%20possible,firewall%20allowing%20the%20server%20component
5. MITRE ATT&CK T1055.003 Process Injection: Thread Execution Hijacking - Picus Security, accessed June 21, 2025, https://www.picussecurity.com/resource/blog/t1055-003-thread-execution-hijacking
6. process hijacking - Glossary | CSRC, accessed June 21, 2025, https://csrc.nist.gov/glossary/term/process_hijacking
7. MITRE ATT&CK T1055 Process Injection - Picus Security, accessed June 21, 2025, https://www.picussecurity.com/resource/blog/t1055-process-injection
8. Process Injection & Shellcode Techniques | Redfox Security, accessed June 21, 2025, https://redfoxsec.com/blog/process-injection-harnessing-the-power-of-shellcode/
9. Privilege Escalation Attacks: Techniques & Examples | Proofpoint US, accessed June 21, 2025, https://www.proofpoint.com/us/blog/identity-threat-defense/privilege-escalation-attack
10. What is Privilege Escalation? | Attack and Defense… - BeyondTrust, accessed June 21, 2025, https://www.beyondtrust.com/blog/entry/privilege-escalation-attack-defense-explained
11. Recognizing the seven stages of a cyber-attack - DNV, accessed June 21, 2025, https://www.dnv.com/cyber/insights/articles/recognizing-the-seven-stages-of-a-cyber-attack/
12. Cyber Kill Chain Model Breakdown and How It Works? - SentinelOne, accessed June 21, 2025, https://www.sentinelone.com/cybersecurity-101/threat-intelligence/cyber-kill-chain/
13. Privilege Escalation Attacks: Types, Examples, And Prevention - PurpleSec, accessed June 21, 2025, https://purplesec.us/learn/privilege-escalation-attacks/
14. Process Injection Techniques - Cynet, accessed June 21, 2025, https://www.cynet.com/attack-techniques-hands-on/process-injection-techniqu

es/
15. What is Process Hollowing? - Portnox, accessed June 21, 2025, https://www.portnox.com/cybersecurity-101/what-is-process-hollowing/
16. Process Injection Attacks and Detection Course - HTB Academy, accessed June 21, 2025, https://academy.hackthebox.com/course/preview/process-injection-attacks-and-detection
17. What is Process Injection? Techniques & Preventions - SentinelOne, accessed June 21, 2025, https://www.sentinelone.com/cybersecurity-101/cybersecurity/process-injection/
18. MITRE ATT&CK T1055.014 Process Injection: VDSO Hijacking, accessed June 21, 2025, https://www.picussecurity.com/resource/blog/t1055-014-vdso-hijacking
19. Advanced process tampering techniques: What are they and how do you detect them?, accessed June 21, 2025, https://www.manageengine.com/log-management/cyber-security/detect-process-tampering-techniques-with-sysmon.html
20. MITRE ATT&CK T1055.012 Process Injection: Process Hollowing - Picus Security, accessed June 21, 2025, https://www.picussecurity.com/resource/blog/t1055-012-process-hollowing
21. Understanding DLL Injection: A Comprehensive Guide to Dynamic Link Library Manipulation, accessed June 21, 2025, https://qa.connect.redhat.com/dll-injection
22. DLL Injection | CQR, accessed June 21, 2025, https://cqr.company/web-vulnerabilities/dll-injection/
23. Intruders in the Library: Exploring DLL Hijacking - Palo Alto Networks Unit 42, accessed June 21, 2025, https://unit42.paloaltonetworks.com/dll-hijacking-techniques/
24. What is DLL Hijacking? The Dangerous Windows Exploit - UpGuard, accessed June 21, 2025, https://www.upguard.com/blog/dll-hijacking
25. MITRE ATT&CK T1055.001 Process Injection: DLL Injection - Picus Security, accessed June 21, 2025, https://www.picussecurity.com/resource/blog/t1055-001-dll-injection
26. DLL Side-loading & Hijacking | DLL Abuse Techniques Overview | Google Cloud Blog, accessed June 21, 2025, https://cloud.google.com/blog/topics/threat-intelligence/abusing-dll-misconfigurations
27. Process Injection: Dynamic-link Library Injection, Sub-technique T1055.001 - Enterprise | MITRE ATT&CK®, accessed June 21, 2025, https://attack.mitre.org/techniques/T1055/001/
28. Detect, prevent and respond: A deep dive on malicious DLLs - Logpoint, accessed June 21, 2025, https://www.logpoint.com/en/blog/deep-dive-on-malicious-dlls/
29. DLL Search Order Hijacking - Protect with Red Canary, accessed June 21, 2025, https://redcanary.com/threat-detection-report/techniques/dll-search-order-hijacking/
30. Possible DLL Search Order Hijacking, accessed June 21, 2025,

https://docs-cortex.paloaltonetworks.com/r/Cortex-XDR/Cortex-XDR-Analytics-Alert-Reference-by-data-source/Possible-DLL-Search-Order-Hijacking?contentId=NTRGlLpfFK4BHfVHsEQM0Q

31. Process Replacement - Practical Malware Analysis [Book], accessed June 21, 2025, https://www.oreilly.com/library/view/practical-malware-analysis/9781593272906/ch13s03.html
32. Detecting Access Token Manipulation Attacks Course - HTB Academy, accessed June 21, 2025, https://academy.hackthebox.com/course/preview/detecting-access-token-manipulation-attacks
33. MITRE ATT&CK vulnerability spotlight: Access token manipulation | Infosec, accessed June 21, 2025, https://www.infosecinstitute.com/resources/mitre-attck/mitre-attck-access-token-manipulation/
34. Access Token Manipulation - ManageEngine, accessed June 21, 2025, https://www.manageengine.com/products/eventlog/cyber-security/access-token-manipulation-detection.html
35. Unlocking the Mystery of Unquoted Service Paths: Another ..., accessed June 21, 2025, https://blackpointcyber.com/blog/unlocking-the-mystery-of-unquoted-service-paths-another-opportunity-for-privilege-escalation/
36. Windows Unquoted Service Path Privilege Escalation - Rapid7, accessed June 21, 2025, https://www.rapid7.com/db/modules/exploit/windows/local/unquoted_service_path/
37. Hidden Danger: How To Identify and Mitigate Insecure Windows ..., accessed June 21, 2025, https://offsec.blog/hidden-danger-how-to-identify-and-mitigate-insecure-windows-services/
38. Windows-Local-Privilege-Escalation-Cookbook/Notes ... - GitHub, accessed June 21, 2025, https://github.com/nickvourd/Windows-Local-Privilege-Escalation-Cookbook/blob/master/Notes/WeakServiceBinaryPermissions.md
39. Windows-Local-Privilege-Escalation-Cookbook/Notes/WeakServicePermissions.md at master - GitHub, accessed June 21, 2025, https://github.com/nickvourd/Windows-Local-Privilege-Escalation-Cookbook/blob/master/Notes/WeakServicePermissions.md
40. Windows-Local-Privilege-Escalation-Cookbook/Notes ... - GitHub, accessed June 21, 2025, https://github.com/nickvourd/Windows-Local-Privilege-Escalation-Cookbook/blob/master/Notes/WeakRegistryPermissions.md
41. Registry keys used for privilege escalation - Splunk Lantern, accessed June 21, 2025, https://lantern.splunk.com/Security/UCE/Foundational_Visibility/Compliance/Recognizing_improper_use_of_system_administration_tools/Registry_keys_use

d_for_privilege_escalation
42. AlwaysInstallElevated - Win32 apps | Microsoft Learn, accessed June 21, 2025, https://learn.microsoft.com/en-us/windows/win32/msi/alwaysinstallelevated
43. Unlocking Power Safely: Privilege Escalation via Linux Process Capabilities - Elastic, accessed June 21, 2025, https://www.elastic.co/security-labs/unlocking-power-safely-privilege-escalation-via-linux-process-capabilities
44. What is Privilege Escalation | Prevention Techniques - Imperva, accessed June 21, 2025, https://www.imperva.com/learn/data-security/privilege-escalation/
45. Linux Privilege Escalation Guide (Updated for 2024) - Payatu, accessed June 21, 2025, https://payatu.com/blog/a-guide-to-linux-privilege-escalation/
46. Linux Privilege Escalation: Techniques, Prevention & More - StrongDM, accessed June 21, 2025, https://www.strongdm.com/blog/linux-privilege-escalation
47. Linux Privilege Escalation: Techniques and Security Tips - Vaadata, accessed June 21, 2025, https://www.vaadata.com/blog/linux-privilege-escalation-techniques-and-security-tips/
48. Mastering Privilege Escalation: Techniques & Prevention Strategies - Admin By Request, accessed June 21, 2025, https://www.adminbyrequest.com/en/blogs/mastering-privilege-escalation-techniques-prevention-strategies
49. The Essential Guide to Privilege Escalation Attacks - Suridata, accessed June 21, 2025, https://www.suridata.ai/blog/the-essential-guide-to-privilege-escalation-attacks/
50. Windows Privilege Escalation Fundamentals - BugBase Blogs, accessed June 21, 2025, https://bugbase.ai/blog/windows-privilege-escalation-fundamentals
51. How Privilege Escalation Works and 6 Ways to Prevent It - Exabeam, accessed June 21, 2025, https://www.exabeam.com/explainers/insider-threats/how-privilege-escalation-works-and-6-ways-to-prevent-it/
52. Threats A Deep Dive Into Privilege Escalation - Packetlabs, accessed June 21, 2025, https://www.packetlabs.net/posts/a-deep-dive-into-privilege-escalation/
53. How Credential Attacks Work and 5 Defensive Measures [2025 Guide] - Exabeam, accessed June 21, 2025, https://www.exabeam.com/explainers/insider-threats/how-credential-attacks-work-and-5-defensive-measures/
54. What Is Privilege Escalation? Types and Prevention Strategies - Wiz, accessed June 21, 2025, https://www.wiz.io/academy/privilege-escalation
55. 5 Strategies to Stop Privilege Escalation Attacks | Fidelis Security, accessed June 21, 2025, https://fidelissecurity.com/threatgeek/threat-detection-response/stop-privilege-escalation-attacks/
56. How Hackers Hijack Applications Using Malicious DLLs: And How To Improve Cyber Defenses Against It, accessed June 21, 2025,

https://insanecyber.com/what-is-dll-hijacking/
57. What Is Privilege Escalation? Types, Examples, and Prevention - Legit Security, accessed June 21, 2025, https://www.legitsecurity.com/aspm-knowledge-base/what-is-privilege-escalation
58. The Nightmare of Proc Hollow's Exe - TrustedSec, accessed June 21, 2025, https://trustedsec.com/blog/the-nightmare-of-proc-hollows-exe
59. A Comprehensive Guide to Privilege Escalation: Understanding, Prevention, and Mitigation, accessed June 21, 2025, https://www.startupdefense.io/blog/a-comprehensive-guide-to-privilege-escalation-understanding-prevention-and-mitigation
60. Windows Privilege Escalation Course - HTB Academy, accessed June 21, 2025, https://academy.hackthebox.com/course/preview/windows-privilege-escalation
61. Process Injection - Red Canary Threat Detection Report, accessed June 21, 2025, https://redcanary.com/threat-detection-report/techniques/process-injection/
62. nickvourd/Windows-Local-Privilege-Escalation-Cookbook - GitHub, accessed June 21, 2025, https://github.com/nickvourd/Windows-Local-Privilege-Escalation-Cookbook
63. Post-Exploitation Privilege Escalation Tools | Ethical Hacking - Armur AI, accessed June 21, 2025, https://armur.ai/ethical-hacking/post/post-1/post-exploitation-privilege-escalation-tools/
64. Penetration Testing Methodologies - WSTG - Latest | OWASP Foundation, accessed June 21, 2025, https://owasp.org/www-project-web-security-testing-guide/latest/3-The_OWASP_Testing_Framework/1-Penetration_Testing_Methodologies
65. Testing for Privilege Escalation - WSTG - Latest | OWASP Foundation, accessed June 21, 2025, https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/03-Testing_for_Privilege_Escalation