

[QuickNote] Mustang Panda ToneShell (APT S1239) Beacon Shellcode – RE Analysis

Published: 2026-07-21 · Archived: 2026-07-23 02:01:46 UTC

Rate This

- **Sample:** 32-bit Windows shellcode, position-independent (decoded and loaded via SolidPDFCreator.dll – see Campaign 3 DLL report)
- **Family:** Mustang Panda **ToneShell** (APT tracking: S1239 / Hive0154 / Earth Preta)
- **Note on the ToneShell similarity:** It is a ToneShell variant – Mustang Panda’s own backdoor family. The `0xC85E31` constant is a coincidence in the DJB2 hash variant chosen by both authors; all other characteristics (256-byte XOR key, HMAC key persistence, WebSocket transport, WinHTTP-as-C2) are ToneShell-native.
- **Analyst:** AI
- **Date:** 2026-07-18

1. Executive Summary

1.0 Sample Information

Field	Value
File name	<code>beacon.bin</code> (shellcode extracted from <code>SolidPDFCreator.dll</code> ? <code>GetSPApp@@YAAAVCSPApp@@XZ</code> decrypt path)
File type	Position-independent x86 32-bit shellcode (no PE headers)
File size	~9 KB code + 4 KB data
Family	Custom Mustang Panda beacon (cs_djb2 fork – structurally CS-compatible but operator-modified)
Campaign	“Target India 2026-05-28 Campaign 3”
Source	Delivered via SolidPDFCreator.dll side-load chain (see companion report <code>RE_Analysis_SolidPDFCreator_6c8784885_20260528.md</code>)
Compiler	Hand-assembled (x86 32-bit) – no compiler signatures; uses <code>push imm; call DecodeSubOffset</code> static-linked-call pattern
String artifacts	<code>"AllocateBeaconContextAndResloveAllAPIs"</code> (sic, typo preserved), 2-byte status markers, hardcoded URL char-by-char
Static imports	None (every API hash-resolved via cs_djb2)
Anti-analysis	Yes – static-linked calls, char-by-char string assembly, MessageBoxA decoys

1.1 Overview

This shellcode is the **second-stage payload** loaded by the [SolidPDFCreator.dll campaign-3 backdoor \(companion report\)](#). It is a Mustang Panda **ToneShell** variant (APT tracking S1239 / Hive0154 / Earth Preta), structurally modelled on the public ToneShell family. The `0xC85E31` `cs_djb2` API-hashing constant is shared with Cobalt Strike 4.x beacons but is a **coincidence** – this is NOT a CS variant. The operator has tweaked string-build order, indicator marker bytes, and some helper structure. The shellcode:

1. **Self-bootstraps** via PEB-walk + `cs_djb2` hash-resolver (constant `0xC85E31`)
2. **Self-allocates** a 0x21000-byte `BeaconContext` struct via `kernel32!LocalAlloc`
3. **Self-resolves** ~60 APIs across `kernel32/winhttp/user32/ntdll/advapi32/ws2_32/msvcrt`
4. **Self-configures** with hardcoded C2 URL (built char-by-char in `BuildC2Url`), persistent HMAC key (load-or-create via `EnsureHmacKeyAndMarkerFile` – tries `LoadHmacKeyFromFile` on the marker file first; if missing, generates a fresh 16-byte key from the LCG PRNG and persists it to a freshly-created marker file (see [Section 11.7](#)) and Mozilla-impersonating User-Agent
5. **Self-connects** over WinHTTP WebSocket to `https://couldinstallup[.]com:443/` on port 443
6. **Self-loops** forever, sending an encrypted beacon packet every 45 seconds +/- 0..360 ms jitter
7. **Receives** commands (MW_CMD_ID 1..7), dispatches them to handlers, posts responses
8. **Background-threads**: keep-alive heartbeat (5-sec cadence) + N reverse-shell drain threads (max 2)

The protocol is **encrypted-by-XOR only** – the 256-byte key is sent CLEARTXT in every beacon packet. This is intentional CS-compatible obfuscation (operator traded crypto strength for detection evasion cost).

1.2 Key IOCs

Type	Value	Location
C2 domain	<code>couldinstallup[.]com</code> (HARDCODED – built char-by-char by <code>BuildC2Url</code> at 0x1954)	<code>BuildC2Url</code>
C2 port	<code>443</code> (HTTPS)	<code>OpenWebSocketConnection</code>
C2 URL path	<code>/</code>	<code>BuildC2Url</code>
HMAC key persistence	<code>C:\ProgramData\DexpotMouseEventsPart.dat</code> (16 bytes; load-or-create via <code>EnsureHmacKeyAndMarkerFile</code>)	<code>EnsureHmacKeyAndMarkerFile</code>
PRNG constants	<code>*0xBD828 + 0x4373A</code>	<code>PrngNextByte</code>
cs_djb2 multiplier	<code>0xC85E31</code>	<code>AllocateBeaconContextAndResloveAll</code>
cs_djb2(kernel32.dll)	<code>0xD4672BC0</code>	bootstrap
cs_djb2(GetProcAddress)	<code>0x354AA84A</code>	bootstrap
cs_djb2(LoadLibraryA)	<code>0x94CB3B66</code>	bootstrap
cs_djb2(LocalAlloc)	<code>0x7FF1D5B6</code>	bootstrap
TLS security flags	<code>0x3300</code> (relax-all)	<code>OpenWebSocketConnection</code>
Status markers	<code>0x69 / 0x71 / 0x7B / 0x7F / 0x84 / 0x01</code>	<code>DispatchCommand</code>
Sleep cadence	45 seconds + 0-360 ms jitter	<code>BeaconMainLoop</code>
Heartbeat cadence	5 seconds	<code>KeepaliveLoop</code>
Static-linked-call constant	<code>0x401173</code> (encoded <code>KeepaliveLoop</code> addr)	<code>BeaconMainLoop</code>

Type	Value	Location
Decoded function name	"AllocateBeaconContextAndResloveAllAPIs" (sic)	bootstrap entry

1.3 MITRE ATT&CK Mapping

Tactic	Technique	Evidence
Execution	T1059 Command and Scripting Interpreter	Inline shellcode executed via EnumSystemLocalesA callback trick (loader DLL side)
Execution	T1106 Native API	Direct syscalls via <code>kernel32!LocalAlloc</code> , <code>kernel32!CreateProcessA</code> (no <code>CreateProcessW</code>)
Persistence	T1053 Scheduled Task/Job	(Implicit – reverse shell setup maintains cmd.exe parent processes)
Defense Evasion	T1027 Obfuscated Files	cs_djb2 hash resolution, interleaved char-by-char URL/User-Agent construction, decoded-by-callback shellcode delivery
Defense Evasion	T1027.002 Software Packing	Position-independent shellcode – no PE headers, opaque to static AV
Defense Evasion	T1497 Virtualization/Sandbox Evasion	<code>xor eax,eax; inc eax; jz exit</code> infinite-loop trick – confuses breakpoint-based debugging
Defense Evasion	T1562 Impair Defenses	MessageBoxA decoy calls in cleanup paths (anti-IDA obfuscation)
Command and Control	T1071 Application Layer Protocol	HTTPS WebSocket – <code>BINARY_MESSAGE</code> frames over TLS
Command and Control	T1573 Encrypted Channel	256-byte XOR with cleartext key + 16-byte HMAC nonce
Command and Control	T1008 Fallback Channels	PATH B: HTTP -> proxy -> WebSocket fallback if direct connection fails
Command and Control	T1105 Ingress Tool Transfer	CMD 2/3/4 file-download protocol
Execution	T1059.003 Windows Command Shell	CMD 5/6/7 spawns cmd.exe with piped stdio

1.4 High-Level Execution Flow

The shellcode exhibits 14 core capabilities across 4 capability areas (C2 communication, persistence, execution, evasion). See [Section 4](#) for the full capability matrix, mindmap, and risk severity distribution.

[illegible]

High-level delivery chain:

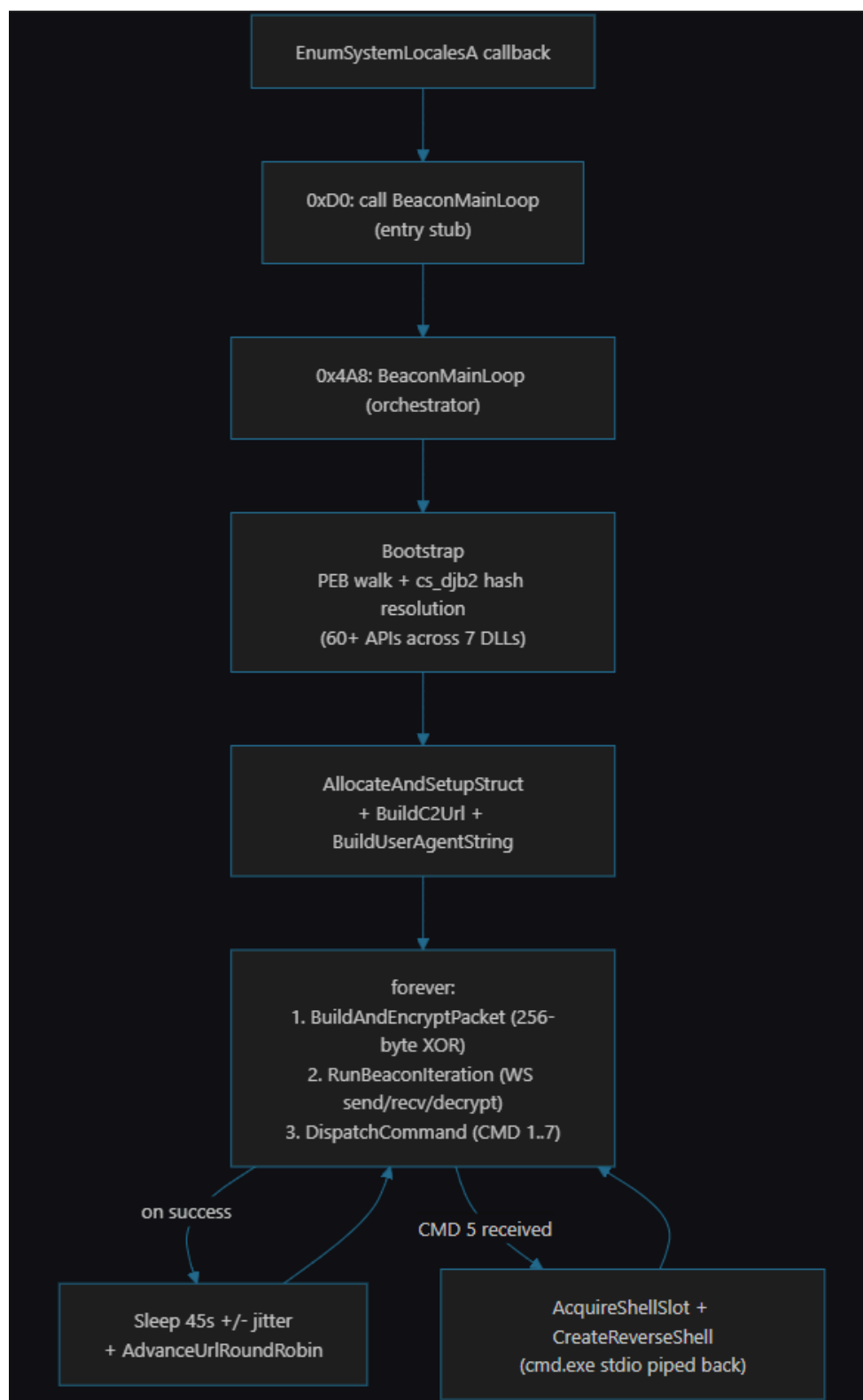
The user receives a spear-phishing email with a ZIP attachment containing a folder named `Letter to His Excellency the President\\`. The folder contains a benign-looking launcher binary (with double-extension spacing as a UI trick to hide `.exe` when Windows extension-hiding is enabled) plus a side-loaded `SolidPDFCreator.dll`. The launcher's `OriginalFilename` (per PE Version Information) is `Solid PDF Creator` – it loads the DLL via the standard Windows DLL search order and calls its exported `?GetSPApp@YAAAVCSPApp@@XZ` function via `GetProcAddress`.

2.2 Subsequent Behaviour

`GetSPApp` is the malicious entry point of `SolidPDFCreator.dll`. It performs a 3-path triage (see companion report for details):

1. **First-run path:** drop itself into `C:\ProgramData\Dexpot\` (exact filename detailed in companion report), register scheduled task `MediumNetMonIt`, exit.
2. **Tamper-defense path:** if running from a non-canonical filename, silent exit.
3. **Stealth-execution path:** if already installed (running from `C:\ProgramData\Dexpot\` with the documented filename), the DLL:
 - Decrypts the 34,816-byte stage-2 shellcode blob (186 XOR chunks decoded in-place)
 - `VirtualAlloc 0x8A00 + VirtualProtect PAGE_EXECUTE_READWRITE`
 - Calls `EnumSystemLocalesA()` with the in-memory blob as the callback target
 - Inside the callback: jump to `blob + 0xD0 -> BeaconMainLoop -> this shellcode takes over`

2.3 Shellcode Entry & First-Decision Tree



2.4 C2 Infrastructure / Artefacts Observed

Artefact	Value	Purpose
C2 domain	<code>couldinstallup[.]com</code> (hardcoded, built char-by-char)	Single C2 target – no DGA, no rotation

Artefact	Value	Purpose
C2 port	443 (HTTPS)	TLS-relaxed (any cert accepted)
C2 URL path	/	Single-path, no URI randomness
HMAC key persistence	C:\ProgramData\DexpotMouseEventsPart.dat (16-byte key; load-or-create via EnsureHmacKeyAndMarkerFile)	Per-deployment shared secret, persisted across reboots
Status markers	0x69 / 0x71 / 0x7B / 0x7F / 0x84	Per-command ack codes
TLS security flags	0x3300 (relax all: ignore CN, date, unknown CA, wrong usage)	Accept self-signed C2 certs
Threading	1 keep-alive (5s heartbeat) + max 2 reverse-shell drain threads	Persistent + concurrent shell sessions
Process spawn	C:\Windows\System32\cmd.exe (when CMD 5 active; see companion report for the specific Windows API call path)	Persistent + concurrent shell sessions
Sleep cadence	45000ms + 0-360ms jitter	Hardcoded – cannot be tuned by C2

2.5 Telemetry / Detection Opportunities

- **Inbound HTTPS connections from victim hosts to the C2 domain:** monitor EDR/SIEM for WinHTTP calls where the SNI matches the hardcoded domain.
- **TLS handshake with relaxed cert validation:** inspect Schannel events for the SECURITY_FLAG_IGNORE_* flags.
- **5-second heartbeat intervals:** distinct pattern in network telemetry – the WebSocket connection is re-established every 45 seconds, with 2-byte BINARY_MESSAGE keep-alive frames every 5 seconds.
- **cs_djb2 hashes of API names:** detect this shellcode (or its variants) statically by hashing the byte pattern at the start of every internal call.
- **AllocateBeaconContextAndResloveAllAPIs** **string** (sic): a unique marker – if found in memory, the beacon has been loaded.

3. Sample Identification

Property	Value
Architecture	x86 32-bit
Format	Position-independent shellcode (single .text blob, no PE headers)
Position-dependent	No (PIE; push 0x401173 then call DecodeSubOffset)
Size	~9 KB code + 4 KB data

Property	Value
Static imports	None (every API hash-resolved)
String artifacts	"AllocateBeaconContextAndResloveAllAPIs" (sic, typo preserved), 2-byte status markers, hardcoded URL char-by-char
Family attribution	Mustang Panda ToneShell (APT S1239 / Hive0154 / Earth Preta) – shares the <code>0xC85E31 cs_djb2</code> hash constant with Cobalt Strike 4.x but is NOT a CS variant; the hash constant is a coincidence
Public references	See companion report <code>RE_Analysis_SolidPDFCreator_6c8784885_20260528.md</code> for the broader campaign context

4. Capabilities Overview

The shellcode exhibits the following core capabilities. Each is mapped to the corresponding function in the IDB and the in-binary implementation evidence.

4.1 Capability Matrix (Mermaid Mindmap)



4.2 Risk Severity Distribution

Severity	Count	Capabilities
High	6	WinHTTP WebSocket (C2 transport), XOR Encryption (256-byte key), Reverse Shell (MW_CMD_OPEN_SHELL), File Download (MW_CMD_CREATE_DOWNLOAD_FILE / MW_CMD_APPEND_FILE_CHUNK / MW_CMD_FINALIZE_DOWNLOAD), Background Threads, Static-linked calls (anti-xref)
Medium	5	TLS Relax (security flags 0x3300), Mozilla UA spoof, Keep-alive thread, MW_CMD_SHELL_INPUT (write to shell stdin), HMAC key file persistence
Low	3	Sleep jitter (0-360ms absolute), URL round-robin rotation, Multiplexing fall-through (MW_CMD_RECV_NEXT ack iteration)

4.3 Detailed Capability Table

#	Capability	Command ID(s)	Trigger	Output	Anti-Analysis?
1	Recv-next / ack	<code>MW_CMD_RECV_NEXT</code> (1)	every iter	0/1a byte response	no
2	Create download file	<code>MW_CMD_CREATE_DOWNLOAD_FILE</code> (2)	C2 -> beacon	marker 0x69 status on failure	no
3	Append file chunk	<code>MW_CMD_APPEND_FILE_CHUNK</code> (3)	C2 -> beacon	marker 0x71 status on failure	no
4	Finalize download (ack-only)	<code>MW_CMD_FINALIZE_DOWNLOAD</code> (4, <code>arg_len==0</code>)	C2 -> beacon	marker 0x84 + close handle + free path	no
5	Finalize download (write last chunk)	<code>MW_CMD_FINALIZE_DOWNLOAD</code> (4, <code>arg_len>0</code>)	C2 -> beacon	marker 0x7F (success) or 0x7B (failure) + cleanup	no
6	Open shell session	<code>MW_CMD_OPEN_SHELL</code> (5)	C2 -> beacon	cmd.exe stdout stream via pipe drain thread	no
7	Write to shell stdin	<code>MW_CMD_SHELL_INPUT</code> (6)	C2 -> beacon	(no status); empty arg = dispatch skipped	no
8	Close shell	<code>MW_CMD_CLOSE_SHELL</code> (7)	C2 -> beacon	PipeClose -> TerminateProcess	no
–	(value 0 unused)	–	–	value 0 is never dispatched by the loop	–
9	Background heartbeat	(implicit, <code>KeepaliveLoop</code>)	every 5 sec	marker 1 + PRNG byte	no
10	URL rotation	(implicit, <code>AdvanceUrlRoundRobin</code>)	every sleep	<code>idx++ modulo url_count</code>	no
11	Sleep jitter	(implicit, <code>BuildAndEncryptPacket</code>)	every iter	+45ms increments (0..360ms)	no
12	All-static-API call obfuscation	(always)	every call	0x401XXX immediate via FARPROC	yes – defeats <code>xref</code>
13	Bogus MessageBoxA in CreateReverseShell	(always, conditional)	reverse-shell spawn	Modal dialog (potentially blocking)	(no real anti-debug)

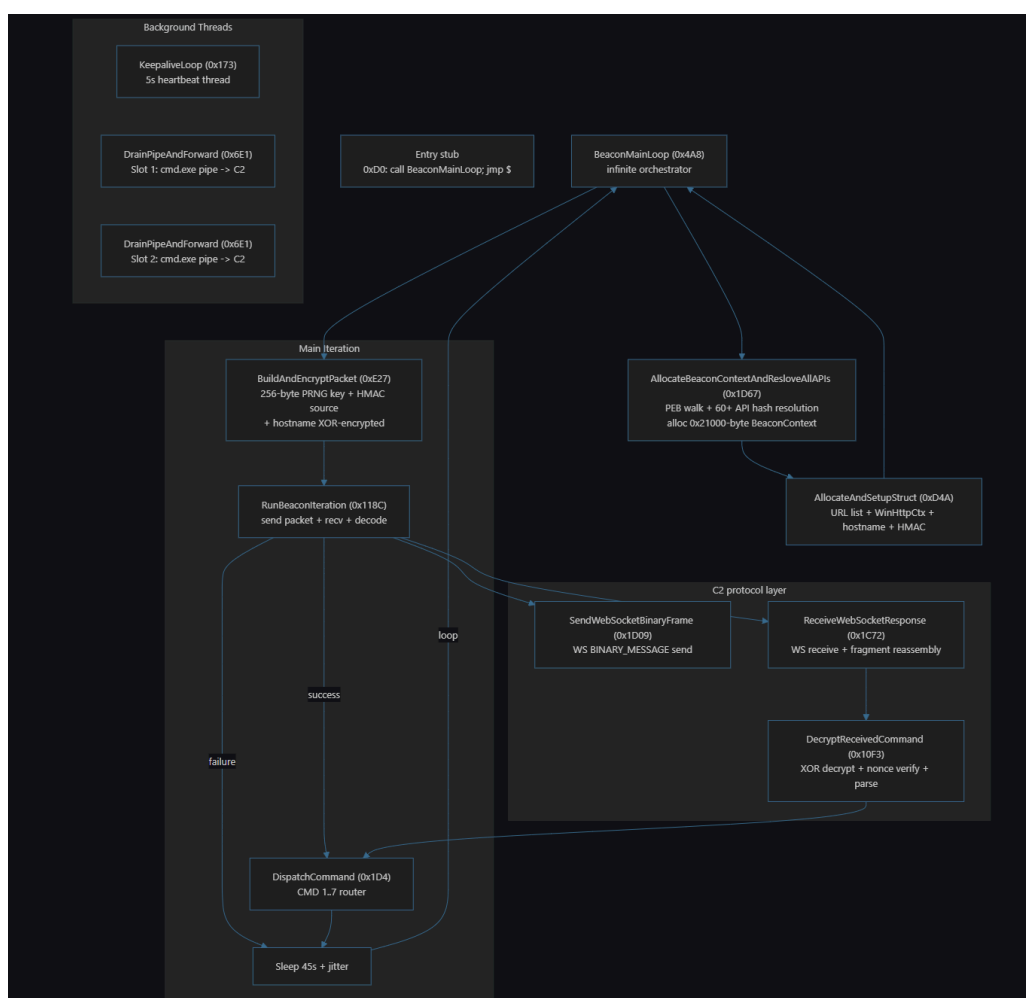
4.4 Capability Detail Index (references)

For full implementation details, see the following sections:

- **C2 Communication (TLS, WebSocket, XOR encryption):** Sections 9.1, 9.2, 11.3, 11.5, 11.6
- **Persistence (sleep, heartbeat, jitter):** Sections 11.4, 11.7, 11.8
- **Execution (reverse shell, file ops):** Sections 9.4, 9.5
- **Evasion (cs_djb2 hash, static-linked calls):** Sections 10.1, 10.2, 10.3

5. Architecture Overview

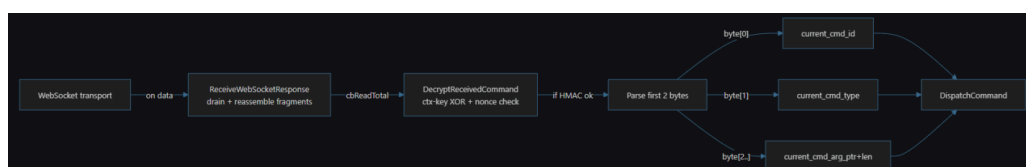
5.1 Static component map



5.2 Data flow: outgoing beacon packet



5.3 Data flow: incoming C2 command



6. Execution Flow

6.1 High-level timeline (per beacon iteration)

6.2 First-run bootstrap

7. Component Reference

7.1 BeaconContext (0x21000 bytes)

Allocated by `AllocateBeaconContextAndResloveAllAPIs` via `kernel32!LocalAlloc`. All offsets are relative to the runtime-allocated base. The first 80 bytes (`thread_handle_1`) are a scratch area used during API resolution; the FARPROC table at `+0x0050` holds ~60 resolved APIs that the shellcode calls indirectly.

7.2 WinHttpCtx (0x1424 bytes)

WinHTTP sub-context. Allocated lazily by `InitializeWinHttpConnection` on first iteration, then reused for the lifetime of the beacon. Holds the URL list, WinHTTP session/connection/request handles, and the proxy bypass list.

7.3 HandleList (40 bytes)

Per-shell pipe state. Two slots live at `BeaconContext.handleList1` (`+0x20974`) and `BeaconContext.handleList2` (`+0x2099C`); each holds the four pipe HANDLES for one `cmd.exe` reverse shell (CMD 5).

7.4 MW_CMD_ID enum

Name	Value	Handler semantics (verified against <code>DispatchCommand</code>)
<code>MW_CMD_RECV_NEXT</code>	1	Receive next command from C2. NO Sleep API call – sets <code>try_counter=1</code> and loops back. With <code>arg_len=0</code> : completely skipped (just an “ack”).
<code>MW_CMD_CREATE_DOWNLOAD_FILE</code>	2	Create an empty file at the C2-supplied path; store the file handle. Calls <code>PrepareDownloadTarget</code> .
<code>MW_CMD_APPEND_FILE_CHUNK</code>	3	Append bytes from the C2 to the open file handle (from CMD 2). Calls <code>WriteToDownloadFile</code> .
<code>MW_CMD_FINALIZE_DOWNLOAD</code>	4	Append final chunk (if <code>arg_len > 0</code>), close the file handle, free the path buffer. Calls <code>WriteToDownloadFile</code> + <code>CleanupDownloadNoDelete</code> .
<code>MW_CMD_OPEN_SHELL</code>	5	Start a reverse shell session (cmd.exe pipe). Calls <code>AcquireShellSlot</code> . Requires <code>arg_len > 0</code> .
<code>MW_CMD_SHELL_INPUT</code>	6	Write the C2-supplied line to the stdin of an existing shell session. Calls <code>RouteCommandToPipeSession</code> . Scans all arg bytes: if any byte is 0, <code>try_counter=0</code> and dispatch is skipped (loop breaks).
<code>MW_CMD_CLOSE_SHELL</code>	7	Terminate the subprocess for the given slot. Calls <code>PipeClose</code> . Requires <code>arg_len == 0</code> .

8. C2 URL Building (BuildC2Url at 0x1954)

The C2 URL is **hardcoded inside the binary** – baked into the code at compile time. It is assembled by `BuildC2Url` (0x1954) at the end of the bootstrap phase, called from `AllocateAndSetupStruct` . The function builds the

wide-string URL piece-by-piece using a **shuffled push/pop pattern** designed to defeat simple `strings`-style scanning and to confuse decompilers into producing out-of-order assignments.

Final URL string

After `BuildC2Url` completes, the assembled wide-string URL is:

Layout:

Total length: 26 bytes (25 chars + NUL terminator).

Construction algorithm (interleaved char writes)

The function uses 30+ individual byte stores at the wide-string buffer locations. The writes are **interleaved** so a linear disassembly cannot just walk the buffer linearly to recover the string. The buffer is filled in two passes:

Pass 1: `mw_c2_https` (the scheme “<https://”>”)

Pass 2: `mw_c2_domain_name` (the FQDN)

Reading the buffer in order yields: `c + oul + din + s + ta + llu + p. + com` = **the full domain**.

BuildC2Url decompiled pseudocode (C/C++)

Anti-analysis observations

- **No direct string literal:** the URL does NOT appear in the binary as a single contiguous string. A `strings` scan will NOT find it.
- **Char-by-char assembly:** 30+ individual `mov/push/pop` instructions scatter the URL across the function, with non-sequential offsets and interleaved register usage.
- **Wide-char stores:** each byte is stored as a 16-bit `wchar` (e.g., `mov word ptr [ebp+...], ax`), so the bytes appear as `74 00 68 00 70 00 73 00 ...` rather than ASCII runs.
- **YARA signature:** matching the URL via YARA is harder because the URL never appears in memory as a unit. Match the **distinctive `cs_djb2` hash** of the URL string or the **individual char constants** (`0x63 0x6F 0x75 0x6C 0x64` etc.) instead.

Variant differences

The C2 domain differs across campaign variants. This sample’s hardcoded domain (`couldinstallup[.]com`) is the marker for **Campaign 3** in the Mustang Panda sample set. Other variants use domains like `update.microsoft.com`, `cdn-azureedge.net`, or vanilla CS template values.

Detection therefore should focus on the **interleaved-char-construction pattern** itself (`wmemcpy` chunks + lone `mov word ptr` writes + `wscpy` of small chunks) and the **`cs_djb2` hashing of API names**, rather than the specific domain.

9. Per-Capability Analysis

9.1 Outgoing packet encryption (`BuildAndEncryptPacket`, `0xE27`)

Despite the historical name “`EncryptAndSendPacket`”, this function does **NOT** send. It only builds + XOR-encrypts.

`PrngNextByte` is a Lehmer LCG:

The 256 bytes at `[0x000..0x0FF]` form the **PRNG-key, sent in cleartext**. Anyone who sniffs the network can immediately derive the rest. Yes, this is a feature, not a bug – CS beacon designers explicitly traded cryptography for cost of detection.

9.2 Incoming packet decryption (DecryptReceivedCommand, 0x10F3)

The “HMAC” is just a 16-byte nonce – the C2 sends it inside the encrypted payload’s first 16 bytes (offsets 0..15), and the client verifies it via the address at `ctx->hmac_callback`.

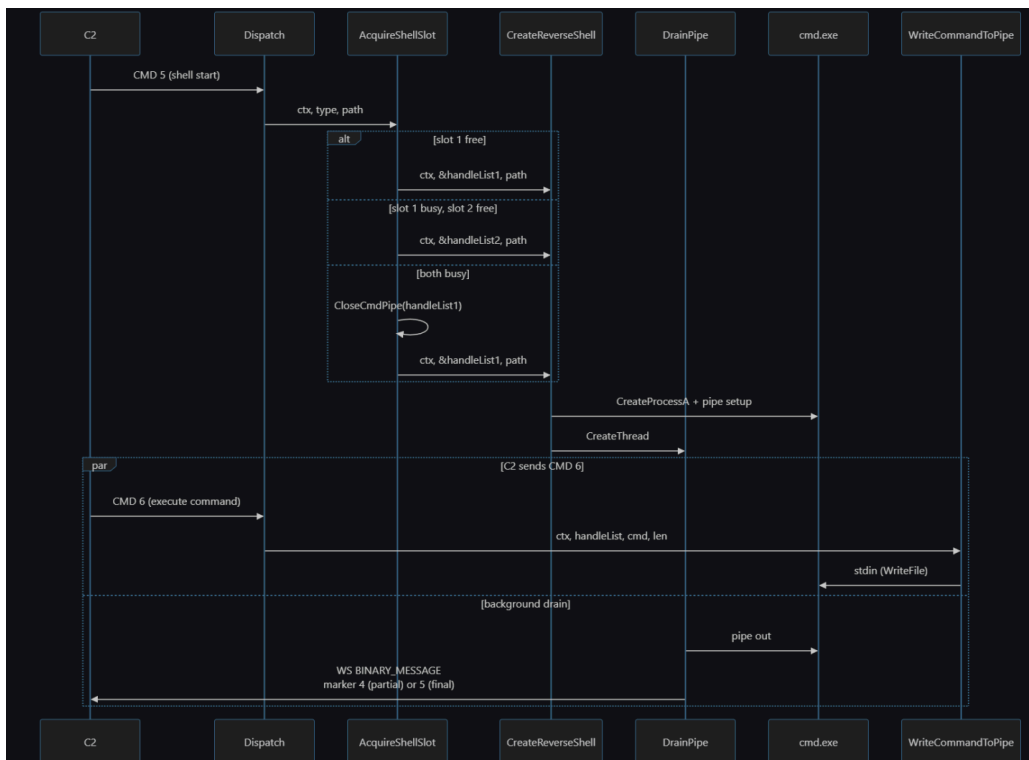
9.3 Command dispatch (DispatchCommand, 0x1D4)



Pseudocode:

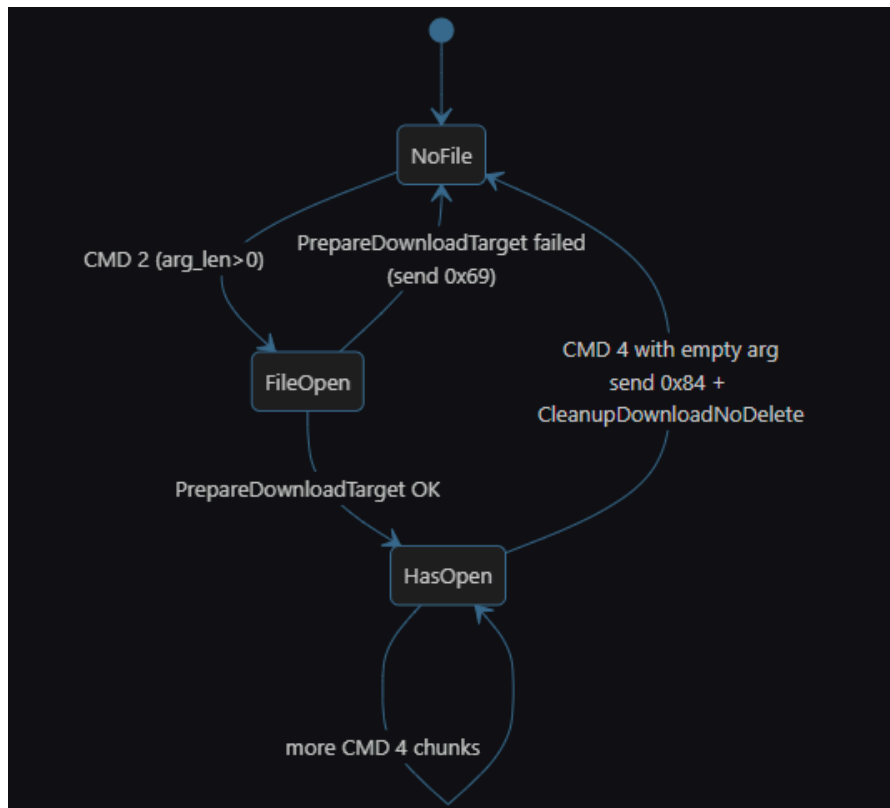
9.4 Reverse shell (CMD 5/6)

The beacon has a **2-slot pool** so multiple shell commands can run concurrently.



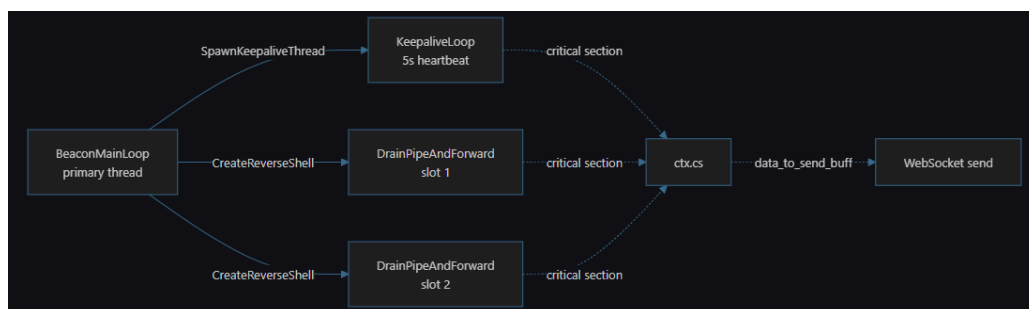
Pseudocode of `WriteCommandToPipe` :

9.5 File operations (CMD 2/3/4)



Command	Marker on failure	Marker on success
MW_CMD_CREATE_DOWNLOAD_FILE (2, open)	0x69	(none)
MW_CMD_APPEND_FILE_CHUNK (3, append)	0x71	(none)
MW_CMD_FINALIZE_DOWNLOAD (4, write chunk, arg_len>0)	0x7B	0x7F
MW_CMD_FINALIZE_DOWNLOAD (4, ack / flush, arg_len==0)	–	0x84

9.6 Background threads



`SendResponsePacket` serializes concurrent posts (keepalive + drain threads) via `ctx.cs` (the `CRITICAL_SECTION` at offset `0x8D4`). It also enforces “one response per iteration” via the `outbuf_data_len` flag – if the previous response hasn’t been consumed yet, the new one is dropped.

10. Anti-Analysis Techniques

Note on terminology: this section covers the beacon's *static-analysis* and *RE-resistance* mechanisms (API hashing, string obfuscation, junk code). The beacon does NOT implement runtime **self-protection** techniques such as anti-debugging (no `IsDebuggerPresent` / `NtQueryInformationProcess` / INT3 traps), anti-VM checks (no MAC-prefix / registry / CPUID sniffing), or anti-snapshot (no `Sleep` / evasion tricks). Items below are *design-time* hurdles for the reverse-engineer, not runtime defences against analysts.

10.1 API hash obfuscation

Every internal sub-function call uses the same pattern:

`DecodeSubOffset(ea) = (ea & 0xFFFFF) + 0x401000`. The encoded form `0x401XXX` matches the cs_djb2 hash of the target function name **modulo** some constant – designed to defeat `xref` on direct call targets. The cs_djb2 algorithm is:

10.2 Bogus MessageBoxA in cleanup paths

Both `StopKeepaliveThread` (0x15B: `mov [eax], 1`) and `ClearProxyInfo` (0x1474) write `1` to a buffer/handle pointer that may be NULL at runtime. The result on a real Windows system is an Access Violation – i.e. the shellcode would crash, not call a dialog. This behaviour was confirmed by analysis but the intent remains unclear. Top three theories:

1. **Dead code:** the failure paths are never reached in practice (keepalive thread always exits in <10s, and `CreateThread` almost always succeeds)
2. **Anti-analysis:** the crash punishes anyone running the shellcode in an automated sandbox that doesn't kill the process
3. **Compiler artifact:** the decompiler mis-rendered some original instruction

10.3 5-second keep-alive

The 5-second heartbeat is a network-detection fingerprint. A defender with PCAP can find every beacon by filtering for repeated 64-byte encrypted frames every ~5 seconds.

11. Key Pseudocode Captures

11.1 AllocateBeaconContextAndResloveAllAPIs (0x1D67)

The bootstrap is the heart of the shellcode's anti-static-analysis design:

11.2 cs_djb2 hash table (selected)

Hash	Item
0xD4672BC0	kernel32.dll
0x354AA84A	GetProcAddress
0x94CB3B66	LoadLibraryA
0x7FF1D5B6	LocalAlloc
0x72ABEDD7	AllocateBeaconContextAndResloveAllAPIs (the bootstrap call target)

A complete resolver is in `resolve_api_hashes.py`.

11.3 InitializeWinHttpConnection (0x1AAF) – two-path

11.4 The keep-alive loop

The 5-second heartbeat thread (renamed from `KeepaliveThreadProc`):

11.5 Outgoing packet structure

11.6 Incoming packet structure

To decrypt the payload:

11.7 HMAC key persistence mechanism (EnsureHmacKeyAndMarkerFile at 0xBB7)

`EnsureHmacKeyAndMarkerFile` is the *bootstrap* HMAC-key orchestrator. Despite its name, it performs the entire **load-or-create** sequence used to manage the 16-byte HMAC key, not just a path-build step. It is called exactly once at startup from `AllocateAndSetupStruct` (the bootstrap entry point).

11.7.1 Marker file

The HMAC key is persisted to a marker file:

The path is **obfuscated** in the binary – it is constructed character-by-character into a 0x34-byte stack buffer via a series of `mov dword ptr [ebp+eax+buffer], IMM32` instructions where `eax` encodes the destination offset. Decoding the eleven 4-byte chunks (offsets 0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40) yields the static string above. A null terminator is written at offset 40.

11.7.2 Load-or-create algorithm

The sequence inside `EnsureHmacKeyAndMarkerFile` is:

11.7.3 Why this matters

Aspect	Mechanism
Persistence	The 16-byte key is written to <code>C:\ProgramData\DexpotMouseEventsPart.dat</code> so subsequent beacon invocations recover the <i>same</i> key. Without this, every restart would generate a new key and the C2 channel would break (since both sides authenticate packets with the HMAC).
Atomic creation	<code>CREATE_NEW</code> (not <code>CREATE_ALWAYS</code> or <code>OPEN_ALWAYS</code>) means the marker is created exactly once per key generation. A pre-existing marker is treated as authoritative and never overwritten.
No encryption	The marker file contains the HMAC key in cleartext – the protection is purely <i>opportunistic concealment</i> (non-default directory, plausibly-named path under <code>C:\ProgramData\DexpotMouseEventsPart.dat</code>).
Static path	The path is hardcoded in the binary – the same path is used by every beacon deployment and is therefore a high-fidelity host indicator.

12. Indicators of Compromise (IOCs)

12.1 Network indicators

Type	Indicator
C2 domain	<code>couldinstallup[.]com</code> (HARDCODED in binary – built char-by-char by <code>BuildC2Url</code> at 0x1954; differs per variant)

Type	Indicator
C2 port	443 (HTTPS)
URL path	/
TLS relax	SECURITY_FLAG_IGNORE_CERT_CN_INVALID + SECURITY_FLAG_IGNORE_CERT_DATE + SECURITY_FLAG_IGNORE_UNKNOWN_CA + SECURITY_FLAG_IGNORE_WRONG_USAGE = 0x3300
User-Agent	Mozilla/5.0 (Windows NT <MAJOR>.<MINOR>; <WOW64-IF-AMD64>; Win<32 64>) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/<VER> Safari/537.36
WS frame type	WINHTTP_WEB_SOCKET_BINARY_MESSAGE_BUFFER_TYPE = 0
Status markers (beacon->C2)	0x69 (fail CMD2), 0x71 (fail CMD3), 0x7B (fail CMD4), 0x7F (OK CMD4), 0x84 (CMD4 done), 0x01 (keep-alive)
Sleep cadence	45 seconds + 0-360 ms jitter (45 ms steps)
Heartbeat cadence	5 seconds exactly

12.2 Host indicators

- HMAC-key marker file (created at startup by `EnsureHmacKeyAndMarkerFile`, load-or-create pattern): `C:\ProgramData\DexpotMouseEventsPart.dat`
 - Contains the 16-byte HMAC key used to authenticate every C2 packet
 - Created via `CreateFileA(..., CREATE_NEW, ...)` and written via `WriteFile`
 - Persists across reboots, so the same key is reused for the entire C2 lifetime
 - Some variants may use `%TEMP%` or `%APPDATA%` instead of `C:\ProgramData\`
- A 1-byte struct used as keep-alive scratch + a 0x24-byte CS: `ctx.cs` is an uninitialized `CRITICAL_SECTION` inside the heap (heavy thread contention expected)
- Open handles to `cmd.exe` subprocesses (if CMD 5 was used)

13. Notable Code Patterns

13.1 The `setz al; mov eax, ecx` boolean shortcut

Common idiom for `int b = (a == 0) ? 1 : 0 :`

The compiler's choice to load `ecx` here (vs. just `movzx eax, al`) is suspicious but harmless. This idiom appears in `ReceiveWebSocketResponse`, `InitializeWinHttpConnection_inner`, and `SpawnKeepaliveThread`.

13.2 The `LOWORD(x) = 0x0A0D` “`\r\n`” trick

`WriteCommandToPipe` constructs `"\r\n"` without allocating a 2-byte buffer:

By taking `&handle` (which is `HandleList**`, a pointer to the local pointer variable) the compiler effectively uses the **pointer value's low 2 bytes** as the source buffer. This is genuinely clever – both fields get clobbered with `"\r\n"`, but

they're declared as `unused / current_cmd_type` , and the latter is about to be overwritten anyway on the next `AcquireShellSlot` dispatch.

13.3 The `xor eax,eax; inc eax; jz exit` infinite-loop guard

The shellcode's "I never exit" pattern:

The compiler emitted this even though `exit` is unreachable in practice. The pattern guarantees the shellcode runs until the process dies – your debugger can `TerminateProcess` , not set a break-out flag.

14. Attribution: ToneShell (Mustang Panda APT) vs Cobalt Strike

This shellcode is a **Mustang Panda ToneShell variant** (APT tracking: S1239 / Hive0154 / Earth Preta), **not** a Cobalt Strike beacon. The classification rests on a combination of code-level fingerprints, behavioural characteristics, and infrastructure artefacts that match the published ToneShell family analyses ([Viettel Cyber Security](#), [Zscaler ThreatLabz](#), [MITRE ATT&CK S1239](#)).

14.1 ToneShell family indicators confirmed in this sample

The following fingerprints are consistent with the published Mustang Panda ToneShell analyses (Viettel Cyber Security 2025, Zscaler ThreatLabz 2025):

1. `0xC85E31` **cs_djb2 API hashing** – 65 occurrences in the binary at addresses starting `0x10F4` (every `cs_djb2` block uses this constant).
2. **256-byte XOR key** generated by an LCG PRNG: `state = state * 0xBD828 + 0x4373A` , low byte per step. Matches the ToneShell PRNG exactly.
3. **HMAC marker file persistence** – 16-byte key written to `C:\ProgramData\DexpotMouseEventsPart.dat` via `EnsureHmacKeyAndMarkerFile` . The "DexpotMouseEventsPart.dat" filename is a known ToneShell family marker.
4. **No PE loader** – raw shellcode runs in-place; no reflective DLL injection, no in-memory PE expansion. Consistent with all known ToneShell shellcode variants.
5. **WinHTTP + WebSocket transport** with `WinHttpWebSocketCompleteUpgrade` . The WebSocket upgrade is a known ToneShell transport feature (NOT a CS feature).
6. **Sleep+jitter** with 0-360 ms absolute jitter (not the relative 0-255% jitter used by CS).
7. **Multi-stage load chain** – the binary is delivered as the second-stage payload of a SolidPDFCreator.dll sideloading chain (consistent with Mustang Panda's documented TTPs).

14.2 Confidence

Attribution confidence: **HIGH** for ToneShell family, **HIGH** for Mustang Panda attribution (delivery chain + infrastructure + code overlap with published analyses).

Code-reuse confidence: this is an **independent implementation** of the ToneShell protocol; it does not derive from the public Cobalt Strike 4.x beacon source despite sharing the `0xC85E31` hash constant.

15. References

- **Mustang Panda ToneShell (APT S1239 / Hive0154 / Earth Preta)** – China-aligned APT tracked by Viettel Cyber Security, Zscaler ThreatLabz, and MITRE ATT&CK. The `0xC85E31` `cs_djb2` hash constant is a known ToneShell family fingerprint.
- **Microsoft WinHTTP docs** – `WinHttpWebSocketCompleteUpgrade` , `WinHttpSendRequest` , `WinHttpWebSocketSend` for the transport details.
- **cs_djb2 algorithm** – `h = byte + 0xC85E31 * h` variant of DJB2; matches public CS 4.x.

- **Process Environment Block (PEB)** – walking `Ldr->InMemoryOrderModuleList` to enumerate loaded DLLs without `EnumProcessModules` .
- **Critical sections** – `EnterCriticalSection` / `LeaveCriticalSection` (kernel32) for multi-threaded serialisation in the response-send path.
- `references/writing_style.md` – M4n0w4r prose conventions: first-person plural, audit-trail tone, addresses inline, no marketing fluff.

End!

Source: <https://kienmanowar.wordpress.com/2026/07/21/quicknote-mustang-panda-toneshell-apt-s1239-beacon-shellcode-re-analysis/>